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

Μελέτη τεχνολογιών εικονικοποίησης υποβοηθούμενης από το υλικό και μέθοδοι απομόνωσης διεργασιών και εκτέλεσης τους στο υλικό της εικονικοποίησης.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
(σε συνεργασία με το Ι.Τ.Ε./ Ι.Π, εργαστήριο CARV)

Αστρινός Γ. Δαμιανάκης
Α.Μ.:03105022

Επιβλέπων : Νεκτάριος Κοζύρης, Καθηγητής Ε.Μ.Π.
Συνεπιβλέπων από το Ι.Τ.Ε.: Μανόλης Μαραζάκης, Ε.Λ.Ε. Β', Ι.Τ.Ε.

Αθήνα, Ιούλιος 2017

- *EuroServer* project, funded by the European Commission under FP7/ICT grant agreement 610456

Μελέτη τεχνολογιών εικονικοποίησης υποβοηθούμενης από το υλικό και μέθοδοι απομόνωσης διεργασιών και εκτέλεσης τους στο υλικό της εικονικοποίησης.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
(σε συνεργασία με το Εθνικό Μετσόβιο Πολυτεχνείο)

Αστρινός Γ. Δαμιανάκης

Επιβλέποντες:
Μανόλης Μαραζάκης, Ε.Λ.Ε. Β', Ι.Τ.Ε.
Μανόλης Κατεβαίνης, Καθηγητής Πανεπιστημίου Κρήτης.

Συνεπιβλέπων από το Ε.Μ.Π.:
Νεκτάριος Κοζύρης, Καθηγητής Ε.Μ.Π.

Ηράκλειο, Ιούλιος 2017
Ίδρυμα Τεχνολογίας και Έρευνας (FORTH)
Ινστιτούτο Πληροφορικής (ICS)
Εργαστήριο Αρχιτεκτονικής Υπολογιστών και Συστημάτων VLSI (C.AR.V)

This work was performed at the Computer Architecture and VLSI Systems (CARV) Laboratory of the Institute of Computer Science (ICS) of the Foundation for Research and Technology – Hellas (FORTH), and was financially supported by a FORTH-ICS scholarship, including funding by the European Union 7th Framework Programme, under the *EuroServer* project, FP7/ICT grant agreement 610456.

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

Μελέτη τεχνολογιών εικονικοποίησης υποβοηθούμενης από το υλικό και μέθοδοι απομόνωσης διεργασιών και εκτέλεσης τους στο υλικό της εικονικοποίησης.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αστρινός Γ. Δαμιανάκης
Α.Μ.:03105022

Επιβλέπων : Νεκτάριος Κοζύρης, Καθηγητής Ε.Μ.Π.
Συνεπιβλέπων από το Ι.Τ.Ε.: Μανόλης Μαραζάκης, Ε.Λ.Ε. Β', Ι.Τ.Ε.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ...../...../ 2017.

.....	.....	.....
Νεκτάριος Κοζύρης	Μανόλης Κατεβαίνης	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π.	Καθηγητής Πανεπιστημίου Κρήτης	Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2017

.....
Αστρινός Γ. Δαμιανάκης

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

Περίληψη:

Αποτελεί μια γενική παραδοχή πως στη σύγχρονη επιστήμη υπολογιστών έχουν κυριαρχήσει οι τεχνολογίες των συγκεντρωμένων υποδομών και των λύσεων IAAS και η παροχή cloud υπηρεσιών σε μεγάλους η και μικρούς “παίκτες” του υπολογιστικού και πληροφοριακού πεδίου. Αναπόσπαστο τμήμα στην ανάπτυξη υπηρεσιών cloud είναι η απομόνωση των διεργασιών τόσο για λόγους ασφαλείας όσο και για λόγους απόδοσης. Η χρήση εικονικών μηχανών, ιδιαίτερα με τέτοιο τρόπο ώστε να αξιοποιείται πλήρως το υλικό εικονικοποίησης που διαθέτουν οι σύγχρονοι επεξεργαστές, όπως οι Intel x86_64 και οι ARMv8.1, αποτελεί έναν ελπιδοφόρο τρόπο προσέγγισης τόσο για ερευνητικούς σκοπους όσο και για εμπορικές εφαρμογές.

Λέξεις κλειδιά:
Σύγχρονοι επεξεργαστές, Εικονικοποίηση, Απομόνωση διεργασιών,Υπολογιστικό Νέφος IaaS, KVM, OSv unikernel

Abstract:

It is generally accepted that, in modern computer science, converged infrastructure technologies and IAAS solutions have ruled the field of cloud services providement to large or smaller information and computation technology partners. A strong cornerstone for the development of cloud application services is the isolation of processes for security and performance reasons. The use of virtual machines is a hopeful solution not only for research projects but also for commercial applications, especially in the way that the virtualization hardware of modern processor architectures like Intel x86_64 and ARMv8.1 is totally utilized.

Key words:
Modern processors, Virtualization, Process isolation, Cloud, IaaS, KVM, OSv unikernel

Ευχαριστίες:

Τον Ιούλιο του 2016, έχοντας ολοκληρώσει κατά πολύ τις υποχρεώσεις των μαθημάτων στο πλαίσιο των σπουδών μου στη σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π. και έχοντας μετακομίσει ήδη στην γενέτειρά μου, το Ηράκλειο, αποφάσισα να χτυπήσω τις πόρτες του εργαστηρίου Αρχιτεκτονικής Υπολογιστών και Συστημάτων VLSI (C.AR.V.) του Ι.Π./Ι.Τ.Ε. Κίνητρο γι' αυτή την ενέργεια αποτέλεσε το ενδιαφέρον μου για το hardware. Η ενασχόληση με αυτό το αντικείμενο της επιστήμης των υπολογιστών έδινε τη δυνατότητα εκπόνησης μίας πολύ ενδιαφέρουσας διπλωματικής αλλά και την προοπτική μιας συναρπαστικής επαγγελματικής σταδιοδρομίας.

Ολόκληρο το μέρος της διπλωματικής εκπονήθηκε στο Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Ερευνας του Ηρακλείου Κρήτης, και συγκεκριμένα στο εργαστήριο Αρχιτεκτονικής Υπολογιστών και Συστημάτων VLSI, στο πλαίσιο του ευρωπαικού έργου EuroServer, υπό τον κύριο καθηγητή του Πανεπιστημίου Κρήτης κύριο Μανόλη Κατεβαίνη, και υπό την επίβλεψη του Ειδικού Λειτουργικού Επιστήμονα Β' του Ι.Τ.Ε. κυρίου Μανόλη Μαραζάκη. Από πλευράς Εθνικού Μετσόβιου Πολυτεχνείου, η παρούσα διπλωματική εργασία εκπονήθηκε υπό το Εργαστήριο Υπολογιστικών Συστημάτων της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου, υπό την επίβλεψη του κυρίου καθηγητή κυρίου Νεκτάριου Κοζύρη. Πολλές ευχαριστίες απευθύνονται στους παραπάνω καθηγητές τόσο για την αποδοχή και εμπιστοσύνη που μου έδειξαν οσο και για την οργάνωση της εκπόνησης αυτής της διπλωματικής εργασίας.

Οι θερμότερες ευχαριστίες απευθύνονται στον Ειδικό Λειτουργικό Επιστήμονα Β' του Ι.Τ.Ε. κύριο Μανόλη Μαραζάκη για τη συνεχή του καθοδήγηση κατά τη 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. Επίσης πολλές ευχαριστίες στους διδακτορικούς φοιτητές κ. Αναστάσιο Παπαγιάννη (Ι.Τ.Ε.) και κ. Στέφανο Γεράγγελο (Ε.Μ.Π.) για τις πολύτιμες γνώσεις που μου μετέδωσαν και την ακατάπαυστη βοήθεια που μου παρείχαν, καθώς και στον μεταδιδακτορικό Ερευνητή του Ι.Τ.Ε. κύριο Μανώλη Πλουμίδη.

Τέλος, θέλω να ευχαριστήσω την οικογένεια μου, θερμά τους γονείς μου Ελένη και Γιώργο και την αδερφή μου Ειρήνη, τους συγγενείς μου, τους φίλους μου και όλους όσοι στάθηκαν δίπλα μου και πίστεψαν στη προσπάθεια να ολοκληρώσω τις σπουδές μου όλα αυτά τα χρόνια. Ιδιαίτερα όμως θέλω να ευχαριστήσω την σύζυγό μου Καλλιόπη, για την απίστευτη υπομονή και υποστήριξη που μου παρείχε στις δύσκολες στιγμές, καθώς και για το ότι έφερε στον κόσμο τη νεογέννητη μου κόρη Ελένη, την κινητήριο δύναμη της εργασίας αυτής, την οποία ο μπαμπάς ευχαριστεί από το παρελθόν και στην οποία, πρώτα ο Θεός μαζί με τα μελλοντικά της αδέρφια, η παρούσα διπλωματική εργασία με όλη μου την ψυχή αφιερώνεται. Χωρίς τη συμβολή όλων τους η ολοκλήρωση της διπλωματικής εργασίας αυτής δεν θα ήταν εφικτή.

Αστρινός Γ. Δαμιανάκης

Περιεχόμενα:

Κατάλογος Σχημάτων και Πινάκων:

Μέρος Α’

Μελέτη τεχνολογιών εικονικοποίησης υποβοηθούμενης από το υλικό

ΚΕΦΑΛΑΙΟ 1ο

Υπολογιστικά Συστήματα

1.1 Εισαγωγή

Στο Μέρος Α’ της εργασίας αυτής θα ασχοληθούμε με τεχνολογίες εικονικοποίησης υποβοηθούμενης από το υλικό των επεξεργαστών. Αρχικά, στο κεφάλαιο αυτό γίνεται μια εισαγωγή στις έννοιες των υπολογιστικών συστημάτων, του διαδικτύου, της συγκεντρωμένης υποδομής, του υπολογιστικού νέφους και της εικονικοποίησης. Στα επόμενα αναλύονται η έννοια και τα είδη της εικονικοποίησης, οι τεχνολογίες εικονικοποίησης τριών σύγχρονων αρχιτεκτονικών, και τα πιο διαδεδομένα συστήματα λογισμικού για την εικονικοποίηση.

Ο **ηλεκτρονικός υπολογιστής** είναι μια μηχανή κατασκευασμένη κυρίως από ψηφιακά ηλεκτρονικά κυκλώματα και δευτερευόντως από ηλεκτρικά και μηχανικά συστήματα, και έχει ως σκοπό να επεξεργάζεται πληροφορίες. Ο ηλεκτρονικός υπολογιστής είναι ένα αυτοματοποιημένο, ηλεκτρονικό, ψηφιακό επαναπρογραμματιζόμενο σύστημα γενικής χρήσης το οποίο μπορεί να επεξεργάζεται δεδομένα βάσει ενός συνόλου προκαθορισμένων οδηγιών, των εντολών που συνολικά ονομάζονται πρόγραμμα.

Κάθε υπολογιστικό σύστημα, όσο μεγάλο ή μικρό κι αν είναι, αποτελείται από το υλικό μέρος (hardware) και το λογισμικό (software). Τα βασικά στοιχεία του υλικού μέρους του υπολογιστή είναι η κεντρική μονάδα επεξεργασίας (ΚΜΕ, αγγλ. CPU, Central Prossesing Unit), η κεντρική μνήμη (RAM & ROM-BIOS), οι μονάδες εισόδου - εξόδου (πληκτρολόγιο, ποντίκι, οθόνη κ.α.), οι εσωτερικές (ή εξωτερικές) μονάδες ανάγνωσης και αποθήκευσης δεδομένων όπως σκληρός δίσκος, DVD, SSD (Solid State Drive) και οι περιφερειακές συσκευές όπως εκτυπωτής, σαρωτής, μόντεμ κ.α.).

Υπάρχουν διάφοροι τύποι υπολογιστών οι οποίοι διαφέρουν κατά το μέγεθος, τις δυνατότητες (επεξεργαστική ισχύς) και την αρχιτεκτονική τους, δηλαδή τον τρόπο που τα βασικά τους μέρη συνδέονται και συνεργάζονται μεταξύ τους. Στην πιο διαδεδομένη κατηγορία υπολογιστών ανήκουν οι μικροϋπολογιστές. Στους μικροϋπολογιστές τα βασικά εξαρτήματα, όπως ο επεξεργαστής, η μνήμη κ.ά., βρίσκονται τοποθετημένα σ' ένα τυπωμένο κύκλωμα που ονομάζεται μητρική κάρτα (αγγλ. Motherboard ή MoBo). Εκτός από τον επεξεργαστή και τη μνήμη, πάνω στη μητρική βρίσκονται οι θέσεις επέκτασης στις οποίες τοποθετούνται οι διάφορες κάρτες, γραφικών, ήχου κ.λπ.). Στη μητρική επίσης βρίσκονται υποδοχές για τη σύνδεση διαφόρων άλλων συσκευών (όπως ο σκληρός δίσκος, η οπτική μονάδα ανάγνωσης DVD, card reader κλπ), ή και προς επέκταση των ήδη εγκατεστημένων.

Το λογισμικό του υπολογιστή αποτελείται από τα απαραίτητα προγράμματα που δίνουν τις κατάλληλες εντολές, για να λειτουργεί το υλικό μέρος. Συνίσταται δε από το λειτουργικό σύστημα (το βασικό πρόγραμμα για τη λειτουργία του Η/Υ καθώς και για την επικοινωνία του με τον άνθρωπο) και το λογισμικό εφαρμογών (πακέτα εφαρμογών, γλώσσες προγραμματισμού, εκπαιδευτικό λογισμικό, προγράμματα – εργαλεία κ.α.).

1.2 Διαδίκτυο

Το **Διαδίκτυο** (αγγλ. Internet) είναι ένα παγκόσμιο σύστημα διασυνδεδεμένων δικτύων υπολογιστών, οι οποίοι χρησιμοποιούν καθιερωμένη ομάδα πρωτοκόλλων, η οποία συχνά αποκαλείται "TCP/IP" (αν και αυτή δεν χρησιμοποιείται από όλες τις υπηρεσίες του Διαδικτύου) για να εξυπηρετεί εκατομμύρια χρήστες καθημερινά σε ολόκληρο τον κόσμο. Οι διασυνδεδεμένοι ηλεκτρονικοί υπολογιστές ανά τον κόσμο, οι οποίοι βρίσκονται σε ένα κοινό δίκτυο επικοινωνίας, ανταλλάσσουν μηνύματα (πακέτα) με τη χρήση διαφόρων πρωτοκόλλων (τυποποιημένοι κανόνες επικοινωνίας), τα οποία υλοποιούνται σε επίπεδο υλικού και λογισμικού. Το κοινό αυτό δίκτυο καλείται Διαδίκτυο.

1.2.1 Το Διαδίκτυο και η Επικοινωνία

Με την εμφάνιση οποιουδήποτε νέου μέσου, ο τομέας της επικοινωνίας αναμφισβήτητα επηρεάζεται. Η επίδραση αυτή πηγάζει κυρίως από την τεχνολογία του νέου μέσου. Σε τι επίπεδο μπορεί η τεχνολογία του διαδικτύου να αλλάξει τον τρόπο με τον οποίο επικοινωνούν και πληροφορούνται μαζικά οι άνθρωποι; Υπάρχουν διαφορετικές και αντικρουόμενες προσεγγίσεις πάνω στο θέμα.

Σύμφωνα με την προσέγγιση της "ιντερνετοφιλίας" (ένα μείγμα κλασικής "πλουραλιστικής" προσέγγισης και τεχνολογικού "ντετερμινισμού"), το Διαδίκτυο, αλλά και η ψηφιακή τεχνολογία γενικότερα, έχουν την ικανότητα να δημιουργούν "εικονικούς χώρους", "εικονικές κοινότητες", όπου παύουν να υφίστανται οι κοινωνικές και πολιτιστικές διαχωριστικές γραμμές που υπάρχουν στον πραγματικό κόσμο και που τα παραδοσιακά μέσα επικοινωνίας αδυνατούν να ξεπεράσουν εύκολα. Η επικοινωνία μέσω του διαδικτύου καθίσταται άμεση και αμφίδρομη. Δίνεται η δυνατότητα σε κάθε χρήστη ηλεκτρονικού υπολογιστή συνδεδεμένου στο Διαδίκτυο, να πληροφορηθεί αλλά και να πληροφορήσει ανταλλάσσοντας απόψεις μέσω ενός πιο συμμετοχικού και λιγότερο ελεγχομένου διαύλου επικοινωνίας. Οι χρήστες αποκτούν ολοένα και περισσότερο την ιδιότητα του παγκοσμίου πολίτη. Υπάρχει έντονη τάση, ήδη από την αρχή της εμφάνισης του διαδικτύου, να θεωρείται ένα άκρως δημοκρατικό μέσο μαζικής επικοινωνίας, το οποίο αποδιαμεσολαβεί την επικοινωνία και καθιστά ισχυρότερο τον μέσο άνθρωπο, καθώς δίνει στον τελευταίο τη δυνατότητα πρόσβασης σε μεγάλο όγκο πληροφοριών συγκεντρωμένων σε ένα "χώρο" και την δυνατότητα της προσωπικής επιλογής των πληροφοριών αυτών. Συνεπώς, η βασική θέση της προσέγγισης αυτής είναι ότι το Διαδίκτυο θα εκδημοκρατίσει την κοινωνία με το να βελτιώσει την επικοινωνία καταργώντας την ανάγκη για διαμεσολάβηση.

1.2.2 Η τεχνολογία του Διαδικτύου

Το Διαδίκτυο είναι επικοινωνιακό δίκτυο που επιτρέπει την ανταλλαγή δεδομένων μεταξύ οποιουδήποτε διασυνδεδεμένου υπολογιστή. Η τεχνολογία του είναι κυρίως βασισμένη στην διασύνδεση επιμέρους δικτύων ανά τον κόσμο και σε πολυάριθμα πρωτόκολλα επικοινωνίας. Στην πιο εξειδικευμένη και περισσότερο χρησιμοποιούμενη μορφή του, με τον όρο Διαδίκτυο περιγράφεται το παγκόσμιο πλέγμα διασυνδεδεμένων υπολογιστών και των υπηρεσιών και πληροφοριών που παρέχει στους χρήστες του. Το Διαδίκτυο χρησιμοποιεί μεταγωγή πακέτων και τη στοίβα πρωτοκόλλων. Σήμερα, ο όρος διαδίκτυο κατέληξε στο να αναφέρεται στο παγκόσμιο αυτό δίκτυο. Για να ξεχωρίζει, το παγκόσμιο αυτό δίκτυο γράφεται με κεφαλαίο το αρχικό "Δ". Η τεχνική της διασύνδεσης δικτύων μέσω μεταγωγής πακέτων και της στοίβας πρωτοκόλλων ονομάζεται Διαδικτύωση.

1.3 Συγκεντρωμένη Υποδομή - Converged infrastructure

Η συγκεντρωμένη υποδομή (Σχήμα 1.1) λειτουργεί με την ομαδοποίηση πολλαπλών στοιχείων τεχνολογίας πληροφορικής (IT) σε ένα ενιαίο, βελτιστοποιημένο πακέτο υπολογιστών. Τα συστατικά στοιχεία μιας συγκλίνουσας υποδομής μπορεί να περιλαμβάνουν διακομιστές, συσκευές αποθήκευσης δεδομένων, εξοπλισμό δικτύωσης και λογισμικό για διαχείριση υποδομών πληροφορικής, αυτοματοποίηση και “ενορχήστρωση”.

Οι οργανισμοί πληροφορικής χρησιμοποιούν συγκλινόμενη υποδομή για να συγκεντρώσουν τη διαχείριση των πόρων πληροφορικής, να ενοποιήσουν τα συστήματα, να αυξήσουν τα ποσοστά χρησιμοποίησης των πόρων και να μειώσουν το κόστος. Οι συγκλινόμενες υποδομές ενθαρρύνουν αυτούς τους στόχους με την υλοποίηση ομάδων υπολογιστών δηλαδή πόρων αποθήκευσης και δικτύωσης που μπορούν να μοιράζονται σε πολλαπλές εφαρμογές και να διαχειρίζονται με συλλογικό τρόπο χρησιμοποιώντας διεργασίες που καθοδηγούνται από την εκάστοτε πολιτική λειτουργίας.

Οι πωλητές τεχνολογίας πληροφορικής και οι αναλυτές της βιομηχανίας πληροφορικής χρησιμοποιούν διάφορους όρους για να περιγράψουν την έννοια μιας συγκεντρωμένης υποδομής. Αυτά περιλαμβάνουν το "συγκλίνον σύστημα", το "ενοποιημένο computing", το "computing based on fabric" και τη "δυναμική υποδομή".

Σχήμα 1.1: *Συγκεντρωμένη υποδομή*

1.3.1 Η εξέλιξη των κέντρων δεδομένων

Ιστορικά, για να συμβαδίζουν με την ανάπτυξη των επιχειρηματικών εφαρμογών και των terabytes των δεδομένων που παράγουν, οι πόροι πληροφορικής αναπτύχθηκαν με σιλό. Ένα σύνολο πόρων αφιερώνεται σε μια συγκεκριμένη τεχνολογία πληροφορικής, επιχειρηματική εφαρμογή ή γραμμή επιχειρήσεων. Αυτοί οι πόροι υποστηρίζουν ένα συγκεκριμένο σύνολο υποθέσεων και δεν μπορούν να βελτιστοποιηθούν ή να αναμορφωθούν για να υποστηρίξουν διαφορετικά φορτία χρήσης.

Ο πολλαπλασιασμός της εξάπλωσης της τεχνολογίας πληροφορικής στα κέντρα δεδομένων έχει συμβάλει στην αύξηση των λειτουργικών εξόδων, στη μείωση της παραγωγικότητας και στην άμβλυνση της ευελιξίας. Η συντήρηση και η λειτουργία μπορούν να καταναλώσουν τα δύο τρίτα του τεχνολογικού προϋπολογισμού ενός οργανισμού, σύμφωνα με μια έρευνα του 2009 του InformationWeek από στελέχη 500 εταιρειών με ετήσια έσοδα πάνω από 250 εκατομμύρια δολάρια. Αυτό αφήνει μόνο το ένα τρίτο του προϋπολογισμού για νέες πρωτοβουλίες τεχνολογίας πληροφορικής. Αυτός ο λόγος εμποδίζει την τεχνολογία πληροφορικής να υποστηρίζει νέες επιχειρηματικές πρωτοβουλίες ή να ανταποκρίνεται σε πραγματικές απαιτήσεις εφαρμογής.

Μια συγκλίνουσα υποδομή αντιμετωπίζει το πρόβλημα των αρχιτεκτονικών τύπου σιλό και της εξάπλωσης της πληροφορικής, συγκεντρώνοντας και μοιράζοντας πόρους πληροφορικής. Αντί να αφιερώνει ένα σύνολο πόρων σε μια συγκεκριμένη τεχνολογία υπολογιστών, εφαρμογή ή γραμμή επιχειρήσεων, η συγκλίνουσα υποδομή δημιουργεί ένα σύνολο εικονικοποιημένων servers, δηλαδή χωρητικότητα αποθήκευσης και ικανότητα δικτύωσης που μοιράζεται από πολλαπλές εφαρμογές και γραμμές επιχειρήσεων.

1.3.2 Οφέλη από τη συγκλίνουσα υποδομή

Οι συγκλινόμενες υποδομές παρέχουν τεχνική και επιχειρηματική απόδοση, σύμφωνα με τους ερευνητές και τους παρατηρητές της βιομηχανίας. Αυτά τα οφέλη προέρχονται εν μέρει από την προ-ενοποίηση των τεχνολογικών στοιχείων, τη συγκέντρωση των πόρων πληροφορικής και την αυτοματοποίηση των διαδικασιών πληροφορικής. Η συνεργαζόμενη υποδομή συμβάλλει περαιτέρω στη δημιουργία αποδοτικών κέντρων δεδομένων, ενισχύοντας την ικανότητα των συστημάτων υπολογιστικού νέφους να διαχειρίζονται τεράστια σύνολα δεδομένων, χρησιμοποιώντας μόνο ένα ενιαίο ολοκληρωμένο σύστημα διαχείρισης τεχνολογίας πληροφορικής.

Οι συνεργάτες της Forrester Research, Robert Whiteley, γράφουν στο περιοδικό CIO ότι οι συγκλίνουσες υποδομές, που συνδυάζουν διακομιστές, αποθηκευτικούς χώρους και

δίκτυα σε ένα ενιαίο πλαίσιο, συμβάλλουν στη μετατροπή των οικονομικών δεδομένων λειτουργίας των κέντρων δεδομένων, επιταχύνοντας έτσι τη μετάβαση στην αποθήκευση μέσω IP (IPStorage), ώστε να μπορούν να φτιαχτουν υποδομές που θα είναι "έτοιμες για σύννεφο". Ο συνδυασμός αποθήκευσης και υπολογισμού σε μια ενιαία οντότητα είναι γνωστός ως σύγκεντρωμένη αποθήκευση.

Η μειωμένη πολυπλοκότητα, μέσω της χρήσης προ-ενοποιημένου υλικού με εργαλεία διαχείρισης εικονικοποίησης και αυτοματοποίησης, είναι μια άλλη σημαντική παράμετρος αξίας για τη συγκλίνουσα υποδομή, όπως αναφέρεται σε μια μελέτη IDC.

Τον Απρίλιο του 2012, η open source εταιρεία αναλυτών Wikibon κυκλοφόρησε την πρώτη πρόβλεψη της αγοράς για τη σύγκλιση υποδομών. Με συνολική διαθέσιμη αγορά ύψους 402 δολ. μέχρι το 2017, περίπου τα 2/3 των υποδομών που υποστηρίζουν επιχειρησιακές εφαρμογές θα συσκευάζονται σε κάποιον τύπο συγκλινόμενης λύσης έως το 2017.

Το InformationWeek υπογράμμισε την υπόσχεση δύο μακροπρόθεσμων πλεονεκτημάτων μιας ενοποιημένης υποδομής για τα κέντρα δεδομένων:

-Χαμηλότερο κόστος ως αποτέλεσμα των παρακάτω δύο:

-Χαμηλότερες κεφαλαιουχικές δαπάνες λόγω υψηλότερης χρήσης, λιγότερης καλωδίωσης και λιγότερων συνδέσεων δικτύου.

-Χαμηλότερο λειτουργικό κόστος που προκύπτει από τη μείωση της εργασίας μέσω της αυτοματοποιημένης διαχείρισης του κέντρου δεδομένων και των ενοποιημένων ομάδων υποδομής για τη διαχείριση της αποθήκευσης και της δικτύωσης.

-Αύξηση της ευελιξίας του IT από:

-Εικονικοποίηση της IP και Fibre Channel δικτύωσης αποθήκευσης.

-Επίτρεψη της διαχείρισης με μία κονσόλα.

Τα κέντρα δεδομένων σε όλο τον κόσμο φτάνουν στα όρια τους σε σχέση με την ισχύ, την ψύξη και το χώρο. Ταυτόχρονα, οι κεφαλαιακοί περιορισμοί απαιτούν από τους οργανισμούς να επανεξετάσουν τη στρατηγική λειτουργίας και ανάπτυξης των κέντρων δεδομένων. Η συγκλινόμενη υποδομή προσφέρει μια λύση σε αυτές τις προκλήσεις.

1.3.3 Συγκεντρωμένη υποδομή και υπολογιστικό νέφος

Η ενσωματωμένη υποδομή μπορεί να χρησιμεύσει ως πλατφόρμα ενεργοποίησης για ιδιωτικές και δημόσιες υπηρεσίες cloud computing, όπως οι υπηρεσίες: υποδομή ως

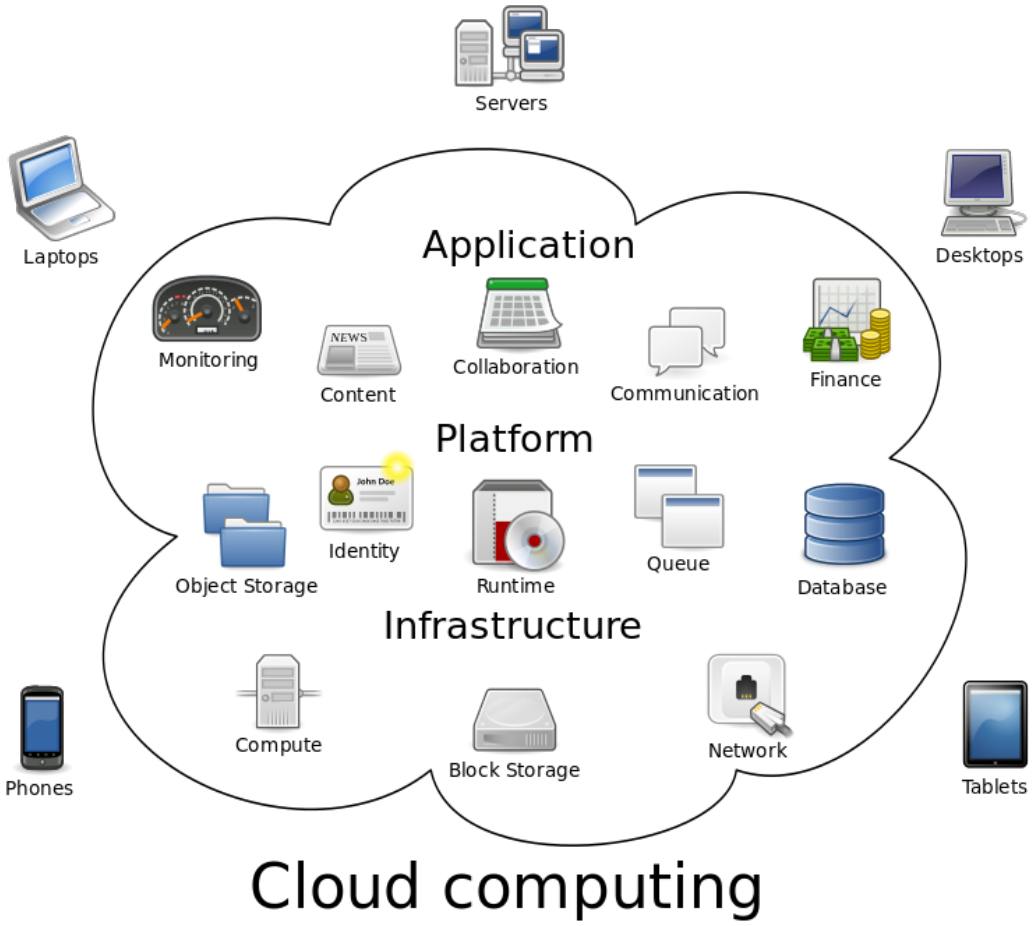
υπηρεσία (IaaS), η πλατφόρμα ως υπηρεσία (PaaS) και οι προσφορές λογισμικού ως υπηρεσία (SaaS).

Πολλά χαρακτηριστικά καθιστούν τη συγκλίνουσα υποδομή κατάλληλη για τις εφαρμογές του cloud. Αυτά περιλαμβάνουν τη δυνατότητα συγκέντρωσης πόρων πληροφορικής, την αυτοματοποίηση της παροχής πόρων και την γρήγορη κλιμάκωση ή μείωση της χωρητικότητας πόρων για την ταχεία κάλυψη των αναγκών του δυναμικού φόρτου εργασίας των υπολογιστών.

1.4 Υπολογιστικό Νέφος - Cloud Computing

Το υπολογιστικό νέφος (Σχήμα 1.2) είναι ένας τύπος υπολογιστικής που βασίζεται στο Διαδίκτυο και παρέχει κοινούς πόρους επεξεργασίας και δεδομένων στους χρήστες υπολογιστές και σε άλλες συσκευές, κατόπιν αιτήσεως τους. Πρόκειται για ένα μοντέλο που επιτρέπει την πρόσβαση σε μια κοινόχρηστη δεξαμενή από διαμορφωμένους υπολογιστικούς πόρους (π.χ. δίκτυα υπολογιστών, διακομιστές, αποθήκευση, εφαρμογές και υπηρεσίες) , τα οποία μπορούν γρήγορα να διατεθούν και να απελευθερωθούν με ελάχιστη προσπάθεια διαχείρισης. Οι λύσεις πληροφορικής και αποθήκευσης cloud παρέχουν στους χρήστες και τις επιχειρήσεις διάφορες δυνατότητες αποθήκευσης και επεξεργασίας των δεδομένων τους είτε σε ιδιωτικά κέντρα δεδομένων, είτε σε κέντρα δεδομένων τρίτων που βρίσκονται μακριά από το χρήστη σε απόσταση που κυμαίνεται από μια πόλη ως σε ολόκληρο τον κόσμο. Το cloud computing βασίζεται στον διαμοιρασμό πόρων για την επίτευξη συνοχής και οικονομίας κλίμακας, παρόμοια με μια κοινοχρηστη υπηρεσία κοινής ωφέλειας όπως το δίκτυο διανόμής και μεταφοράς ηλεκτρικής ενέργειας στη περίπτωση της παροχής ηλεκτρικής ενέργειας.

Οι υποστηρικτές υποστηρίζουν ότι το cloud computing επιτρέπει στις εταιρείες να αποφεύγουν τα αρχικά κόστη υποδομής (π.χ. αγορά διακομιστών). Επίσης, δίνει τη δυνατότητα στους οργανισμούς να επικεντρωθούν στις βασικές τους δραστηριότητες αντί να ξοδεύουν χρόνο και χρήμα στην υποδομή υπολογιστών . Οι υποστηρικτές ισχυρίζονται επίσης ότι το cloud computing επιτρέπει στις επιχειρήσεις να κάνουν πιο γρήγορες τις εφαρμογές τους με βελτιωμένη διαχειρισιμότητα και λιγότερη συντήρηση και επιτρέπει στις ομάδες τεχνολογίας πληροφορικής (IT) να προσαρμόζουν ταχύτερα τους πόρους ώστε να ανταποκρίνονται στις διακυμάνσεις και τις απρόβλεπτες ανάγκες των επιχειρήσεων. Από την άλλη οι παροχείς νέφων συνήθως χρησιμοποιούν ένα μοντέλο "pay as you go". Αυτό μπορεί να οδηγήσει σε απροσδόκητα υψηλές χρεώσεις αν οι διαχειριστές δεν προσαρμοστούν στο μοντέλο τιμολόγησης του νέφους .



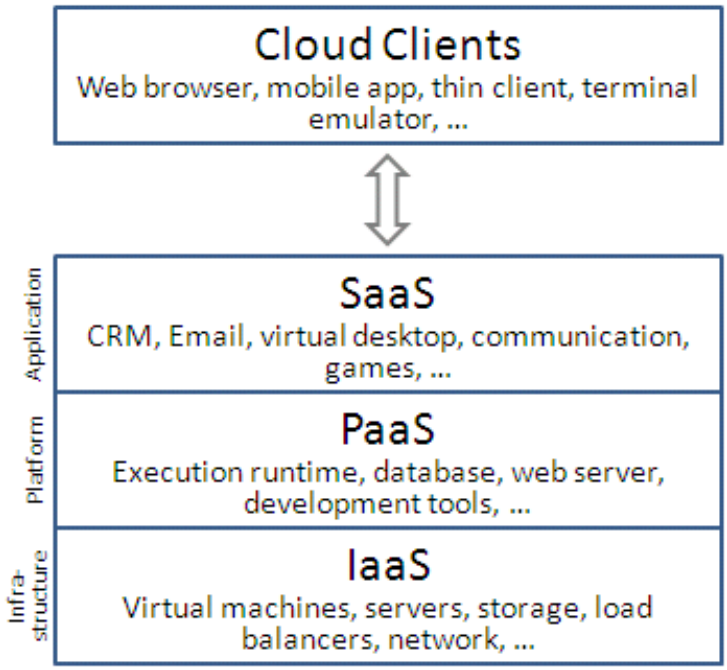
Σχήμα 1.2: *Το υπολογιστικό νέφος*

Το 2009, η διαθεσιμότητα δικτύων μεγάλης χωρητικότητας, οι χαμηλού κόστους υπολογιστές και συσκευές αποθήκευσης καθώς και η ευρεία υιοθέτηση της εικονικοποίησης του υλικού, της αρχιτεκτονικής προσανατολισμένης στις υπηρεσίες, της αυτόνομης υπολογιστικής (autonomic computing) και της υπολογιστικής κοινής ωφέλειας (utility computing), οδήγησαν στην ανάπτυξη του cloud computing. Οι εταιρείες μπορούν να μεγαλώνουν τους πόρους που παρέχουν, καθώς οι ανάγκες σε υπολογιστές αυξάνονται και στη συνέχεια να τους μειώνουν ξανά καθώς οι απαιτήσεις μειώνονται. Το 2013, αναφέρθηκε ότι το cloud computing είχε γίνει μια υπηρεσία με μεγάλη ζήτηση λόγω των πλεονεκτημάτων της υψηλής υπολογιστικής ισχύος, του φθηνού κόστους των υπηρεσιών, της υψηλής απόδοσης, της κλιμακωσιμότητας, της προσβασιμότητας καθώς και της διαθεσιμότητας. Ορισμένοι προμηθευτές νέφους αντιμετωπίζουν ποσοστά ανάπτυξης 50% ετησίως , αλλά εξακολουθούν να βρίσκονται σε στάδιο νηπιακής ηλικίας. Υπάρχουν παγίδες

που πρέπει να αντιμετωπιστούν ώστε να γίνουν οι υπηρεσίες υπολογιστικού νέφους πιο αξιόπιστες και φιλικές προς τον χρήστη.

1.4.1 Μοντέλα υπηρεσιών

Αν και η αρχιτεκτονική προσανατολισμένη στις υπηρεσίες υποστηρίζει το "όλα ως υπηρεσία" (με τα ακρωνύμια EaaS ή XaaS ή απλώς aas), οι πάροχοι υπολογιστικού νέφους προσφέρουν τις "υπηρεσίες" τους σύμφωνα με διαφορετικά μοντέλα, εκ των οποίων τα τρία βασικά μοντέλα κατά το NIST είναι: υποδομή ως υπηρεσία (IaaS), πλατφόρμα ως υπηρεσία (PaaS) και λογισμικό ως υπηρεσία (SaaS) (Σχήμα 1.3). Αυτά τα μοντέλα προσφέρουν αυξανόμενη αφαίρεση. Έτσι, συχνά απεικονίζονται ως στρώματα σε μια στοίβα: υποδομή-, πλατφόρμα- και λογισμικό-ως-υπηρεσία, αλλά παρόλα αυτά δεν πρέπει να σχετίζονται. Για παράδειγμα, μπορεί κανείς να τρέξει ένα πρόγραμμα στο IaaS και να αποκτήσει απευθείας πρόσβαση, χωρίς να το τυλίξει ως SaaS.



Σχήμα 1.3: *Μοντέλα υπηρεσιών Cloud computing που είναι διατεταγμένα ως στρώματα σε μια στοίβα*

Ο ορισμός του NIST (National Institute of Standards and Technology) για το cloud computing ορίζει τα μοντέλα υπηρεσιών ως εξής:

Λογισμικό ως Υπηρεσία (SaaS).

Η δυνατότητα που παρέχεται στον καταναλωτή είναι να χρησιμοποιεί τις εφαρμογές του παρόχου που εκτελούνται σε μια υποδομή νέφους. Οι εφαρμογές είναι προσβάσιμες από διάφορες συσκευές του πελάτη είτε μέσω μίας λεπτης διεπαφής πελάτη, όπως ενός προγράμματος περιήγησης ιστού (π.χ. ηλεκτρονικού ταχυδρομείου μέσω διαδικτύου) είτε μέσω διεπαφής προγράμματος. Ο καταναλωτής δεν διαχειρίζεται ή ελέγχει την υποκείμενη υποδομή του cloud, συμπεριλαμβανομένων των δικτύων, των διακομιστών, των λειτουργικών συστημάτων, των αποθηκευτικών χώρων ή ακόμη και των ατομικών δυνατοτήτων της εφαρμογής, με την πιθανή εξαίρεση περιορισμένων ρυθμίσεων διαμόρφωσης εφαρμογών για συγκεκριμένους χρήστες.

Πλατφόρμα ως υπηρεσία (PaaS).

Η δυνατότητα που παρέχεται στον καταναλωτή είναι η ανάπτυξη στην υποδομή του cloud εφαρμογών που δημιουργούνται ή αποκτώνται από καταναλωτές, οι οποίες δημιουργούνται χρησιμοποιώντας γλώσσες προγραμματισμού, βιβλιοθήκες, υπηρεσίες και εργαλεία που υποστηρίζονται από τον πάροχο. Ο καταναλωτής δεν διαχειρίζεται ή ελέγχει την υποκείμενη υποδομή του cloud, συμπεριλαμβανομένων των δικτύων, των διακομιστών, των λειτουργικών συστημάτων ή του χώρου αποθήκευσης, αλλά έχει τον έλεγχο των αναπτυγμένων εφαρμογών και ενδεχομένως των ρυθμίσεων διαμόρφωσης για το περιβάλλον φιλοξενίας των εφαρμογών αυτών.

Υποδομή ως υπηρεσία (IaaS).

Η δυνατότητα που παρέχεται στον καταναλωτή είναι η επεξεργασία, η αποθήκευση, τα δίκτυα και άλλοι βασικοί υπολογιστικοί πόροι, όπου ο καταναλωτής μπορεί να αναπτύξει και να εκτελέσει οποιοδήποτε λογισμικό, το οποίο μπορεί να περιλαμβάνει λειτουργικά συστήματα και εφαρμογές. Ο καταναλωτής δεν διαχειρίζεται ή ελέγχει την υποκείμενη υποδομή του cloud, αλλά έχει τον έλεγχο των λειτουργικών συστημάτων, των αποθηκευτικών χώρων και των αναπτυγμένων εφαρμογών. Και ενδεχομένως περιορισμένο έλεγχο επιλεγμένων στοιχείων δικτύου (π.χ. τα host firewalls).

1.5 Εικονικές Μηχανές

Στην τεχνολογία της πληροφορικής, μια εικονική μηχανή (VM) είναι μια εξομοίωση ενός συστήματος υπολογιστή. Οι εικονικές μηχανές βασίζονται στις αρχιτεκτονικές υπολογιστών και παρέχουν λειτουργικότητα ενός φυσικού υπολογιστή. Οι υλοποιήσεις τους μπορούν να περιλαμβάνουν εξειδικευμένο υλικό, λογισμικό ή τον συνδυασμό αυτών.

Υπάρχουν διαφορετικά είδη εικονικών μηχανών, με διαφορετικές λειτουργίες:

-Οι εικονικές μηχανές συστήματος (που ονομάζονται επίσης VM πλήρους εικονικοποίησης) αποτελούν υποκατάστατο μιας πραγματικής μηχανής. Παρέχουν τη λειτουργικότητα που απαιτείται για την εκτέλεση ολόκληρων λειτουργικών συστημάτων. Ένας μηχανισμός hypervisor χρησιμοποιεί εγγενή εκτέλεση για να μοιράζεται και να διαχειρίζεται το υλικό, επιτρέποντας πολλαπλά περιβάλλοντα που είναι απομονωμένα το ένα από το άλλο, όμως υπάρχουν στην ίδια φυσική μηχανή. Οι σύγχρονοι hypervisors χρησιμοποιούν ειδικό υλικό εικονικοποίησης, κυρίως από τις κεντρικές μονάδες CPU, για να πετύχουν εικονικοποίηση υποβοηθούμενη από το υλικό.

-Οι εικονικές μηχανές διεργασίας έχουν σχεδιαστεί για να εκτελούν προγράμματα υπολογιστών σε ένα περιβάλλον ανεξάρτητο από πλατφόρμες (όπως το Wine για τα Windows).

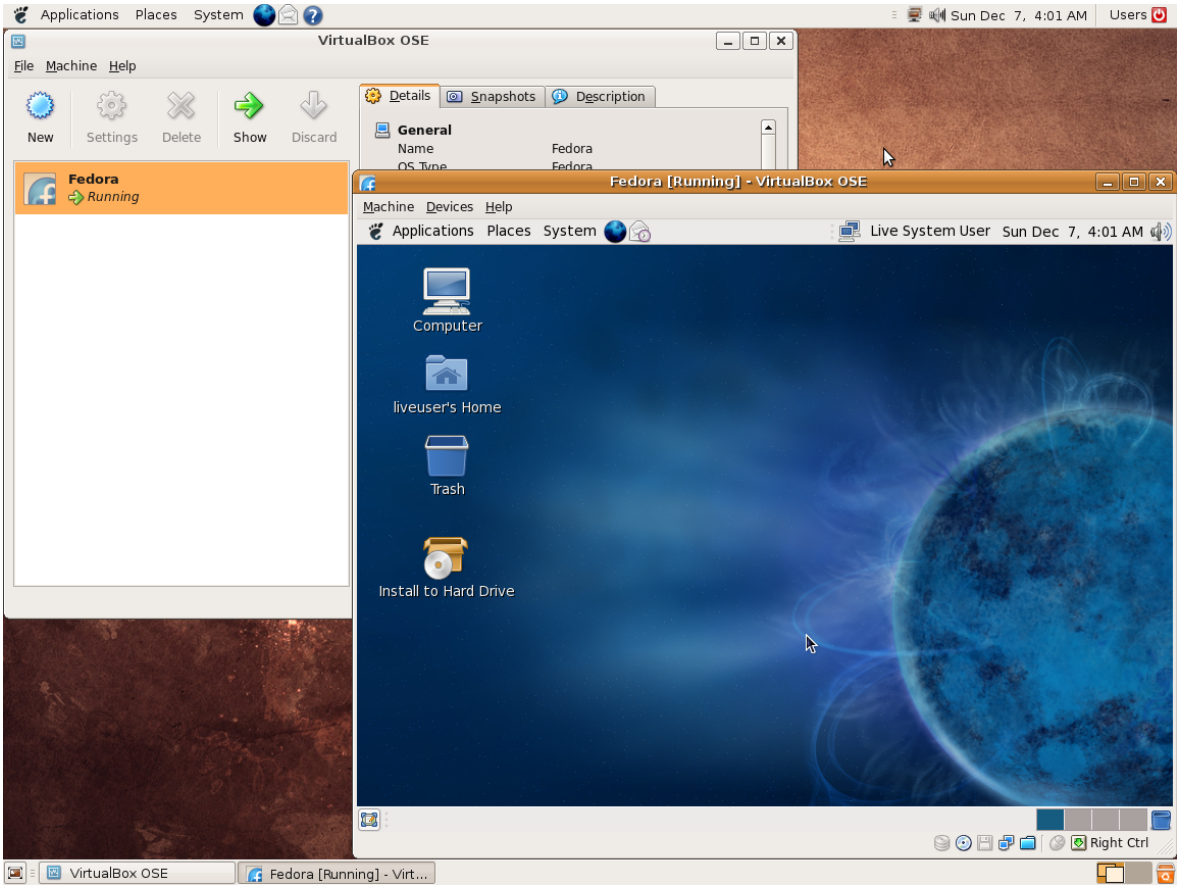
Ορισμένες εικονικές μηχανές, όπως το QEMU, έχουν σχεδιαστεί για να προσομοιώνουν διαφορετικές αρχιτεκτονικές και να επιτρέπουν την εκτέλεση εφαρμογών λογισμικού και λειτουργικών συστημάτων γραμμένων για άλλη CPU ή αρχιτεκτονική. Η εικονικοποίηση σε επίπεδο λειτουργικού συστήματος επιτρέπει την κατανομή των πόρων ενός υπολογιστή, μέσω της υποστήριξης του πυρήνα, σε πολλαπλά στιγμιότυπα απομονωμένων χώρων χρήστη, τα οποία ονομάζονται συνήθως δοχεία και μπορεί να φαίνονται και να λειτουργούν σαν πραγματικές μηχανές στους τελικούς χρήστες.

ΚΕΦΑΛΑΙΟ 2ο

Εικονικοποίηση (Virtualization)

2.1 Εικονικοποίηση πόρων και πλατφόρμας

Στην επιστήμη της πληροφορικής, η εικονικοποίηση (virtualization) είναι ένας ευρύς όρος των υπολογιστικών συστημάτων που αναφέρεται σε έναν μηχανισμό αφαίρεσης, στοχευμένο στην απόκρυψη των λεπτομερειών της υλοποίησης και της κατάστασης ορισμένων υπολογιστικών πόρων από πελάτες των πόρων αυτών (π.χ. εφαρμογές, άλλα συστήματα, χρήστες κλπ). Η εν λόγω αφαίρεση μπορεί είτε να αναγκάζει έναν πόρο να συμπεριφέρεται ως πλειάδα πόρων (π.χ. μία συσκευή αποθήκευσης σε διακομιστή τοπικού δικτύου), είτε πολλαπλούς πόρους να συμπεριφέρονται ως ένας (π.χ. συσκευές αποθήκευσης σε κατανεμημένα συστήματα). Η εικονικοποίηση δημιουργεί μία εξωτερική διασύνδεση η οποία αποκρύπτει την υποκείμενη υλοποίηση (π.χ. πολυπλέκοντας την πρόσβαση από διαφορετικούς χρήστες). Αυτή η προσέγγιση στην εικονικοποίηση αναφέρεται ως *εικονικοποίηση πόρων*. Μία άλλη προσέγγιση, ίδιας όμως νοοτροπίας, είναι η *εικονικοποίηση πλατφόρμας* (Σχήμα 2.1), όπου η αφαίρεση που επιτελείται προσομοιώνει ολόκληρους υπολογιστές. Το αντίθετο της εικονικοποίησης είναι η διαφάνεια: ένας εικονικός πόρος είναι ορατός, αντιληπτός, αλλά στην πραγματικότητα ανύπαρκτος, ενώ ένας διαφανής πόρος είναι υπαρκτός αλλά αόρατος.



Σχήμα 2.1: *Εικονικοποίηση πλατφόρμας* μέσω του προγράμματος VirtualBox

Σε ένα εικονικοποιημένο σύστημα συμμετέχουν τρεις παράγοντες: ο πελάτης μιας υπηρεσίας, ο παροχέας της υπηρεσίας και ένας ενδιάμεσος. Ο πελάτης και ο παροχέας αλληλεπιδρούν μέσω μίας πρότυπης διασύνδεσης, τις κλήσεις προς την οποία όμως αναχαιτίζει ο ενδιάμεσος. Ο τελευταίος επιτελεί την εικονικοποίηση λειτουργώντας ως παροχέας για τον πελάτη και ως πελάτης για τον παροχέα. Ένα παράδειγμα είναι ο μηχανισμός εικονικής μνήμης των σύγχρονων λειτουργικών συστημάτων, όπου ο διαχειριστής εικονικής μνήμης (ο ενδιάμεσος) παρεμβάλλεται μεταξύ ενός πραγματικού χώρου διευθύνσεων (παροχέας) και ενός εικονικού που γίνεται αντιληπτός από κάθε διεργασία (πελάτης). Ο ενδιάμεσος παρέχει την ψευδαίσθηση πολλών ισομεγέθων χώρων διευθύνσεων (ένας για κάθε διεργασία), ενώ στην πραγματικότητα υπάρχει μόνο ένας συνολικά (η πραγματική μνήμη). Ο πελάτης και ο παροχέας δεν γνωρίζουν τίποτα για την εικονικοποίηση και τη μεσολάβηση του ενδιάμεσου.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η εικονικοποίηση πλατφόρμας, όπου ένα λογισμικό ελέγχου («επόπτης» ή *hypervisor*) εκτελούμενο σε πραγματικό υλικό προσομοιώνει ένα υπολογιστικό περιβάλλον, μία «εικονική μηχανή», επάνω από το οποίο μπορεί να τρέξει κάποιο φιλοξενούμενο λογισμικό (συνήθως ένας πλήρης πυρήνας), απομονωμένο από το υπόλοιπο σύστημα. Η θεμελιώδης λογική πίσω από την εικονικοποίηση πλατφόρμας είναι η αρχή πως οποιαδήποτε λειτουργία μπορεί να εκτελεστεί είτε από λογισμικό είτε από εξειδικευμένο υλικό· οι μόνες διαφορές αφορούν την ευελιξία και την απόδοση. Είναι δυνατόν να προσομοιώνονται ταυτόχρονα πολλαπλές εικονικές μηχανές, εντελώς απομονωμένες μεταξύ τους, από το ίδιο λογισμικό ελέγχου. Η εικονικοποίηση πλατφόρμας εμφανίστηκε αρχικά τη δεκαετία του 1960, πριν από την επέλαση των μικροϋπολογιστών, σε μεγάλα, συγκεντρωτικά συστήματα (mainframes), αλλά μετά το 2000 και την αλματώδη αύξηση των επιδόσεων του υλικού των PC έχει γίνει πλέον κοινή πρακτική.

2.2 Είδη Εικονικοποίησης πλατφόρμας

Υπάρχουν πολλά είδη εικονικοποίησης πλατφόρμας. Ακολουθούν τα σημαντικότερα:

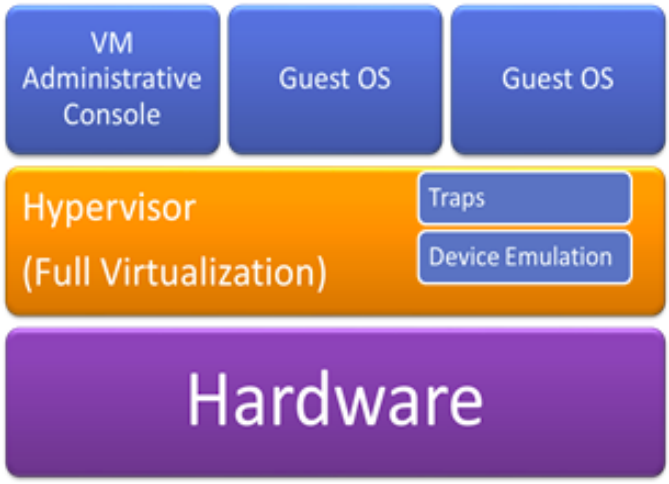
1. Προσομοίωση
2. Πλήρης Εικονικοποίηση
3. Παραεικονικοποίηση
4. Εικονικοποίηση υποβοηθούμενη από το υλικό

2.2.1. Προσομοίωση

Η εικονική μηχανή προσομοιώνει εξολοκλήρου μία αρχιτεκτονική υλικού, πιθανώς διαφορετική από το πραγματικό υποκείμενο υλικό, επιτρέποντας έτσι να εκτελεστεί επάνω της ένα μη τροποποιημένο, φιλοξενούμενο ΛΣ σχεδιασμένο για τον εξομοιούμενο επεξεργαστή (π.χ. QEMU, έκδοση για PowerPC του VirtualPC κλπ). Η εξομοίωση είναι διερμηνεία σε χρόνο εκτέλεσης του κώδικα του φιλοξενούμενου ΛΣ, με έναν κύκλο ανάγνωσης-αποκωδικοποίησης-εκτέλεσης όπου κάθε εντολή που ανήκει στο σύνολο εντολών του επεξεργαστή-πηγή μεταφράζεται σε μία εντολή του συνόλου εντολών του επεξεργαστή-στόχου. Παράλληλα η εικονική μηχανή παρέχει μία αφαίρεση της μνήμης, των συσκευών Εισόδου / Εξόδου κλπ, φροντίζοντας ώστε κάθε μεταφρασμένη εντολή που

απευθύνεται σε αυτά τα υποσυστήματα να τροποποιεί μόνο τις αφαιρέσεις / λογικές αναπαραστάσεις τους, οι οποίες κατευθύνονται και υλοποιούνται από το λογισμικό ελέγχου, και όχι το πραγματικό υλικό. Προκειμένου να αυξηθούν οι επιδόσεις είναι δυνατόν να χρησιμοποιηθεί δυναμική μετάφραση αντί για απλή εξομοίωση, όπου οι μεταφρασμένες εντολές αποθηκεύονται σε κρυφή μνήμη και μπορούν να επαναχρησιμοποιηθούν αργότερα χωρίς εκ νέου μετάφραση, ή δυναμική επαναμεταγλώττιση, όπου εκτός της χρήσης κρυφής μνήμης γίνεται και βελτιστοποίηση κρίσιμων τμημάτων του κώδικα (παρόμοια με τη μεταγλώττιση JIT της Java, του .NET και άλλων παρόμοιων πλατφορμών υψηλού επιπέδου).

2.2.2. Πλήρης Εικονικοποίηση

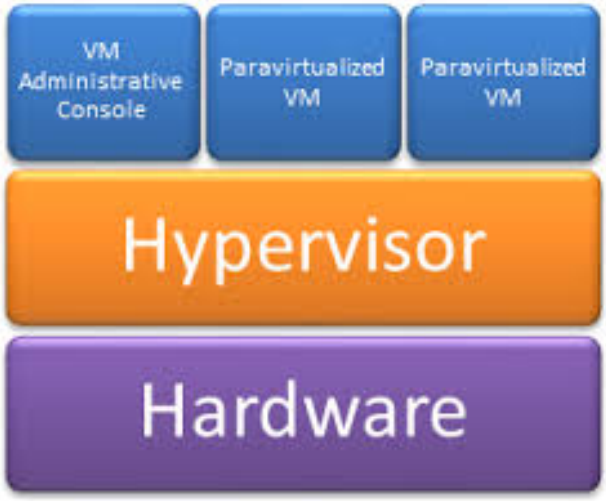


Σχήμα 2.2: *Πλήρης Εικονικοποίηση*

Η εικονική μηχανή προσομοιώνει επαρκές τμήμα του πραγματικού υποκείμενου υλικού ώστε να επιτρέπει την εκτέλεση επάνω της ενός μη τροποποιημένου, φιλοξενούμενου ΛΣ σχεδιασμένου για τον ίδιο τύπο επεξεργαστή με την πραγματική CPU (π.χ. VirtualPC, VMware, Win4Lin κλπ) (Σχήμα 2.2). Στην πλήρη εικονικοποίηση δεν χρειάζεται εξομοίωση του συνόλου εντολών του επεξεργαστή και μάλιστα ένα τμήμα του κώδικα του φιλοξενούμενου ΛΣ μπορεί να εκτελείται απευθείας από το υλικό, χωρίς μεσολάβηση του επόπτη, αρκεί να μην επηρεάζει υποσυστήματα εκτός του άμεσου ελέγχου του τελευταίου. Τα κρίσιμα σημεία του φιλοξενούμενου κώδικα ωστόσο, όπως αυτά που προσπαθούν να αποκτήσουν πρόσβαση στο υλικό (π.χ. κλήσεις συστήματος), συλλαμβάνονται από το λογισμικό ελέγχου και προσομοιώνονται, αφού τα αποτελέσματα κάθε λειτουργίας που επιτελείται σε μία εικονική μηχανή δεν επιτρέπεται να τροποποιούν

την κατάσταση άλλων εικονικών μηχανών, του επόπτη ή του υλικού. Αν το πραγματικό υλικό βοηθά και επιταχύνει τη λειτουργία του λογισμικού ελέγχου τότε η πλήρης εικονικοποίηση ονομάζεται *εγγενής* (native). Η βοήθεια αυτή αφορά κυρίως εύκολη διάκριση μεταξύ εντολών που μπορούν να εκτελεστούν απευθείας και εντολών που πρέπει να προσομοιωθούν από το λογισμικό. Όπως και στην εξομοίωση η εικονική μηχανή παρέχει στο φιλοξενούμενο ΛΣ μία αφαίρεση της μνήμης, των συσκευών Εισόδου / Εξόδου κλπ, ενώ η εγγενής εκτέλεση μεγάλου μέρους του κώδικα παρέχει πολύ καλύτερες επιδόσεις σε σχέση με την εξομοίωση.

2.2.3. Παραεικονικοποίηση

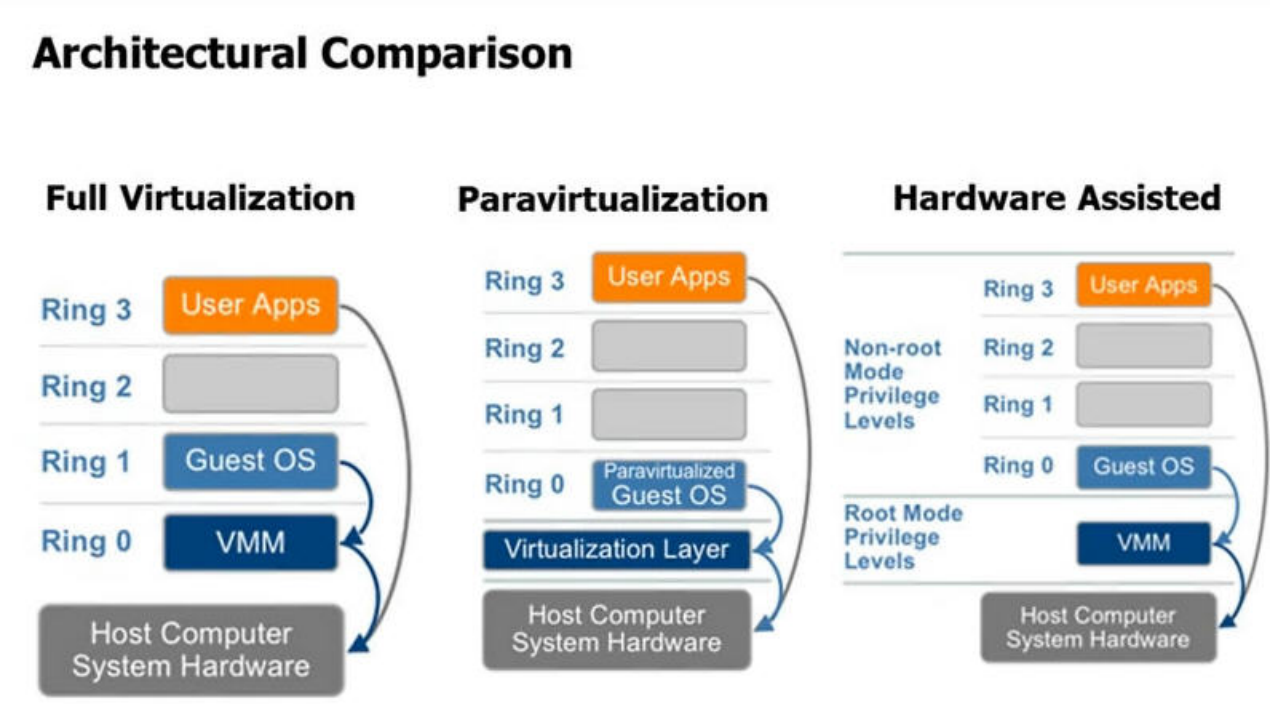


Σχήμα 2.3: *Παραεικονικοποίηση*

Η εικονική μηχανή δεν προσομοιώνει επακριβώς το υλικό αλλά παρέχεται στην εικονική μηχανή ένα API, μία προγραμματιστική διασύνδεση, ώστε να επιτρέπει την εκτέλεση επάνω της ενός τροποποιημένου, φιλοξενούμενου ΛΣ σχεδιασμένου για εκτέλεση από τον συγκεκριμένο επόπτη (π.χ. Denali, XEN) (Σχήμα 2.3). Το προαναφερθέν API ονομάζεται *διασύνδεση υπερκλήσεων* και ένα λειτουργικό σύστημα πρέπει να μεταφερθεί ρητά σε έκδοση κατάλληλη για εκτέλεση από ένα σύστημα παραεικονικοποίησης, ώστε ο φιλοξενούμενος πυρήνας αντί να προσπελαύνει το υλικό άμεσα να εκτελεί υπερκλήσεις και να αναμένει απαντήσεις ή ασύγχρονες ειδοποιήσεις από τον επόπτη. Το όφελος από τη βελτίωση των επιδόσεων και την απλοποίηση της γραφής του επόπτη είναι μεγάλο.

2.2.4. Εικονικοποίηση υποβοηθούμενη από το υλικό

Στην επιστήμη των υπολογιστών, η εικονικοποίηση με υποβοήθηση υλικού είναι μια προσέγγιση εικονικοποίησης πλατφόρμας που επιτρέπει την αποτελεσματική πλήρη εικονικοποίηση χρησιμοποιώντας τη βοήθεια από τις δυνατότητες υλικού, κυρίως από τους επεξεργαστές του οικοδεσπότη. Η πλήρης εικονικοποίηση χρησιμοποιείται για την προσομοίωση ενός πλήρους περιβάλλοντος υλικού ή μιας εικονικής μηχανής, όπου ένα μη τροποποιημένο λειτουργικό σύστημα guest (χρησιμοποιώντας το ίδιο σύνολο εντολών όπως το host μηχάνημα) εκτελείται σε πλήρη απομόνωση. Η εικονικοποίηση με υποβοήθηση υλικού προστέθηκε σε επεξεργαστές x86 (Intel VT-x ή AMD-V) το 2005 και το 2006 (αντίστοιχα). Η εικονικοποίηση με υποβοήθηση υλικού είναι επίσης γνωστή ως επιταχυνόμενη εικονικοποίηση. Το Xen την ονομάζει εικονικό μηχάνημα υλικού (HVM) και το Virtual Iron την ονομάζει εγγενή εικονικοποίηση.



Σχήμα 2.4: *Σύγκριση αρχιτεκτονικών διαφορετικών ειδών εικονικοποίησης*

2.2.4.1 Πλεονεκτήματα

Η εικονικοποίηση με υποβοήθηση υλικού μειώνει τα γενικά έξοδα συντήρησης της παραεικονικοποίησης καθώς μειώνει (ιδανικά, εξαλείφει) τις αλλαγές που απαιτούνται στο λειτουργικό σύστημα guest. Είναι επίσης πολύ πιο εύκολο να επιτευχθεί καλύτερη απόδοση. Ένα πλεονέκτημα στη πράξη της εικονικοποίησης που υποστηρίζεται από το υλικό έχει αναφερθεί από τους μηχανικούς της VMware και το Virtual Iron. Επίσης άλλα πλεονεκτήματα είναι η απουσία της ανάγκης να τροποποιηθεί το λειτουργικό σύστημα του επισκέπτη έναντι της παραεικονικοποίησης και η εκτέλεση του VMM κατευθείαν στο υλικό (Σχήμα 2.4).

2.2.4.2 Μειονεκτήματα

Η εικονικοποίηση με υποβοήθηση υλικού απαιτεί ρητή υποστήριξη στη CPU κεντρικού υπολογιστή, η οποία δεν είναι διαθέσιμη σε όλους τους επεξεργαστές x86 / x86_64. Μια "αμιγώς" υποστηριζόμενη από το υλικό προσέγγιση εικονικοποίησης, χρησιμοποιώντας εξ ολοκλήρου μη τροποποιημένα λειτουργικά συστήματα επισκεπτών, περιλαμβάνει πολλά VM traps και συνεπώς υψηλό κόστος χρόνου χρήσης της CPU. Περιορίζεται έτσι η επεκτασιμότητα και η αποτελεσματικότητα της ενοποίησης των διακομιστών . Αυτό το μειονέκτημα απόδοσης μπορεί να μετριαστεί με τη χρήση οδηγών παραεικονικοποίησης. Ο συνδυασμός ονομάζεται "υβριδικό virtualization". Αυτή είναι ίσως και η πιο αποδοτική προσέγγιση καθώς δεν χρειάζεται η συνεχή εναλλαγή εικονικών μηχανών που γίνεται στο hardware-assisted virtualization ενώ γίνεται εκμετάλλευση των επεκτάσεων εικονικοποίησης για τα κρίσιμα κομμάτια του κώδικα (και μόνο) μέσω του hypervisor παραεικονικοποίησης κάτι που δεν γίνεται στην απλή παραεικονικοποίηση.

ΚΕΦΑΛΑΙΟ 3ο

Τεχνολογίες εικονικοποίησης υποβοηθούμενης από το υλικό

3.1 Τεχνολογίες εικονικοποίησης Intel

Παλαιότερα με την κωδική ονομασία "Vanderpool", το VT-x αντιπροσωπεύει την τεχνολογία της Intel για εικονικοποίηση στην πλατφόρμα x86. Στις 13 Νοεμβρίου 2005, η Intel κυκλοφόρησε δύο μοντέλα Pentium 4 (μοντέλα 662 και 672) ως οι πρώτοι επεξεργαστές της Intel που υποστηρίζουν το VT-x. Η σημαία της CPU για την ικανότητα VT-x είναι "vmx". Στο Linux, αυτό μπορεί να ελεγχθεί μέσω του /proc/cpuinfo ή στο macOS μέσω sysctl machdep.cpu.features. Η τεχνολογία Intel VT-x, VMX είναι αντίστοιχη με την τεχνολογία AMD – V, SVM της AMD.

Το "vmx" αντιπροσωπεύει τις Επεκτάσεις Εικονικής Μηχανής (Virtual Machine eXtensions), οι οποίες προσθέτουν δέκα νέες οδηγίες: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF και VMXON. Αυτές οι οδηγίες επιτρέπουν την είσοδο και την έξοδο από μια λειτουργία

εικονικής εκτέλεσης όπου το λειτουργικό σύστημα επισκέπτη θεωρεί ότι λειτουργεί με πλήρες προνόμιο (δακτύλιος 0), αλλά το λειτουργικό σύστημα παραμένει προστατευμένο.

Υπάρχουν δύο καταστάσεις λειτουργίας που εκτελούνται υπό την εικονικοποίηση: VMX λειτουργία root και VMX non-root λειτουργία (Σχήμα 3.1). Συνήθως, μόνο το λογισμικό ελέγχου εικονικοποίησης, που ονομάζεται Virtual Machine Monitor (VMM), εκτελείται υπό λειτουργία root, ενώ τα λειτουργικά συστήματα που εκτελούνται στην κορυφή των εικονικών μηχανών εκτελούνται υπό non-root λειτουργία. Το λογισμικό που τρέχει πάνω από εικονικές μηχανές ονομάζεται επίσης "φιλοξενούμενο λογισμικό".

Για να εισέλθει στη λειτουργία εικονικοποίησης, το λογισμικό πρέπει να εκτελέσει την εντολή VMXON και στη συνέχεια να καλέσει το λογισμικό VMM. Το λογισμικό VMM μπορεί να εισέλθει σε κάθε εικονική μηχανή χρησιμοποιώντας την εντολή VMLAUNCH και να βγεί με τη χρήση της εντολής VMRESUME. Αν το VMM θέλει να κλείσει και να βγει από τη λειτουργία εικονικοποίησης, εκτελεί την εντολή VMXOFF.

Από το 2015, σχεδόν όλοι οι νεότεροι επεξεργαστές διακομιστών και επιτραπέζιων υπολογιστών Intel υποστηρίζουν το VT-x, με μερικούς από τους επεξεργαστές Intel Atom ως την πρώτη εξαίρεση. Με κάποιες μητρικές πλακέτες, οι χρήστες πρέπει να ενεργοποιήσουν τη δυνατότητα VT-x της Intel στη ρύθμιση του BIOS προτού οι εφαρμογές μπορέσουν να το χρησιμοποιήσουν.

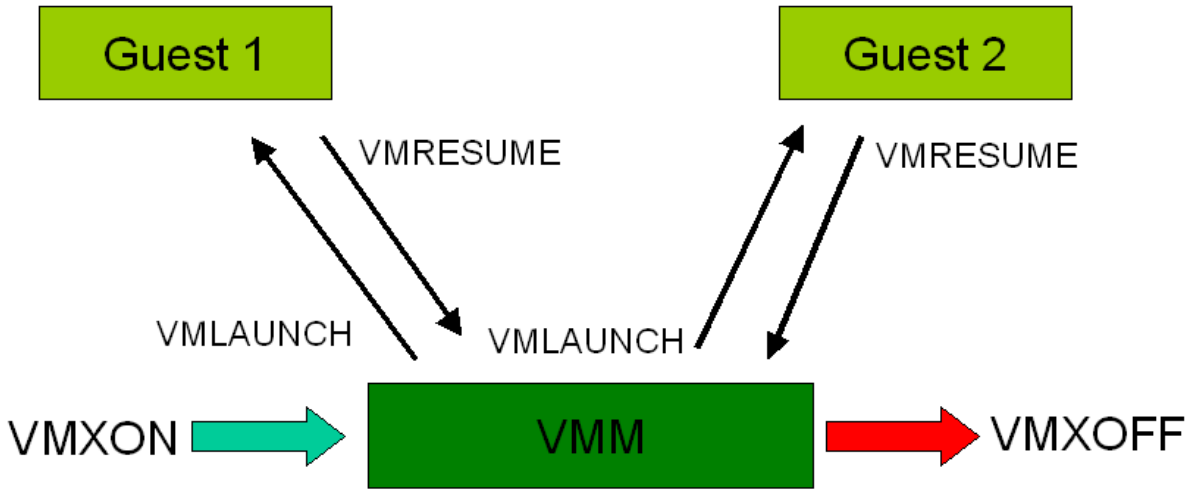
Οι πιο πρόσφατοι επεξεργαστές έχουν μια επέκταση που ονομάζεται EPT (Extended Page Tables), η οποία επιτρέπει σε κάθε επισκέπτη να έχει δικό του πίνακα σελίδων για να παρακολουθεί τις διευθύνσεις μνήμης. Χωρίς αυτήν την επέκταση, το VMM πρέπει να βγεί από την εικονική μηχανή για να πραγματοποιήσει μεταφράσεις διευθύνσεων. Αυτή η εργασία εξόδου και επιστροφής μειώνει την απόδοση. Επομένως, το EPT αυξάνει την απόδοση εικονικοποίησης. Το EPT είναι το αντίστοιχο RVI (Rapid Virtualization Indexing, πρώην Nested Page Tables-NPT) της AMD, και το αντίστοιχο stage-2 SMMU address translation της ARM.

Η Intel άρχισε να συμπεριλαμβάνει την τεχνολογία EPT για την εικονικοποίηση πίνακα σελίδων από την αρχιτεκτονική Nehalem που κυκλοφόρησε το 2008. Το 2010, η Westmere πρόσθεσε την υποστήριξη για την έναρξη του λογικού επεξεργαστή του επισκέπτη απευθείας σε real mode - ένα χαρακτηριστικό που ονομάζεται "απεριόριστος επισκέπτης", το οποίο απαιτεί το EPT για να λειτουργήσει.

Από τη μικροαρχιτεκτονική Haswell (που ανακοινώθηκε το 2013), η Intel άρχισε να συμπεριλαμβάνει τη σκίαση VMCS ως τεχνολογία που επιταχύνει την εμφωλευμένη εικονικοποίηση των VMM. Η δομή ελέγχου εικονικής μηχανής (VMCS) είναι μια δομή δεδομένων στη μνήμη που υπάρχει ακριβώς μία φορά ανά VM, και διαχειρίζεται από το VMM. Με κάθε αλλαγή του πλαισίου εκτέλεσης, μεταξύ διαφορετικών εικονικών μηχανών, το VMCS αποκαθίσταται για την τρέχουσα εικονική μηχανή καθορίζοντας έτσι την κατάσταση του εικονικού της επεξεργαστή. Μόλις χρησιμοποιηθούν περισσότερα από ένα

VMM ή ένθετα VMMs, εμφανίζεται ένα πρόβλημα παρόμοιο με αυτό που απαιτούσε την εφεύρεση του σκιώδους (shadow) πίνακα σελίδων. Σε τέτοιες περιπτώσεις, το VMCS πρέπει να σκιάζεται πολλές φορές (σε περίπτωση εμφώλευσης) και να υλοποείται μερικώς σε λογισμικό σε περίπτωση που δεν υπάρχει υποστήριξη υλικού από τον επεξεργαστή. Για να καταστεί πιο αποτελεσματική η διαχείριση των σκιώδων VMCS, η Intel υλοποίησε υποστήριξη υλικού για σκίαση VMCS.

Η αρχιτεκτονική Intel υποστηρίζει και την τεχνολογία VT-d που είναι ουσιαστικά η υλοποίηση μία μονάδας διαχείρισης μνήμης εισόδου – εξόδου (IOMMU). Η τεχνολογία αυτή αντιστοιχεί στην τεχνολογία εικονικοποίησης AMD – Vi και στην SMMU σταδίου-1 της ARM. Στο αντίστοιχο κεφάλαιο της ARM παρακάτω αναλύεται η χρήση της τεχνολογίας αυτής και τα οφέλη της.



Σχήμα 3.1: *Τρόπος λειτουργίας της τεχνολογίας εικονικοποίησης Intel VT-x*

3.2 Τεχνολογίες εικονικοποίησης AMD

Η AMD ανέπτυξε τις πρώτες γενεές επεκτάσεων εικονικοποίησης με την κωδική ονομασία "Pacifica" και αρχικά τις δημοσίευσε ως AMD Secure Virtual Machine (SVM), αλλά τις κυκλοφόρησε αργότερα με το εμπορικό σήμα AMD Virtualization, συντομογραφία AMD-V. Στις 23 Μαΐου 2006, η AMD κυκλοφόρησε τον Athlon 64 ("Orleans"), τον Athlon 64 X2 ("Windsor") και τον Athlon 64 FX ("Windsor") ως τους πρώτους επεξεργαστές της AMD που υποστηρίζουν αυτήν την τεχνολογία.

Η τεχνολογία AMD-V περιλαμβάνει επίσης την οικογένεια επεξεργαστών Athlon 64 και Athlon 64 X2, με εκδόσεις "F" ή "G" , για το socket AM2, τους Turion 64 X2 και Opteron δεύτερης γενιάς και τους τρίτης γενιάς Phenom και Phenom II επεξεργαστές. Οι επεξεργαστές APU Fusion υποστηρίζουν AMD-V. Το AMD-V δεν υποστηρίζεται από επεξεργαστές Socket 939. Οι μόνοι επεξεργαστές Sempron που την υποστηρίζουν είναι οι Huron και Sargas.

Οι επεξεργαστές AMD Opteron (Σχήμα 3.2) αρχικά της οικογένειας 0x10 της σειράς της Βαρκελώνης και οι επεξεργαστές Phenom II υποστηρίζουν μια δεύτερης γενιάς τεχνολογία εικονικοποίησης υλικού που ονομάζεται Rapid Virtualization Indexing (παλαιότερα γνωστή ως NPT - Nested Page Tables) και η οποία υιοθετήθηκε αργότερα από την Intel ως EPT. Η σημαία της CPU για το AMD-V είναι "svm". Αυτό μπορεί να ελέγχεται στα παράγωγα BSD μέσω dmesg ή sysctl και στο Linux μέσω του /proc/cpuinfo. Η τεχνολογία AMD – V, SVM λειτουργεί με αντίστοιχο τρόπο με την τεχνολογία Intel VT-x, VMX.

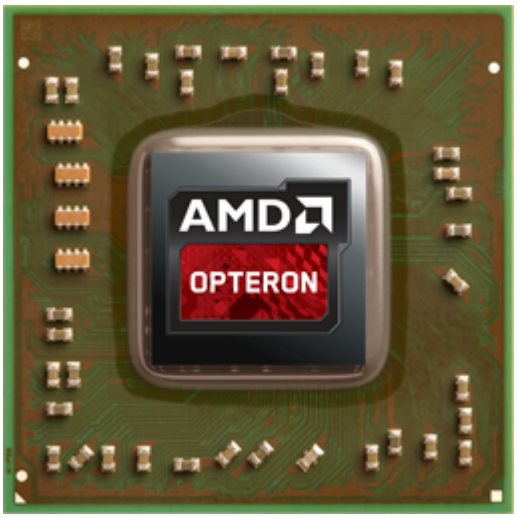
Πιο συγκεριμένα η τεχνολογία AMD-V βασίζεται στην τεχνολογία: αρχιτεκτονική άμεσης σύνδεσης (Direct Connect Architecure), η οποία μειώνει το γενικό κόστος χρόνου εκτέλεσης επιτρέποντας την άμεση επικοινωνία μεταξύ επισκέπτη και φυσικού επεξεργαστή και παρέχοντας βελτιωμένο χειρισμό μνήμης.

-Η ταχεία ευρετηρίαση εικονικοποίησης, Rapid Virtualization Indexing, επιτρέπει στις εικονικές μηχανές να διαχειρίζονται πιο άμεσα τη μνήμη, συμβάλλοντας στη βελτίωση της απόδοσης για πολλές εικονικές εφαρμογές. Χρησιμοποιώντας τους πόρους πυριτίου πάνω στο ίδιο chip και όχι το λογισμικό, η ταχεία ευρετηρίαση εικονικοποίησης μπορεί να ελαχιστοποιήσει τον αριθμό των απαραίτητων κύκλων του επόπτη, καθώς και τη σχετική ποινή εκτέλεσης που συνήθως συνδέεται με την εικονικοποίηση. Η Rapid Virtualization Indexing σχεδιάστηκε επίσης για να ελαχιστοποιήσει τον "χρόνο μετάβασης στον κόσμο" – χρόνος που δαπανάται για την εναλλαγή από μια εικονική μηχανή σε άλλη – ώστε να επιτυγχάνεται έτσι ταχύτερη απόκριση από τις εφαρμογές. Είναι αντίστοιχη των EPT της Intel και της SMMU σταδίου-2 της ARM.

-Ο Translation Look-aside Buffer (TLB) με ετικέτα ο οποίος είναι μοναδικός για τους επεξεργαστές AMD Opteron, επιτρέπει ταχύτερους χρόνους μεταγωγής μεταξύ εικονικών μηχανών διατηρώντας μια χαρτογράφηση στις προσωπικές περιοχές μνήμης που χρησιμοποιούνται από τις εικονικές μηχανές. Διακρίνοντας τους χώρους μνήμης που χρησιμοποιούνται από κάθε εικονική μηχανή, βοηθά στη μείωση της επιβάρυνσης της διαχείρισης μνήμης και βελτιώνει την απόκριση κατά την εναλλαγή μεταξύ των εικονικών μηχανών.

-Η AMD-V εκτεταμένη μετανάστευση, Extended Migration, έχει σχεδιαστεί για να επιτρέπει σε λύσεις λογισμικού εικονικοποίησης να επιτυγχάνουν ζωντανή μετάβαση των εικονικών μηχανημάτων σε ολόκληρο το τρέχον εύρος επεξεργαστών AMD Opteron.

Η αρχιτεκτονική AMD υποστηρίζει και την τεχνολογία AMD-Vi που είναι ουσιαστικά η υλοποίηση μία μονάδας διαχείρισης μνήμης εισόδου – εξόδου (IOMMU). Η τεχνολογία αυτή αντιστοιχεί στην τεχνολογία εικονικοποίησης VT-d της Intel και στην SMMU σταδίου-1 της ARM. Στο αντίστοιχο κεφάλαιο της ARM παρακάτω αναλύεται η χρήση της τεχνολογίας αυτής και τα οφέλη της.



Σχήμα 3.2: *Πλακίδιο επεξεργαστή AMD Opteron*

3.3 Τεχνολογίες εικονικοποίησης ARM

Οι επεκτάσεις ARM Virtualization ενισχύουν την αρχιτεκτονική ARM v8.1 για να υποστηρίξουν την ανάπτυξη εικονικών συστημάτων. Τα βασικά στοιχεία αυτών των επεκτάσεων είναι:

-Η εισαγωγή ενός νέου τρόπου εκτέλεσης Hypervisor, υψηλότερης προτεραιότητας από τη λειτουργία Supervisor. Αυτό θα επιτρέψει στο VMM να εκτελεστεί με υψηλότερο προνόμιο από τα guest OS και τα guest OS να εκτελεστούν με τα παραδοσιακά δικαιώματα του λειτουργικού συστήματος, καταργώντας την ανάγκη χρήσης τεχνικών παραεικονικοποίησης.

-Η παροχή μηχανισμών για τη διευκόλυνση του χειρισμού των διακοπών, με εγγενή διάκριση των διακοπών, ώστε να εξασφαλίζεται η σωστή εκτέλεση της εποπτείας - παρακολούθησης (monitor), των hypervisors, των ενεργών επί του παρόντος λειτουργικών συστημάτων επισκέπτη και τα μή ενεργών - επί του παρόντος - λειτουργικών συστημάτων επισκέπτη. Αυτό θα μειώσει δραματικά την πολυπλοκότητα των χειρισμών των διακοπών, η

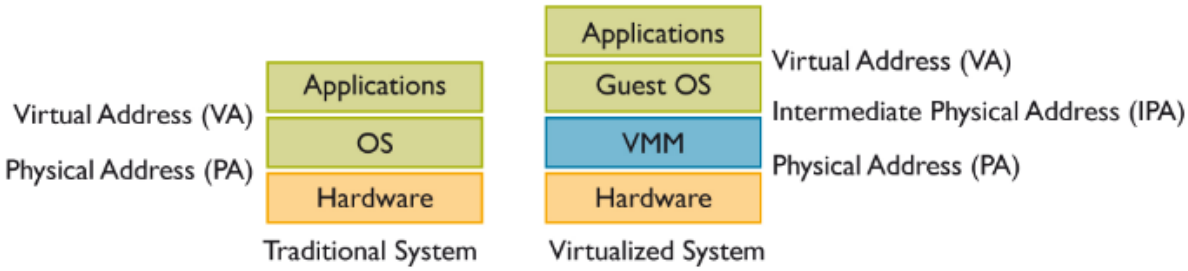
οποία υπάρχει όταν χρησιμοποιούνται τεχνικές εξομοίωσης λογισμικού και δομές σκιάς μέσα στο VMM.

-Η παροχή μίας System MMU για την ενίσχυση της διαχείρισης μνήμης, υποστηρίζοντας: πολλαπλά περιβάλλοντα μετάφρασης για πολλαπλούς ικανούς διαχειριστές DMA, δύο επίπεδα μετάφρασης διευθύνσεων, επιτάχυνση υλικού και λογική αφαίρεση.

-Λειτουργικότητα εντοπισμού σφαλμάτων με στόχο την ενεργοποίηση της πρόσβασης προγραμμάτων εντοπισμού σφαλμάτων σε μεμονωμένα λειτουργικά συστήματα επισκεπτών.

3.3.1. Προκλήσεις διαχείρισης μνήμης σε εικονικά συστήματα

Σε ένα εικονικό σύστημα, το θέμα της διαχείρισης μνήμης είναι πολύ σημαντικό και μπορεί να οδηγήσει σε σημαντική πολυπλοκότητα. Μία από τις βασικές λειτουργίες των περισσότερων λειτουργικών συστημάτων είναι η υποστήριξη ενός σταδίου διαχείρισης εικονικής μνήμης για τη διαίρεση της φυσικής μνήμης που ελέγχεται από το λειτουργικό σύστημα σε πολλαπλές διεργασίες. Σε ένα σύστημα όπου κάθε guest OS λειτουργεί μέσα σε μια εικονική μηχανή, η μνήμη που διατίθεται από το guest OS δεν είναι η πραγματική φυσική μνήμη του συστήματος, αλλά είναι μια ενδιάμεση φυσική μνήμη. Το VMM ελέγχει άμεσα την κατανομή της πραγματικής φυσικής μνήμης, εκπληρώνοντας έτσι το ρόλο του διαιτητή των κοινών φυσικών πόρων.



Σχήμα 3.3: *Στάδια μετάφρασης διευθύνσεων σε παραδοσιακά και εικονικά συστήματα*

Υπάρχουν δύο προσεγγίσεις για το χειρισμό των δύο σταδίων της μετάφρασης διευθύνσεων (από VA σε IPA και από IPA σε PA) (Σχήμα 3.3). Σε υπάρχοντα συστήματα όπου παρέχεται από το υλικό μόνο ένα στάδιο μετάφρασης του χώρου διευθύνσεων μνήμης, για παράδειγμα χρησιμοποιώντας την MMU μέσα στη CPU, ο hypervisor πρέπει να διαχειρίζεται άμεσα τη σχέση μεταξύ VA, IPA και PA. Αυτό γίνεται γενικά από τον hypervisor διατηρώντας τους δικούς του πίνακες μετάφρασης (που ονομάζονται πίνακες μετάφρασης σκιάς), οι οποίοι προέρχονται από την μετάφραση κάθε πίνακα μετάφρασης των λειτουργικών συστημάτων επισκέπτη. Ο hypervisor πρέπει να διασφαλίσει ότι όλες οι αλλαγές στους πίνακες μετάφρασης του λειτουργικού συστήματος του επισκέπτη συγχρονίζονται με τους πίνακες στις δομές της σκιάς, καθώς να διασφαλίσει και την επιβολή της προστασίας και την ανακατεύθυνση των σφαλμάτων πρόσβασης στο κατάλληλο στάδιο. Ο απαιτούμενος μηχανισμός μπορεί να είναι περίπλοκος και να προσθέτει επιβάρυνση απόδοσης. Η εναλλακτική λύση είναι να χρησιμοποιηθεί βοήθεια υλικού για τα δύο στάδια της μετάφρασης, και αυτό είναι που επιτρέπει η ARM SMMU.

3.3.1.1 Κατακερματισμός μνήμης

Ανάλογα με τον τρόπο με τον οποίο υλοποιείται το VMM και την τροποποίηση του λειτουργικού συστήματος επισκέπτη, υπάρχει μια σειρά πιθανών σχέσεων μεταξύ των χώρων μνήμης της ενδιάμεσης φυσικής διεύθυνσης (IPA) και της φυσικής διεύθυνσης μνήμης (PA). Το IPA και το PA μπορούν να χαρτογραφηθούν απευθείας (αφαιρώντας την ανάγκη μετάφρασης) ή να αντισταθμίζονται από μια σταθερά, με το VMM να διαχειρίζεται την προστασία της μνήμης. Αυτό επιτρέπει σε κάθε λειτουργικό σύστημα να κατέχει ένα ποσοστό συνεχούς φυσικής μνήμης. Στην πραγματικότητα η μνήμη είναι δαπανηρή. Ως εκ τούτου είναι πιο συνηθισμένο ότι οποιοδήποτε μπλοκ (συνήθως μια σελίδα) φυσικής μνήμης επιτρέπεται να αντιστοιχιστεί σε κάθε guest OS, ανεξάρτητα. Αυτό αναφέρεται ως κατακερματισμένη σχέση. Οι επεκτάσεις εικονικοποίησης ARM επιτρέπουν μια κατακερματισμένη σχέση μεταξύ των χώρων IPA και PA. Καθώς αυξάνεται ο αριθμός των λειτουργικών συστημάτων επισκεπτών, η σχέση μεταξύ της μνήμης IPA ενός guest OS και της μνήμης PA μπορεί να γίνει πολύπλοκη. Ορισμένα συστήματα ενδέχεται να προβλέπουν τη δημιουργία και την καταστροφή των λειτουργικών συστημάτων επισκεπτών ζωντανά κατά τον χρόνο εκτέλεσης, ή περιοχές φυσικής μνήμης μπορεί να μοιράζονται μεταξύ πολλών λειτουργικών συστημάτων επισκεπτών. Ο κατακερματισμός της φυσικής μνήμης μπορεί να είναι ένα πραγματικό ζήτημα σε συστήματα όπου η χρήση μεγάλων τμημάτων συνεχόμενης φυσικής μνήμης είναι σημαντική.

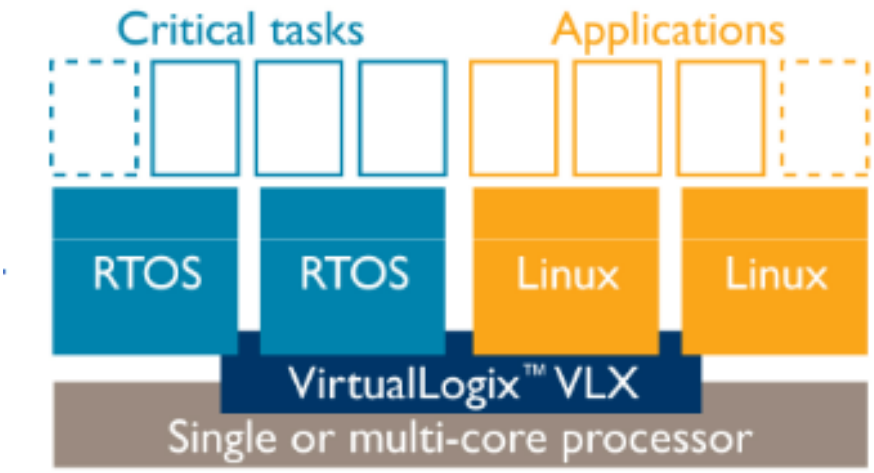
3.3.1.2 Πολλαπλοί κύριοι διαχειριστές με δυνατότητα άμεσης προσπέλασης μνήμης (Multiple DMA capable masters)

Η εισαγωγή ενός νέου επιπέδου μετάφρασης διευθύνσεων έχει σημαντικές επιπτώσεις σε άλλους επεξεργαστές δεδομένων και περιφερειακά του συστήματος που μπορεί να μοιράζονται το ίδιο μέσο φυσικής μνήμης. Γενικά, οι περιφερειακές συσκευές ανήκουν στο λειτουργικό σύστημα και προγραμματίζονται σε χώρο PA. Επίσης, όλες οι συσκευές που έχουν δυνατότητα άμεσης πρόσβασης σε μνήμη (DMA) δημιουργούν διευθύνσεις στο χώρο PA. Σε ένα εικονικοποιημένο περιβάλλον, αυτές οι φυσικές διευθύνσεις είναι στην πραγματικότητα ενδιάμεσες IPA και ως εκ τούτου απαιτούν περαιτέρω μετάφραση. Επί του παρόντος, αυτό θα χειριζόταν από το VMM, είτε με εξομοίωση λογισμικού είτε με τεχνικές άμεσης αντιστοίχισης συσκευών (εγγενής πρόσβαση στη συσκευή από τα φιλοξενούμενα λειτουργικά συστήματα που διαιτητεύονται από το VMM). Οποιαδήποτε λύση απαιτεί την μεταφορά του οδηγού συσκευής, σε κάθε guest OS, και μπορεί να προσθέσει μεγάλη πολυπλοκότητα στο VMM καθώς προκαλεί και σημαντικά γενικά κόστη απόδοσης. Αυτό είναι ένα σημαντικό ζήτημα. Οι πλατφόρμες ηλεκτρονικών ειδών ευρείας κατανάλωσης είναι σύνθετα ετερογενή συστήματα, όπου συνυπάρχει μια μεγάλη ποικιλία επεξεργαστών με άμεση πρόσβαση στο σύστημα μνήμης. Για παράδειγμα, μια τυπική πλατφόρμα υπολογιστών όπως το Tegra 2 ™ της NVIDIA®, η πλατφόρμα OMAP4 ™ της Texas Instruments® ή η πλατφόρμα U8500 ™ της ST-Ericsson®, περιλαμβάνει έναν πολυπύρηνο επεξεργαστή εφαρμογών ARM με ενσωματωμένο MMU για κάθε πυρήνα, μονάδα επεξεργασίας γραφικών OpenGL ES2® με ενσωματωμένο MMU, μονάδα επεξεργασίας ψηφιακού σήματος (DSP), επεξεργαστή σημάτων εικόνας (ISP), μικροελεγκτή Power Manager (PM) και ελεγκτή άμεσης μνήμης (DMAC). Στο όχι πολύ απομακρυσμένο μέλλον μπορεί να θεωρηθεί ότι τα ετερογενή κατανεμημένα μοντέλα υπολογιστών, όπως η GPU γενικής χρήσης OpenCL (GPGPU), θα είναι κοινή θέση. Εκτός από τα προαναφερθέντα γενικά κόστη πολυπλοκότητας και επιδόσεων που σχετίζονται με τη διαδικασία μετάφρασης διεύθυνσης, δεν υπάρχει σίγουρος μηχανισμός που να μπορεί να σταματήσει οποιαδήποτε από τις συσκευές ή τους επεξεργαστές έχουν δυνατότητα άμεσης πρόσβασης μνήμης (DMA) από το να καταστρέψουν τη φυσική μνήμη που χρησιμοποιείται από άλλα λειτουργικά συστήματα επισκεπτών.

3.3.1.3 Εξασφάλιση της ακεραιότητας του συστήματος

Ένα άλλο σημαντικό ζήτημα μπορεί να παρουσιαστεί όταν εξετάζουμε την εικονικοποίηση για την ενοποίηση του υλικού σε συστήματα όπου μια εικονική μηχανή είναι αφιερωμένη σε κρίσιμους σκοπούς ασφαλείας. Για παράδειγμα, σε ένα

αυτοκινητιστικό υποσύστημα, η εικονικοποίηση μπορεί να επιτρέψει την αυστηρή απομόνωση των εφαρμογών που σχετίζονται με την οδήγηση (όπως κάμερα οπίσθιας προβολής, αυτόματο σύστημα στάθμευσης) και τον έλεγχο της ταυτότητας του οδηγού, από λιγότερο κρίσιμες εφαρμογές που σχετίζονται με τα συστήματα πληροφοριών και ψυχαγωγίας (στερεοφωνικό σύστημα, μπροστινό ηλεκτρονικό τμήμα, μονάδα ψυχαγωγίας πίσω καθίσματος κλπ.) (Σχήμα 3.4). Οι δυσλειτουργίες, τα σφάλματα ή τα ατυχήματα κατά το χρόνο εκτέλεσης, σε μη κρίσιμο περιβάλλον, δεν επιτρέπεται να θέσουν σε κίνδυνο άλλα τμήματα του συστήματος.

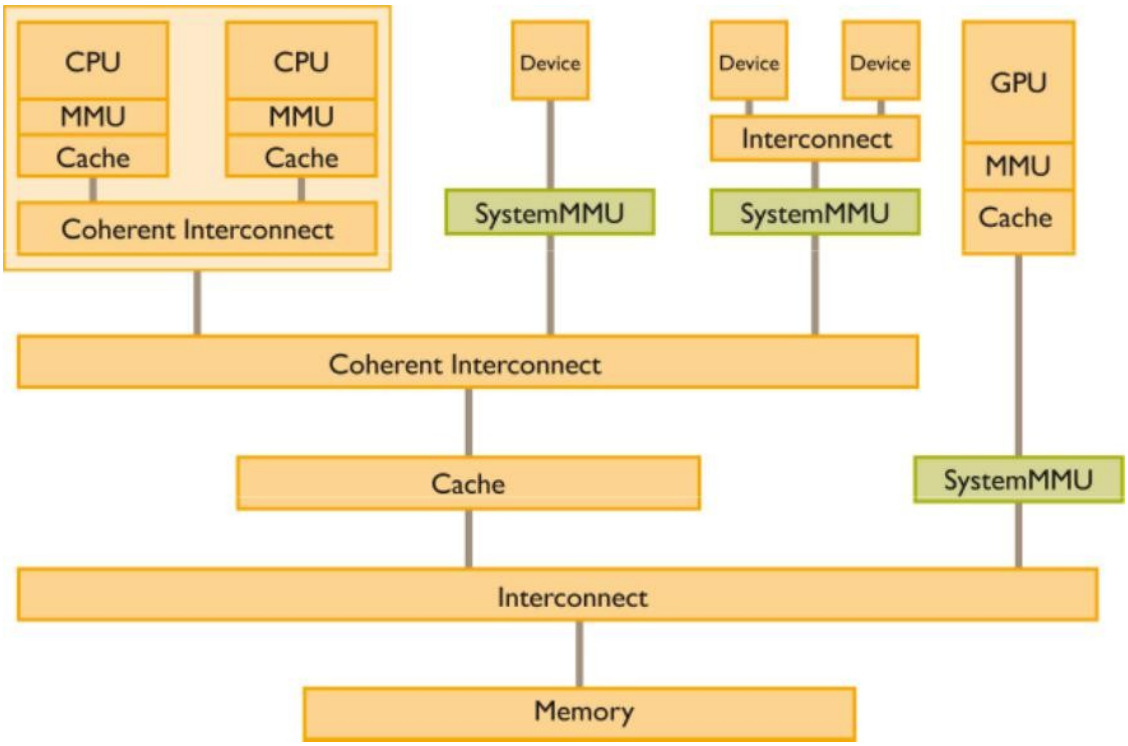


Σχήμα 3.4: *Το VirtualLogix (τώρα RedBend) VLX Hypervisor χρησιμοποιείται για την απομόνωση κρίσιμων εργασιών.*

3.3.2 Η System MMU (SMMU)

Για την αντιμετώπιση όλων των προαναφερθέντων ζητημάτων και περιορισμών, οι επεκτάσεις εικονικοποίησης της αρχιτεκτονικής ARM εισάγουν την έννοια της Μονάδας Διαχείρισης Μνήμης Συστήματος (SMMU). Ένα MMU συστήματος είναι μια συσκευή υλικού που έχει σχεδιαστεί για να παρέχει υπηρεσίες μετάφρασης διευθύνσεων και λειτουργίες προστασίας σε οποιαδήποτε συσκευή του συστήματος με δυνατότητα αμεσης προσπέλασης μνήμης DMA, εκτός από τον κύριο επεξεργαστή. Αυτό περιλαμβάνει επιταχυντές υλικού όπως GPU και μηχανές βίντεο (VEs), απλούς ελεγκτές DMA καθώς και πλήρη υποσυστήματα. Το SMMU μπορεί να υλοποιηθεί ως αυτόνομη συσκευή ή να ενσωματωθεί σε μια υπάρχουσα μονάδα επεξεργασίας με δυνατότητα DMA. Το διάγραμμα

δίνει μερικά παραδείγματα για το πού μπορεί να εντοπιστεί το SMMU στο σύστημα (Σχήμα 3.5).



Σχήμα 3.5: *Παραδείγματα όπου μπορεί να εντοπιστεί μια μονάδα διαχείρισης μνήμης συστήματος (SMMU) στο σύστημα. Οι συνεκτικές διασυνδέσεις εξασφαλίζουν τη συνοχή μεταξύ των κυρίων συσκευών*

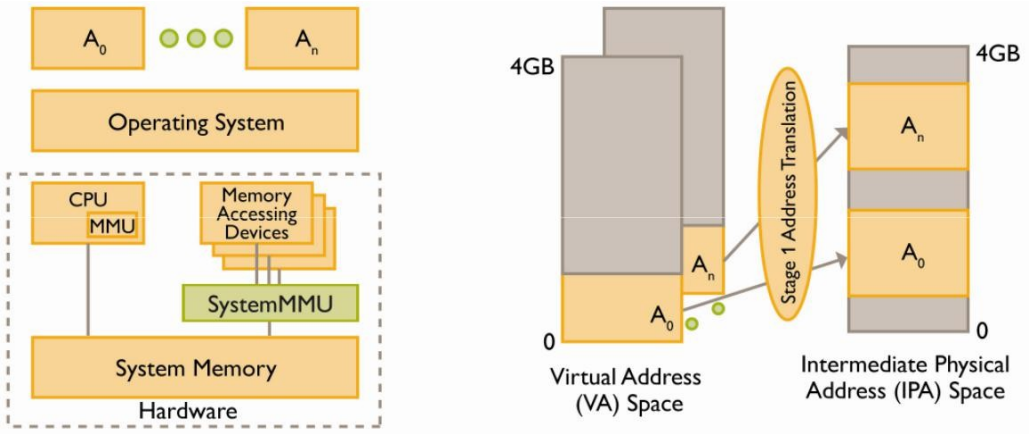
Το SMMU έχει σχεδιαστεί για χρήση σε ένα εικονικοποιημένο σύστημα όπου πολλά λειτουργικά συστήματα επισκεπτών διαχειρίζονται από ένα VMM. Για το σκοπό αυτό υποστηρίζονται δύο ξεχωριστά στάδια μετάφρασης διευθύνσεων. Το πρώτο στάδιο ή το Στάδιο 1 μεταφράζει την εικονική διεύθυνση (VA) στην ενδιάμεση φυσική διεύθυνση (IPA) και είναι σχεδιασμένο για χρήση από το λειτουργικό σύστημα επισκέπτη. Το δεύτερο στάδιο ή το Στάδιο 2 μεταφράζει την ενδιάμεση φυσική διεύθυνση (IPA) στη φυσική διεύθυνση (PA), φέρνοντας μια σειρά από οφέλη στο VMM. Οι ιδιότητες και τα οφέλη της SMMU επεκτείνονται επίσης σε μη εικονικοποιημένα συστήματα.

Ο σχεδιαστής του συστήματος έχει την επιλογή να υλοποιήσει είτε το στάδιο μετάφρασης φάσης 1 είτε το στάδιο 2 για κάθε SMMU. Για παράδειγμα, το SMMU μπορεί να τοποθετηθεί μπροστά από μια συσκευή ικανή για άμεση πρόσβαση μνήμης (DMA) η

οποία δεν διαθέτει ήδη ένα MMU, και έχει διαμορφωθεί έτσι ώστε να παρέχει μετάφραση σταδίου 1 για να επιτρέπει σε μια τέτοια συσκευή να βλέπει την κατακερματισμένη φυσική μνήμη ως συνεκτική. Το SMMU μπορεί επίσης να διαμορφωθεί για να παρέχει μόνο μεταφράσεις σταδίου 2, για παράδειγμα, για να επιτρέψει σε μια συσκευή που έχει ήδη ένα MMU να διαχειρίζεται από έναν hypervisor του συστήματος.

3.3.2.1 Μετάφραση SMMU σταδίου 1

Η μετάφραση του σταδίου 1 προορίζεται να βοηθήσει τα λειτουργικά συστήματα, τόσο όταν αυτά εκτελουνται εγγενώς όσο και μέσα σε ένα VM. Το SMMU παρέχει πολλαπλά οφέλη και λύνει αρκετά προβλήματα που σε διαφορετική περίπτωση σχετίζονται με την εφαρμογή δαπανηρών λύσεων λογισμικού. Η μετάφραση σταδίου 1 λειτουργεί παρόμοια με μια παραδοσιακή MMU της κεντρικής μονάδας επεξεργασίας (CPU). Το ακόλουθο διάγραμμα συνοψίζει την εφαρμογή και τη λειτουργικότητα της μετάφρασης διευθύνσεων 1ου σταδίου (Σχήμα 3.6).



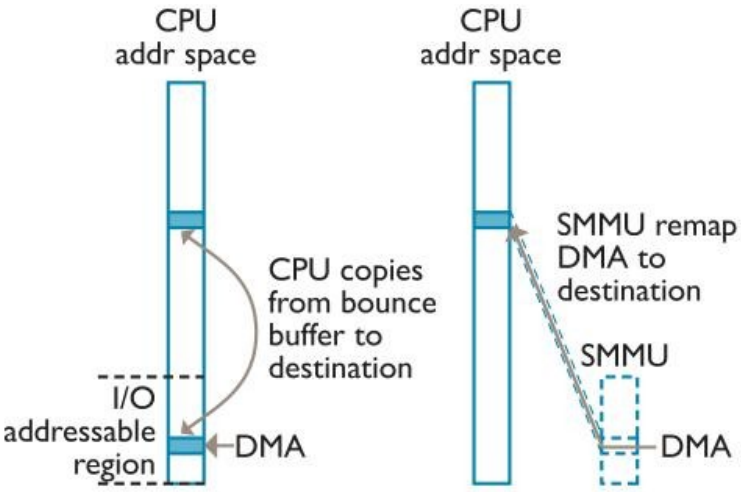
Σχήμα 3.6: *Εφαρμογή και λειτουργικότητα της μετάφρασης διευθύνσεων σταδίου 1*

Χαρακτηριστικά, τα λειτουργικά συστήματα προκαλούν μεγάλο κατακερματισμό της φυσικής μνήμης με συνεχή δέσμευση και απελευθέρωση χώρου στον σωρό, τόσο για τον πυρήνα όσο και για τις εφαρμογές. Ένα σύστημα που εφαρμόζει ένα κατακερματισμένο μοντέλο μεταξύ των χώρων IPA και PA, όπου πολλά λειτουργικά συστήματα επισκέπτη μοιράζονται το ίδιο φυσικό μέσο, θα υποφέρει ακόμα περισσότερο εξαιτίας αυτού του ζητήματος. Μια τυπική συσκευή καταναλωτών φιλοξενεί εφαρμογές που από τη φύση τους απαιτούν μεγάλες ποσότητες συνεχούς μνήμης για να λειτουργούν αποτελεσματικά. Για παράδειγμα, μια ψηφιακή φωτογραφική μηχανή χρειάζεται ένα μεγάλο χώρο για την

γρήγορη απόσπαση εικόνων που έχουν μόλις ληφθεί. Μια συσκευή Smartphone μπορεί να χρειαστεί να αναπαράγει κάποιο περιεχόμενο βίντεο, συνεπώς απαιτεί μεγάλη προσωρινή μνήμη εργασίας για τον αποκωδικοποιητή βίντεο. Είναι πολύ συνηθισμένο τα μεγάλα συνεχόμενα τμήματα φυσικής μνήμης να μην είναι διαθέσιμα για δυναμική δέσμευση λόγω κατακερματισμού της φυσικής μνήμης. Μια τυπική λύση είναι να δεσμευθούν από πριν τέτοιες μνήμες. Αυτό είναι πολύ αναποτελεσματικό δεδομένου ότι ο buffer είναι απαραίτητος μόνο κατά το χρόνο εκτέλεσης και το πρόσθετο ποσό της φυσικής μνήμης που δεσμεύεται για το σκοπό αυτό (και δεν είναι χρησιμοποιήσιμο από δραστηριότητες γενικού σκοπού του λειτουργικού συστήματος), προσθέτει απευθείας στο κόστος εξοπλισμού. Επίσης, σε μια εικονική διαμόρφωση, αυτή η λύση θα απαιτήσει άμεσες τροποποιήσεις στο VMM.

Όταν μια συσκευή με δυνατότητα άμεσης πρόσβασης μνήμης (DMA) χρειάζεται να λειτουργεί σε κατακερματισμένη φυσική μνήμη, η τυπική λύση είναι η υιοθέτηση τεχνικών λογισμικού όπως το DMA Scatter-Gather. Αυτό έχει γενικά πολυπλοκότητα και κόστος επίδοσης. Με ένα SMMU, αυτά τα κόστη μπορούν να αποφευχθούν επιτρέποντας ένα περαιτέρω επίπεδο μετάφρασης διευθύνσεων, όπου μπορούν να συγκεντρωθούν ουσιαστικά μικρότερα μπλοκ μνήμης (κάτω της τάξης μεγέθους των 4kb) στον χώρο φυσικής διεύθυνσης, ώστε να εκθέσουν ένα συνεχές μπλοκ μνήμης στον εικονικό χώρο διεύθυνσης (εάν χρησιμοποιείται σε μη εικονικοποιημένο μοντέλο) ή από το διάστημα ενδιάμεσης φυσικής διεύθυνσης έως τον εικονικό χώρο διεύθυνσης (εάν χρησιμοποιείται σε εικονικό μοντέλο).

Ένα άλλο σημαντικό ζήτημα αποκαλύπτεται όταν ορισμένες συσκευές στο σύστημα δεν μπορούν να έχουν πρόσβαση στο πλήρες φάσμα μνήμης που είναι διαθέσιμο στη CPU, όπως συσκευές 16 ή 24 bit σε αρχιτεκτονικές 32 bit ή συσκευές 32 bit σε αρχιτεκτονικές 64 bit. Παραδοσιακά, η λύση ήταν να παρέχεται μια ενδιάμεση περιοχή μνήμης σε χαμηλή διεύθυνση για να λειτουργεί ως γέφυρα. Αυτή είναι γνωστή ως προσωρινή μνήμη αναπήδησης. Το λειτουργικό σύστημα κατανέμει τις σελίδες σε ένα χώρο διευθύνσεων ορατό στη συσκευή και τις χρησιμοποιεί ως σελίδες προσωρινής αποθήκευσης για άμεση πρόσβαση DMA από και προς αυτό. Μόλις ολοκληρωθεί η είσοδος / έξοδος , το περιεχόμενο αυτών των σελίδων αντιγράφεται από τον πυρήνα στον προορισμό του, έξω από το διευθυνσιολογικό εύρος της συσκευής εισόδου / εξόδου. Υπάρχει σημαντική επιβάρυνση σε αυτή τη λειτουργία, καθώς πρόκειται για αντιγραφή τουλάχιστον μιας πλήρους σελίδας. Τα buffers αναπήδησης μπορούν να αποφευχθούν εντελώς χρησιμοποιώντας ένα SMMU (Σχήμα 3.7). Η μετάφραση σταδίου 1 μπορεί να επιτρέψει σε οποιονδήποτε παράγοντα άμεσης προσπέλασης μνήμης (DMA) να έχει πρόσβαση σε οποιαδήποτε διεύθυνση στο σύστημα χωρίς περιορισμούς που σχετίζονται με το πλάτος του διαύλου του.



Σχήμα 3.7: *Χρησιμοποιώντας το SMMU για να αποφύγουμε τα buffers αναπήδησης*

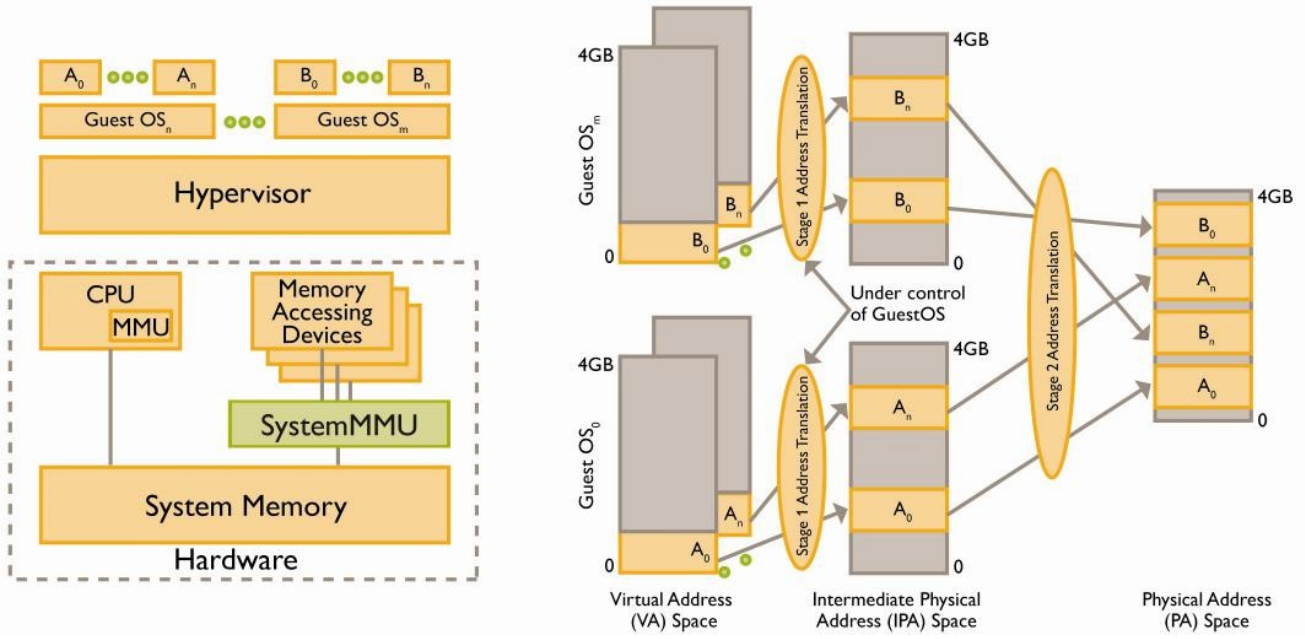
3.3.2.2 Μετάφραση SMMU σταδίου 2

Η μετάφραση του σταδίου 2 προορίζεται να ωφελήσει τους Virtual Machine Monitors. Το διάγραμμα της επόμενης σελίδας (Σχήμα 3.8) συνοψίζει την εφαρμογή και τη λειτουργικότητα της μετάφρασης διευθύνσεων 2ου σταδίου.

Η προσθήκη μιας συσκευής SMMU για τη μετάφραση σταδίου 2 καταργεί την ανάγκη για τον hypervisor να διαχειρίζεται τους πίνακες μετάφρασης σκιάς, που διατηρούσε εξ'ολοκλήρου στο λογισμικό. Η μετάφραση χώρου διευθύνσεων που υποστηρίζεται από το υλικό έχει σαφή οφέλη απόδοσης. Με τη μετάφραση διευθύνσεων σταδίου 2, το SMMU επιτρέπει σε οποιοδήποτε λειτουργικό σύστημα επισκεπτών να ρυθμίζει απευθείας όλες τις συσκευές με δυνατότητα άμεσης πρόσβασης μνήμης (DMA) στο σύστημα, καθώς μοιράζεται μαζί τους τον ίδιο χώρο διευθύνσεων σε επίπεδο ενδιάμεσων διευθύνσεων (IPA). Το SMMU μπορεί επίσης να ρυθμιστεί έτσι ώστε να διασφαλίζεται ότι οι συσκευές που λειτουργούν για λογαριασμό ενός λειτουργικού συστήματος επισκέπτη θα εμποδίζονται να καταστρέψουν τη μνήμη ενός άλλου λειτουργικού συστήματος επισκέπτη. Το σύστημα μετάφρασης διευθύνσεων σταδίου 2 βασίζεται σε ένα αλγόριθμο μετάφρασης με έναν περιγραφέα 64 bit, για να του επιτρέπεται να απευθύνεται σε μια φυσική μνήμη μεγαλύτερη από 4GBytes.

Ο διαχωρισμός υλικού μεταξύ των δύο σταδίων της μετάφρασης διευθύνσεων επιτρέπει τον σαφή καθορισμό του επιπέδου λογισμικού που διαχειρίζεται μία διεύθυνση, μεταξύ του guest OS (Στάδιο 1) και του VMM (Στάδιο 2). Επομένως, τα σφάλματα

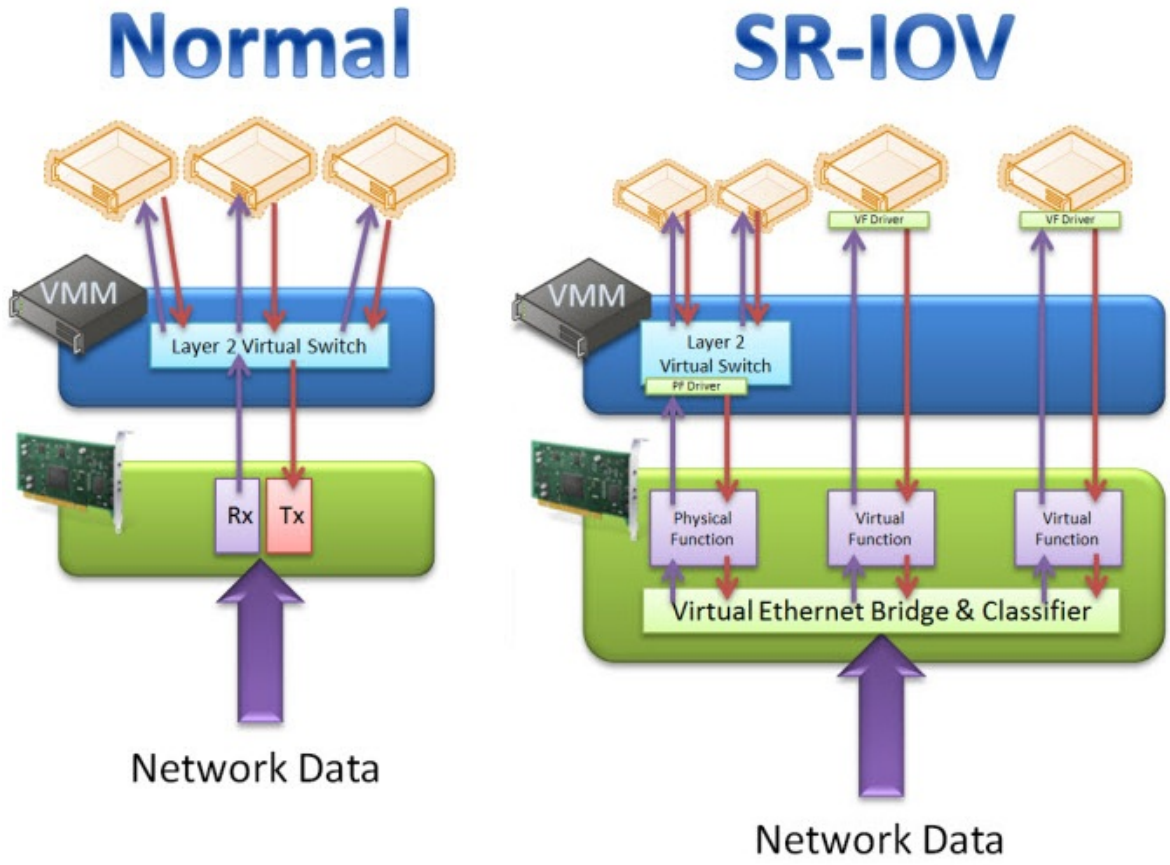
μετάφρασης μπορούν να μεταφερθούν από το υλικό, στο κατάλληλο επίπεδο λογισμικού, επιτρέποντας έτσι τον χειρισμό των λειτουργιών διαχείρισης (διαχείριση TLB, ενεργοποίηση MMU, διαμορφώσεις καταχωρητών) στο κατάλληλο στάδιο της διαδικασίας μετάφρασης διεύθυνσης μνήμης. Αυτό μπορεί να βελτιώσει την απόδοση μειώνοντας σημαντικά τον αριθμό καταχωρήσεων στο VMM.



Σχήμα 3.8: *Εφαρμογή και λειτουργικότητα της μετάφρασης διευθύνσεων σταδίου 2*

3.4 Τεχνολογία SR-IOV

Η τεχνολογία SR-IOV είναι τεχνολογία εικονικοποίησης υλικού που διευκολύνει την εικονικοποίηση δικτύου. Συγκεκριμένα η εικονικοποίηση εισόδου / εξόδου μονής διαδρομής (Single Root Input-Output Virtualization - SR-IOV) είναι μια διεπαφή δικτύου που επιτρέπει την απομόνωση των πόρων PCI Express ώστε να βελτιώνεται η διαχειρισιμότητα και η απόδοση. Χρησιμοποιώντας τις προδιαγραφές SR-IOV μια ενιαία φυσική PCI Express μπορεί να διαμοιραστεί σε διαφορετικά εικονικά στοιχεία ενός εικονικού περιβάλλοντος (Σχήμα 3.9). Το SR-IOV προσφέρει διαφορετικές εικονικές λειτουργίες σε διαφορετικά εικονικά στοιχεία (όπως σε έναν προσαρμογέας δικτύου) σε μια μηχανή φυσικού διακομιστή. Η τεχνολογία SR-IOV αρχικά υλοποιήθηκε από την Intel.



Σχήμα 3.9: *Η τεχνολογία εικονικοποίησης δικτύου SR-IOV (Single Root Input—Output Virtualization). Ο οδηγός της εικονικής μηχανής, εξυπηρετεί μέσω των επεκτάσεων SR-IOV, αιτήσεις από το δίκτυο απευθείας.*

ΚΕΦΑΛΑΙΟ 4ο

Λογισμικό Εικονικοποίησης - Προσομοίωσης

4.1 Λογισμικό QEMU/KVM

Το QEMU (συντόμευση για το Quick Emulator) είναι ένας ελεύθερος και ανοιχτού κώδικα hypervisor εγκατεστημένος στον host που εκτελεί εικονικοποίηση υλικού. Το QEMU είναι ένας επόπτης εικονικής μηχανής εγκατεστημένος στον host: εξομοιώνει CPUs μέσω δυναμικής μετάφρασης γλώσσας μηχανής και παρέχει ένα σύνολο μοντέλων συσκευών, που του επιτρέπουν να τρέχει μια ποικιλία από μη τροποποιημένα λειτουργικά συστήματα επισκέπτη. Μπορεί επίσης να χρησιμοποιηθεί μαζί με το KVM για να εκτελέσει εικονικές μηχανές σε σχεδόν εγγενή ταχύτητα (απαιτώντας επεκτάσεις εικονικοποίησης υλικού σε υπολογιστές x86). Το QEMU μπορεί επίσης να κάνει εξομοίωση CPU για διεργασίες σε επίπεδο χρήστη, επιτρέποντας σε εφαρμογές που έχουν συνταχθεί για μια αρχιτεκτονική να τρέχουν σε άλλη.

Το QEMU γράφτηκε από τον Fabrice Bellard και είναι ελεύθερο λογισμικό και έχει άδεια κυρίως υπό την GNU General Public License (GPL). Διάφορα τμήματα κυκλοφορούν υπό την άδεια BSD, GNU Lesser General Public License (LGPL) ή άλλες άδειες συμβατές με GPL.

4.1.1 Καταστάσεις λειτουργίας QEMU

Το QEMU έχει πολλούς τρόπους λειτουργίας:

-Προσομοίωση κατάστασης λειτουργίας χρήστη:

Σε αυτή τη λειτουργία το QEMU εκτελεί μόνο προγράμματα Linux ή macOS που έχουν συνταχθεί για διαφορετικό σύνολο εντολών. Οι κλήσεις συστήματος ελέγχονται εξωτερικά για να αποφεύγονται δυσαρμονίες endianess και αναντιστοιχίες 32/64 bit. Η γρήγορη μεταγλώττιση και αποσφαλμάτωση προγραμμάτων διαφορετικών αρχιτεκτονικών είναι οι κύριοι στόχοι χρήσης της εξομοίωσης της κατάστασης λειτουργίας χρήστη.

-Εξομοίωση συστήματος

Σε αυτή τη λειτουργία, το QEMU εξομοιώνει ένα πλήρες σύστημα υπολογιστή, συμπεριλαμβανομένων των περιφερειακών. Μπορεί να χρησιμοποιηθεί για την παροχή φιλοξενίας πολλών εικονικών υπολογιστών σε έναν μόνο υπολογιστή. Το QEMU μπορεί να εκκινήσει πολλά λειτουργικά συστήματα επισκεπτών, συμπεριλαμβανομένων των Linux, Solaris, Microsoft Windows, DOS και BSD. Υποστηρίζει την προσομοίωση πολλών συνόλων εντολών, συμπεριλαμβανομένων των x86, MIPS, ARMv7, ARMv8, PowerPC, SPARC, ETRAX CRIS MicroBlaze.

-KVM Hosting

Εδώ το QEMU ασχολείται με τη δημιουργία και τη μετανάστευση εικόνων εικονικών μηχανων για το KVM. Συμμετέχει ακόμα στην εξομοίωση υλικού, αλλά η εκτέλεση του επισκέπτη γίνεται από το KVM όπως ζητήθηκε από το QEMU.

-Xen Hosting

Το QEMU ασχολείται μόνο με την εξομοίωση υλικού. Η εκτέλεση του επισκέπτη γίνεται εντός Xen και είναι εντελώς κρυμμένη από το QEMU.

4.2 Το KVM αναλυτικοτερα

Η βασισμένη στο πυρήνα εικονική μηχανή (kernel virtual machine - KVM) είναι μια υποδομή εικονικοποίησης για τον πυρήνα του Linux που τον μετατρέπει σε έναν hypervisor. Ήταν συγχωνευμένη στην κεντρική γραμμή του πυρήνα του Linux στην έκδοση πυρήνα 2.6.20, η οποία κυκλοφόρησε στις 5 Φεβρουαρίου 2007. Το KVM απαιτεί επεξεργαστή με επεκτάσεις εικονικοποίησης υλικού. Το KVM μεταφέρθηκε επίσης σε FreeBSD με τη μορφή φορητών ενοτήτων (module) πυρήνα.

Το KVM αρχικά υποστήριζε επεξεργαστές x86 και έχει μεταφερθεί σε S/390, PowerPC, και IA-64. Ένα port για ARM ενσωματώθηκε κατά τη συγχώνευση της 3.9 έκδοσης πυρήνα.

Μια μεγάλη ποικιλία από λειτουργικά συστήματα επισκεπτών λειτουργούν με KVM, συμπεριλαμβανομένων πολλών γεύσεων και εκδόσεων Linux, BSD, Solaris, Windows, Haiku, ReactOS, Plan 9, λειτουργικό σύστημα έρευνας AROS και OS X. Επιπλέον, είναι γνωστό ότι, τα λειτουργικά συστήματα Android 2.2, GNU/Hurd (Debian K16), Minix 3.1.2a, Solaris 10 U3 και Darwin 8.0.1 μαζί με άλλα λειτουργικά συστήματα και ορισμένες νεότερες εκδόσεις αυτών, δουλεύουν, με συγκεκριμένους περιορισμούς.

Η υποστήριξη παραεικονικοποίησης (paravirtualization) για ορισμένες συσκευές είναι διαθέσιμη για Linux, OpenBSD, FreeBSD, NetBSD, Plan 9 και τους επισκέπτες των Windows που χρησιμοποιούν το API VirtIO. Αυτό υποστηρίζει μια παραεικονικοποιημένη κάρτα Ethernet, έναν παραεικονικοποιημένο ελεγκτή εισόδου/εξόδου δίσκου, μια συσκευή μπαλονιού για τη ρύθμιση της χρήσης της μνήμης των επισκεπτών και μια διασύνδεση γραφικών VGA, χρησιμοποιώντας προγράμματα οδήγησης SPICE ή Vmware.

4.2.1 Εσωτερικά

Από μόνο του, το KVM δεν εκτελεί καμία εξομοίωση. Αντίθετα, εκθέτει τη διεπαφή /dev/kvm, την οποία μπορεί στη συνέχεια να χρησιμοποιήσει ένας οικοδεσπότης χώρου χρήστη ώστε να επιτυγχάνονται τα παρακάτω (Σχημα 4.1) :

-Ρύθμιση του χώρου διευθύνσεων της εικονικης μηχανής του επισκέπτη. Ο οικοδεσπότης πρέπει επίσης να παρέχει ένα firmware image (συνήθως ένα προσαρμοσμένο BIOS, όταν εξομοιώνονται ολόκληροι υπολογιστές) που μπορεί να χρησιμοποιήσει ο

επισκέπτης ώστε να μπορεί να ξεκινήσει ασφαλώς και “ελαφριά” (bootstrap) το βασικό του λειτουργικό σύστημα.

-Διαχείριση της προσομοίωσης εισόδου/εξόδου του επισκέπτη.

-Να απεικονίζεται η εικόνα του επισκέπτη πίσω στον οικοδεσπότη.

Στο Linux, οι εκδόσεις του QEMU 0.10.1 και οι νεότερες αποτελούν έναν τέτοιο οικοδεσπότη χώρου χρήστη. Το QEMU χρησιμοποιεί KVM όταν είναι διαθέσιμο για να εικονικοποιήσει τους επισκέπτες σε σχεδόν εγγενείς ταχύτητες, ενώ σε άλλη περίπτωση επιστρέφει στην εξομοίωση μόνο με λογισμικό.

Εσωτερικά, το KVM χρησιμοποιεί το SeaBIOS ως υλοποίηση ανοιχτού κώδικα ενός BIOS 16-bit x86.

4.2.2 Αδειοδότηση

Τα τμήματα του KVM έχουν ώς άδεια χρήσης διάφορες άδειες GNU:

-Τμήμα πυρήνα KVM: GPL v2

-Τμήμα χρήστη KVM: LGPL v2

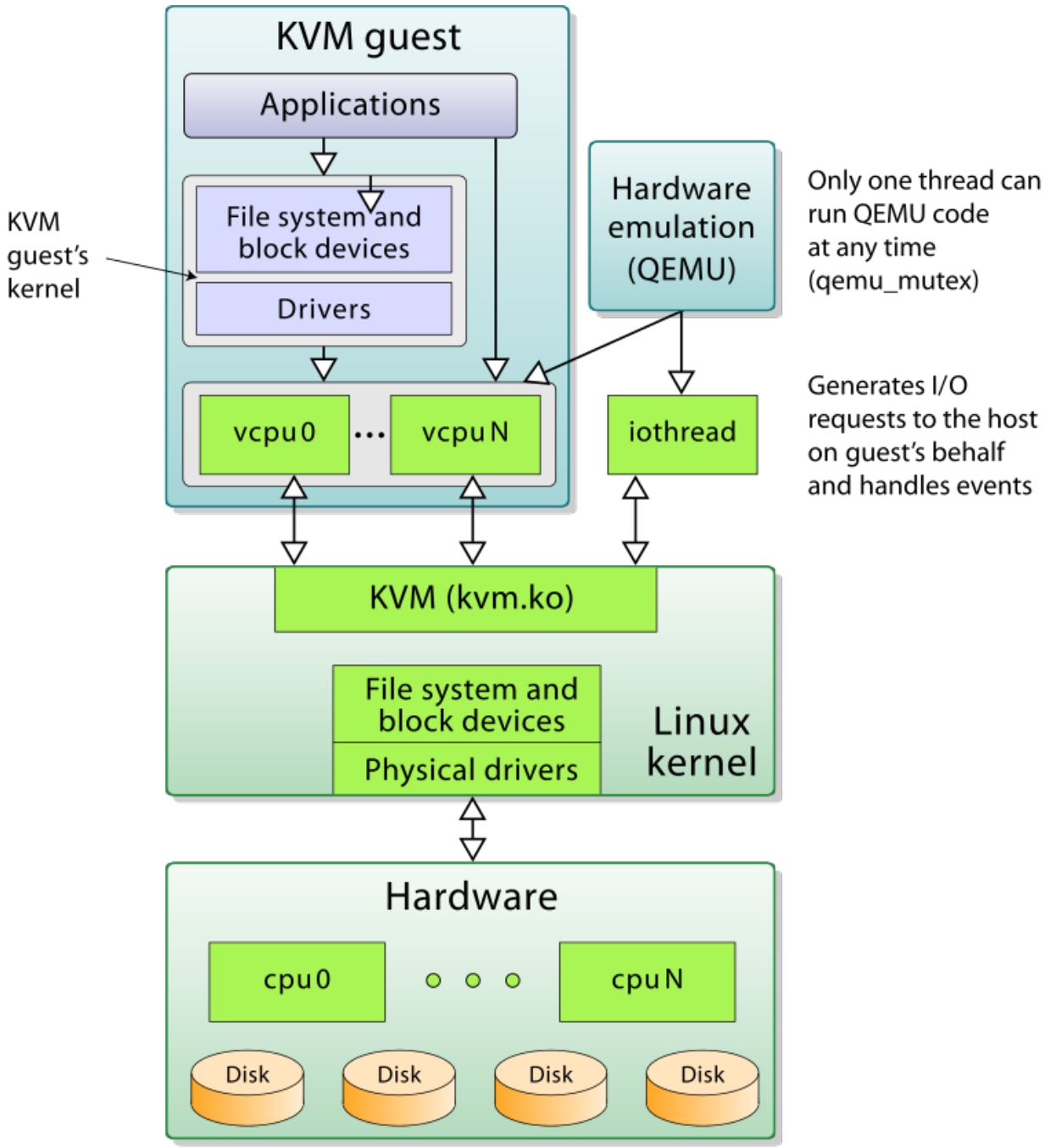
-Εικονική κεντρική βιβλιοθήκη επεξεργαστών CPU (libqemu.a) και εξομοιωτής συστήματος PC QEMU: LGPL

- Εξομοιωτής QEMU λειτουργίας χρήστη Linux: GPL

-Αρχεία BIOS (bios.bin, vgabios.bin και vgabios-cirrus.bin): LGPL v2 ή νεότερη έκδοση

4.2.3 Ιστορία

Ο Avi Kivity ξεκίνησε την ανάπτυξη του KVM στην Qumranet, μία start-up εταιρεία τεχνολογίας που εξαγοράστηκε από την Red Hat το 2008. Το KVM συγχωνεύθηκε στην κύρια γραμμή του πυρήνα του Linux στην έκδοση πυρήνα 2.6.20, η οποία κυκλοφόρησε στις 5 Φεβρουαρίου 2007. Το KVM διατηρείται από τον Paolo Bonzini.



Σχήμα 4.1: *Μια επισκόπηση υψηλού επιπέδου για το περιβάλλον εικονικοποίησης KVM / QEMU*

4.2.4 Τα σημαντικότερα KVM ioctls

Το KVM είναι μια λύση πλήρους εικονικοποίησης για το Linux σε υλικό που περιέχει επεκτάσεις εικονικοποίησης (π.χ. Intel VT , AMD-V). Αποτελείται από ένα kernel module με δυνατότητα φόρτωσης, kvm.ko, το οποίο παρέχει την βασική υποδομή εικονικοποίησης και ένα συγκεκριμένο module επεξεργαστή, για παράδειγμα kvm-intel.ko ή kvm-amd.ko.

Τα σημαντικότερα ioctls του KVM kernel module είναι τα εξής:

1) KVM_GET_API_VERSION

Δυνατότητα: βασική

Αρχιτεκτονικές: όλες

Τύπος: ioctl συστήματος

Παράμετροι: καμία

Επιστρέφει: τη σταθερά KVM_API_VERSION (= 12)

Αυτό το ioctl αναγνωρίζει την έκδοση API ως το σταθερό kvm API. Δεν αναμένεται ότι αυτός ο αριθμός θα αλλάξει. Ωστόσο, τα Linux 2.6.20 και 2.6.21 αναφέρουν προηγούμενες εκδόσεις. Αυτές δεν είναι τεκμηριωμένες και δεν υποστηρίζονται. Οι αιτήσεις θα πρέπει να αρνούνται να εκτελεστούν εάν το KVM_GET_API_VERSION επιστρέψει μια τιμή διαφορετική από 12. Εάν ο έλεγχος αυτός περάσει, όλα τα ioctls που περιγράφονται ως «βασικά» θα είναι διαθέσιμα.

2) KVM_CREATE_VM

Δυνατότητα: βασική

Αρχιτεκτονικές: όλες

Τύπος: ioctl συστήματος

Παράμετροι: αναγνωριστικό τύπου μηχανής (KVM_VM_ *)

Επιστρέφει: ένα VM περιγραφητή αρχείου (fd) που μπορεί να χρησιμοποιηθεί για τον έλεγχο της νέας εικονικής μηχανής.

Το νέο VM δεν διαθέτει εικονικά cpus και καμία μνήμη. Ένα mmap() ενός VM fd θα έχει πρόσβαση στον φυσικό χώρο διεύθυνσης της εικονικής μηχανής. Το offset μηδέν αντιστοιχεί στη φυσική διεύθυνσή μηδέν του επισκέπτη. Η χρήση του mmap() σε ένα VM fd αποθαρρύνεται εάν είναι διαθέσιμη η δέσμευση μνήμης επιπέδου χώρου χρήστη (KVM_CAP_USER_MEMORY). Χρησιμοποιείται το 0 ως τύπος μηχανής.
Για να δημιουργηθούν εικονικές μηχανές ελεγχόμενες από το χρήστη στο S390, χρησιμοποιείται το KVM_CAP_S390_UCONTROL και η σημαία KVM_VM_S390_UCONTROL από προνομιούχο χρήστη (CAP_SYS_ADMIN).

3) KVM_CHECK_EXTENSION

Δυνατότητα: βασική, KVM_CAP_CHECK_EXTENSION_VM για vm ioctl

Αρχιτεκτονικές: όλες

Τύπος: ioctl συστήματος, vm ioctl

Παράμετροι: αναγνωριστικό επέκτασης (KVM_CAP_ *)

Επιστρέφει: 0 εάν δεν υποστηρίζεται, 1 (ή κάποιος άλλος θετικός ακέραιος αριθμός) εάν υποστηρίζεται

Το API επιτρέπει στην εφαρμογή να ρωτάει για επεκτάσεις του κεντρικού kvm API. Ο χώρος χρήστη περνάει ένα αναγνωριστικό επέκτασης (έναν ακέραιο αριθμό) και λαμβάνει έναν ακέραιο αριθμό που περιγράφει τη διαθεσιμότητα της επέκτασης. Γενικά το 0 σημαίνει όχι και το 1 σημαίνει ναι, αλλά μερικές επεκτάσεις μπορούν να αναφέρουν πρόσθετες πληροφορίες στην τιμή του ακέραιου αριθμού επιστροφής.
Με βάση την αρχικοποίησή τους, τα διαφορετικά VM μπορούν να έχουν διαφορετικές δυνατότητες. Συνεπώς, ενθαρρύνεται η χρήση αυτού του vm ioctl για την αναζήτηση των δυνατοτήτων αυτών (χρησιμοποιώντας το KVM_CAP_CHECK_EXTENSION_VM στο vm fd).

4) KVM_GET_VCPU_MMAP_SIZE

Δυνατότητα: βασική

Αρχιτεκτονικές: όλες

Τύπος: ioctl συστήματος

Παράμετροι: καμία

Επιστρέφει: μέγεθος της περιοχής vcpu mmap, σε bytes

Το KVM_RUN ioctl επικοινωνεί με το χώρο χρήστη μέσω μίας περιοχής κοινής μνήμης. Το KVM_GET_VCPU_MMAP_SIZE ioctl επιστρέφει το μέγεθος αυτής της περιοχής. Στην περιγραφή του KVM_RUN δίνονται περισσότερες λεπτομέρειες.

5) KVM_CREATE_VCPU

Δυνατότητα: βασική

Αρχιτεκτονικές: όλες

Τύπος: vm ioctl

Παράμετροι: vcpu id (apic id σε x86)

Επιστρέφει: vcpu fd στην επιτυχία, -1 στο σφάλμα

Αυτό το API προσθέτει ένα vcpu σε μια εικονική μηχανή. Δεν είναι δυνατή η προσθήκη περισσότερων από το max_vcpus. Το vcpu id είναι ένας ακέραιος στην περιοχή [0, max_vcpu_id). Η συνιστώμενη τιμή max_vcpus μπορεί να ανακτηθεί χρησιμοποιώντας το KVM_CAP_NR_VCPUS του KVM_CHECK_EXTENSION ioctl κατά τη διάρκεια εκτέλεσης. Η μέγιστη δυνατή τιμή για το max_vcpus μπορεί να ανακτηθεί χρησιμοποιώντας το KVM_CAP_MAX_VCPUS του KVM_CHECK_EXTENSION ioctl κατά τη διάρκεια εκτέλεσης.

Εάν το KVM_CAP_NR_VCPUS δεν υπάρχει, θα πρέπει να υποτεθεί ότι το max_vcpus είναι 4 cpus το μέγιστο. Εάν το KVM_CAP_MAX_VCPUS δεν υπάρχει, θα πρέπει να υποτεθεί ότι το max_vcpus είναι ίδιο με την τιμή που επιστρέφεται από το KVM_CAP_NR_VCPUS. Η μέγιστη δυνατή τιμή για το max_vcpu_id μπορεί να ανακτηθεί χρησιμοποιώντας το KVM_CAP_MAX_VCPU_ID του KVM_CHECK_EXTENSION ioctl κατά το χρόνο εκτέλεσης. Εάν το KVM_CAP_MAX_VCPU_ID δεν υπάρχει, πρέπει να υποτεθεί ότι το max_vcpu_id είναι το ίδιο με την τιμή που επιστρέφεται από το KVM_CAP_MAX_VCPUS.

Στο PowerPC χρησιμοποιώντας τη λειτουργία book3s_hv, τα vcpus αντιστοιχίζονται σε εικονικά νήματα σε έναν ή περισσότερους εικονικούς πυρήνες CPU. (Αυτό οφείλεται στο γεγονός ότι το υλικό απαιτεί όλα τα νήματα υλικού ενός πυρήνα CPU, ο οποίος θα “τρέξει” και τον εικονικό στην ουσία καθώς μιλάμε για εικονικοποίηση υποβοηθούμενη από το υλικό, να βρίσκονται στο ίδιο λογικό διαμέρισμα.) Η δυνατότητα KVM_CAP_PPC_SMT υποδεικνύει τον αριθμό των vcpus ανά εικονικό πυρήνα (vcore). Το αναγνωριστικό vcore αποκτάται διαιρώντας το vcpu id με τον αριθμό vcpus ανά vcore. Τα vcpus σε ένα δεδομένο vcore θα είναι πάντα στον ίδιο φυσικό πυρήνα (αν και αυτός μπορεί να είναι ένας διαφορετικός φυσικός πυρήνας από καιρό σε καιρό). Ο χώρος του χρήστη

μπορεί να ελέγξει την ταυτόχρονη πολυνηματική κατάσταση λειτουργίας (SMT) του επισκέπτη με την δέσμευση αναγνωριστικών vcpu. Για παράδειγμα, εάν ο χώρος χρήστη θέλει vcpus με ένα νήμα, θα πρέπει όλα τα vcpu IDs να είναι πολλαπλάσια του αριθμού των vcpus ανά vcore. Για τα εικονικά cpus που έχουν δημιουργηθεί σε εικονικές μηχανές χρήστη αρχιτεκτονικής S390, το προκύπτον vcpu fd αντιστοιχιζεται πάνω στο offset σελίδας KVM_S390_SIE_PAGE_OFFSET. Έτσι θα απεικονιστεί πάνω στο μπλοκ ελέγχου υλικού του εικονικού cpu.

6) KVM_RUN

Δυνατότητα: βασική

Αρχιτεκτονικές: όλες

Τύπος: vcpu ioctl

Παράμετροι: καμία

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

Σφάλματα:
EINTR: εκκρεμεί ένα unmasked signal

Αυτό το ioctl χρησιμοποιείται για να τρέξει ένα εικονικό επεξεργαστή του επισκέπτη. Παρόλο που δεν υπάρχουν ρητές παράμετροι, υπάρχει ένα σιωπηρό μπλοκ παραμέτρων που μπορεί να ληφθεί με την απεικόνιση του vcpu fd στο offset 0, και με μέγεθος αυτό που δίνεται από το KVM_GET_VCPU_MMAP_SIZE. Το μπλοκ παραμέτρων διαμορφώνεται ως 'struct kvm_run' .

7) KVM_GET_REGS

Δυνατότητα: βασική

Αρχιτεκτονικές: όλες εκτός από ARM, arm64

Τύπος: vcpu ioctl

Παράμετροι: struct kvm_regs (out)

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

Διαβάζει τους γενικούς καταχωρητές από τη vcpu.

```
/ * x86 * /
struct kvm_regs {

 / * out (KVM_GET_REGS) / in (KVM_SET_REGS) * /
      __u64 rax, rbx, rcx, rdx;
      __u64 rsi, rdi, rsp, rbp;
      __u64 r8, r9, r10, r11,
      __u64 r12, r13, r14, r15;
      __u64 rip, rflags;
};

/ * mips * /
struct kvm_regs {
 / * out (KVM_GET_REGS) / in (KVM_SET_REGS) * /
      __u64 gpr [32].
      __u64 hi;
      __u64 lo;
      __u64 pc;
};
```

8) KVM_SET_REGS

Δυνατότητα: βασική

Αρχιτεκτονικές: όλες εκτός από ARM, arm64

Τύπος: vcpu ioctl

Παράμετροι: struct kvm_regs (in)

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

Γράφει τους καταχωρητές γενικής χρήσης της vcpu. Οι δομές δεδομένων φαίνονται παραπάνω.

9) KVM_GET_SREGS

Δυνατότητα: βασική

Αρχιτεκτονικές: x86, ppc

Τύπος: vcpu ioctl

Παράμετροι: struct kvm_sregs (out)

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

Διαβάζει ειδικούς καταχωρητές από τη vcpu.

```
/* x86 */
struct kvm_sregs {
        struct kvm_segment cs, ds, es, fs, gs, ss;
        struct kvm_segment tr, ldt;
        struct kvm_dtable gdt, idt;
        __u64 cr0, cr2, cr3, cr4, cr8;
        __u64 efer;
        __u64 apic_base;
        __u64 interrupt_bitmap[(KVM_NR_INTERRUPTS + 63) / 64];
};

/* ppc -- see arch/powerpc/include/uapi/asm/kvm.h */
```

Το interrupt_bitmap είναι το bitmap των εκκρεμών εξωτερικών διακοπών. Μπορεί να οριστεί το πολύ ένα bit. Η εκκρεμής διακοπή έχει αναγνωριστεί από το APIC αλλά δεν έχει αρχίσει να εκτελείται στον πυρήνα του επεξεργαστή.

10) KVM_SET_SREGS

Δυνατότητα: βασική

Αρχιτεκτονικές: x86, ppc

Τύπος: vcpu ioctl

Παράμετροι: struct kvm_sregs (in)

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

Γράφει ειδικούς καταχωρητές στη vcpu. Η δομή δεδομένων φαίνεται παραπάνω.

11) KVM_GET_DEBUGREGS

Δυνατότητα: KVM_CAP_DEBUGREGS

Αρχιτεκτονικές: x86

Τύπος: vm ioctl

Παράμετροι: struct kvm_debugregs (out)

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

Διαβάζει καταχωρητές εντοπισμού σφαλμάτων από τη vcpu.

```
struct kvm_debugregs {
        __u64 db[4];
        __u64 dr6;
        __u64 dr7;
        __u64 flags;
        __u64 reserved[9];
};
```

12) KVM_SET_DEBUGREGS

Δυνατότητα: KVM_CAP_DEBUGREGS

Αρχιτεκτονικές: x86

Τύπος: vm ioctl

Παράμετροι: struct kvm_debugregs (in)

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

Γράφει καταχωρητές εντοπισμού σφαλμάτων στη vcpu. Η δομή δεδομένων φαίνεται παραπάνω. Το πεδίο flags δεν χρησιμοποιείται ακόμα και πρέπει να είναι 0 κατά την είσοδο.

13) KVM_SET_USER_MEMORY_REGION

Δυνατότητα: KVM_CAP_USER_MEM

Αρχιτεκτονικές: όλες

Τύπος: vm ioctl

Παράμετροι: struct kvm_userspace_memory_region (in)

Επιστρέφει: 0 στην επιτυχία, -1 στο σφάλμα

```
struct kvm_userspace_memory_region {
        __u32 slot;
        __u32 flags;
        __u64 guest_phys_addr;
        __u64 memory_size; /* bytes */
        __u64 userspace_addr; /* start of the userspace allocated memory
*/
};

/* for kvm_memory_region::flags */
#define KVM_MEM_LOG_DIRTY_PAGES (1UL << 0)
#define KVM_MEM_READONLY        (1UL << 1)
```

Αυτό το ioctl επιτρέπει στο χρήστη να δημιουργεί ή να τροποποιεί μια υποδοχή (slot) φυσικής μνήμης. Όταν αλλάζει μια υπάρχουσα υποδοχή, μπορεί μόνο να μετακινηθεί μέσα στον χώρο φυσικής μνήμης του επισκέπτη ή να τροποποιηθούν οι σημαίες της. Δεν μπορεί να γίνει αλλαγή μεγέθους. Οι υποδοχές πρέπει να μην αλληλεπικαλύπτονται στο χώρο φυσικής μνήμης του επισκέπτη.

Τα δυαδικά ψηφία 0-15 της "slot" καθορίζουν το id της υποδοχής και αυτή η τιμή θα πρέπει να είναι μικρότερη από τον μέγιστο αριθμό υποδοχών μνήμης που υποστηρίζονται ανά εικονική μηχανή. Ο μέγιστος αριθμός επιτρεπόμενων υποδοχών μπορεί να αναζητηθεί χρησιμοποιώντας το KVM_CAP_NR_MEMSLOTS, αν αυτή η δυνατότητα υποστηρίζεται από την αρχιτεκτονική.

Εάν είναι διαθέσιμο το KVM_CAP_MULTI_ADDRESS_SPACE, τα bits 16-31 της "slot" καθορίζουν τον χώρο διευθύνσεων που μπορεί να αλλάξει. Η τιμή της διεύθυνσης που σχηματίζεται από αυτά τα bit πρέπει να είναι μικρότερη από την τιμή που επιστρέφει το KVM_CHECK_EXTENSION για την ικανότητα KVM_CAP_MULTI_ADDRESS_SPACE. Οι υποδοχές μνήμης σε ξεχωριστούς χώρους διευθύνσεων δεν σχετίζονται. Ο περιορισμός των επικαλυπτόμενων υποδοχών μνήμης ισχύει μόνο μέσα σε κάθε ένα χώρο διεύθυνσης.

Η μνήμη για την περιοχή λαμβάνεται ξεκινώντας από τη διεύθυνση που δηλώνεται στο πεδίο userpace_addr, η οποία πρέπει να δείχνει σε μνήμη διευθυνσιοδοτούμενη από το χρήστη για όλο το μέγεθος της υποδοχής. Οποιοδήποτε αντικείμενο μπορεί να αποτελεί αυτή τη μνήμη, συμπεριλαμβανομένης ανώνυμης μνήμης, συνηθισμένων αρχείων και hugetlbfs (huge pages). Συνιστάται τα χαμηλότερα 21 bits των guest_phys_addr και userspace_addr να είναι ταυτόσημα. Αυτό επιτρέπει στις μεγάλες σελίδες του επισκέπτη να υποστηρίζονται από μεγάλες σελίδες στον οικοδεσπότη.

Το πεδίο flags υποστηρίζει δύο σημαίες: KVM_MEM_LOG_DIRTY_PAGES και KVM_MEM_READONLY. Η πρώτη μπορεί να ρυθμιστεί ώστε να δίνει εντολή στο KVM να παρακολουθεί τις εγγραφές στη μνήμη μέσα στην υποδοχή μνήμης. Το ioctl KVM_GET_DIRTY_LOG χρησιμοποιείται για να παρακολουθούνται οι εγγραφες αυτές. Η δευτερη μπορεί να ρυθμιστεί, αν το επιτρέπει η δυνατότητα KVM_CAP_READONLY_MEM, ώστε να κάνει μια νέα υποδοχή μνήμης μόνο-για-ανάγνωση (read-only). Σε αυτή την περίπτωση, η εγγραφή σε αυτή τη μνήμη θα αναρτηθεί στο χώρο του χρήστη ως έξοδος KVM_EXIT_MMIO.

Όταν η δυνατότητα KVM_CAP_SYNC_MMU είναι διαθέσιμη, οι αλλαγές στην πίσω πλευρά της περιοχής μνήμης (στον οικοδεσπότη) αντανακλώνται αυτόματα στον επισκέπτη. Για παράδειγμα, ένα mmap() που επηρεάζει την περιοχή θα γίνει άμεσα ορατό. Ένα άλλο παράδειγμα είναι το madvise (MADV_DROP). Συνιστάται να χρησιμοποιείται αυτό το API αντί του KVM_SET_MEMORY_REGION ioctl. Το KVM_SET_MEMORY_REGION δεν επιτρέπει τον λεπτομερή έλεγχο της δέσμευσης της μνήμης και έχει καταργηθεί.

4.3 XEN

Το **XEN** είναι ένα λογισμικό για την παροχή υπηρεσιών εικονικοποίησης. Ο hypervisor του XEN δημιουργήθηκε, στα πλαίσια της ερευνητικής εργασίας XENOLINUX, στο Πανεπιστήμιο του Cambridge από τους Keir Fraser και Ian Pratt στα τέλη της δεκαετίας του 1990. Το Πανεπιστήμιο ανέπτυξε τις πρώτες εκδόσεις του λογισμικού ως το 2012. Έκτοτε η κοινότητα του XEN συντηρεί και αναπτύσσει το εν λόγω λογισμικό ως ελεύθερο λογισμικό υπό την άδεια GNU General Public License (GPLv2).

4.3.1 Οι οντότητες του XEN

Οι βασικές οντότητες που υπάρχουν στο XEN είναι οι ακόλουθες :

-O hypervisor

Ο hypervisor του XEN είναι ένα "στρώμα" λογισμικού το οποίο προσφέρει υπηρεσίες εικονικοποίησης, και "τρέχει" απευθείας στο υλικό αντικαθιστώντας το λειτουργικό σύστημα. Λόγω της θέσης που κατέχει ο hypervisor αποτελεί τον εξυπηρετητή όλων των κλήσεων προς το υλικό (CPU, είσοδος/έξοδος). Η χρήση του hypervisor αποτελεί θετικό στοιχείο για την αρχιτεκτονική του XEN καθώς υπάρχει ουσιαστικός διαχωρισμός του υλικού από τα φιλοξενούμενα λειτουργικά συστήματα, με άμεση συνέπεια την αύξηση του επιπέδου ασφαλείας του συστήματος (πιθανές επιθέσεις στα φιλοξενούμενα λειτουργικά συστήματα δεν θα επηρεάσουν τον κεντρικό εξυπηρετητή).

-Domain0 guest

Το Domain0, που αναφέρεται και ως Dom0 , ενεργοποιείται από τον hypervisor του XEN κατά την αρχική εκκίνηση του συστήματος και μπορεί να τρέξει οποιοδήποτε λειτουργικό σύστημα εκτός από Windows. (Στην ουσία με άλλους όρους θα λέγαμε ότι εκεί εκτελείται το λειτουργικό σύστημα του οικοδεσπότη).

-DomU Guests

Τα φιλοξενούμενα domain που αναφέρονται και ως DomU , ελέγχονται από το Dom0 και λειτουργούν ξεχωριστά στο σύστημα. Τα φιλοξενούμενα Domain είτε τρέχουν με ένα ειδικά τροποποιημένο λειτουργικό σύστημα, κάτι το οποίο είναι γνωστό σαν

paravirtualization, είτε με ένα μη-τροποποιημένο λειτουργικό σύστημα, το οποίο τρέχει σε ειδικό υλικό που το υποστηρίζει (επεξεργαστές Intel VT, AMD-V), κάτι το οποίο είναι γνωστό σαν εικονική μηχανή υλικού (Hardware Virtual Machine – HVM).

Είναι σημαντικό το γεγονός πως ενώ ο hypervisor έχει άμεση επαφή με το υλικό, η πρόσβαση των φιλοξενούμενων λειτουργικών συστημάτων στις φυσικές συσκευές, γίνεται μέσω του dom0.

4.3.2 Τεχνικές εικονικοποίησης στο XEN

-Paravirtualization

Όπως έχει περιγραφεί παραπάνω με τον όρο παραεικονικοποίηση. Το ειδικά τροποποιημένο λειτουργικό σύστημα ονομάζεται Xenolinux.

-Εικονική μηχανή υλικού (hardware virtual machine – HVM)

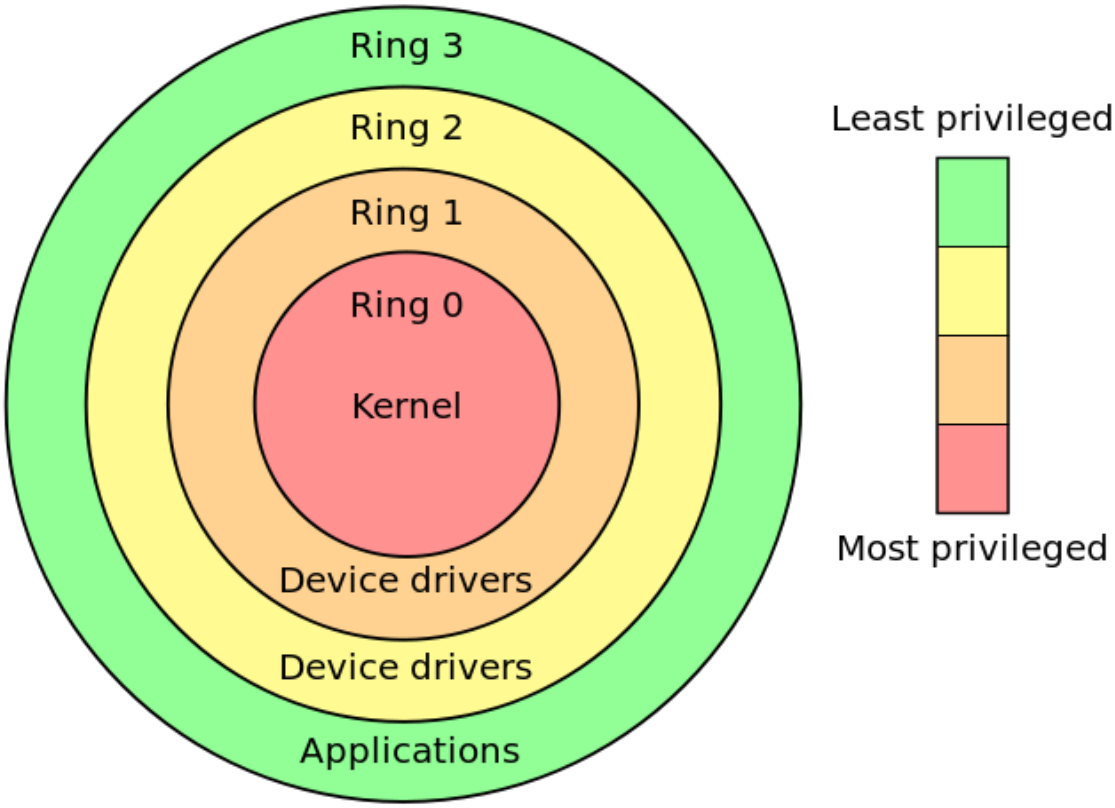
Το γνωστό από παραπάνω hardware assisted virtualization – εικονικοποίηση υποβοηθούμενη από το υλικό.

4.3.3 Η αρχιτεκτονική του XEN

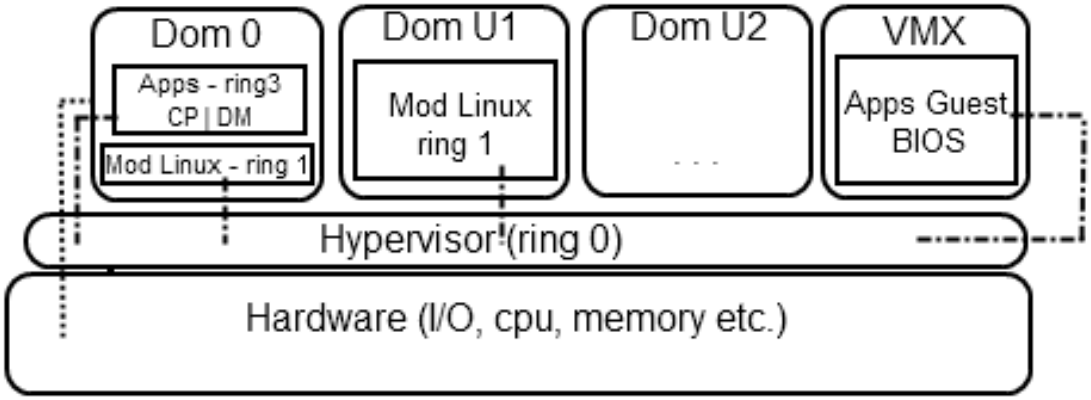
Η αρχιτεκτονική του XEN βασίζεται στην λογική των δακτυλίων προστασίας (Σχήμα 4.2). Βάσει της αρχιτεκτονικής των δακτυλίων, όσο βγαίνουμε από τον πυρήνα προς τα εξωτερικά στρώματα, τα δικαιώματα μειώνονται. Το πρώτο μέρος της αρχιτεκτονικής του XEN είναι ο hypervisor που και τρέχει απευθείας στο υλικό – δακτύλιος 0 (Σχήμα 4.3). Η εκτέλεση του πυρήνα του domain0 που καλείται και XEN0 γίνεται στον δακτύλιο-1. Όταν χρησιμοποιείται η τεχνική του paravirtualization, το φιλοξενούμενο λειτουργικό σύστημα το οποίο «τρέχει» στο domain U ονομάζεται XENOLINUX και περιλαμβάνει έναν τροποποιημένο πυρήνα Linux που τρέχει επίσης στον δακτύλιο-1. Όταν το XEN παρέχει τη δυνατότητα για full-virtualization (εικονική μηχανή υλικού), ο πυρήνας (kernel) του φιλοξενούμενου λειτουργικού συστήματος δεν χρειάζεται τροποποίηση. Σε αυτό βοηθάει η τεχνολογία των επεξεργαστών που υποστηρίζουν την εικονικοποίηση. Όπως οι Intel VT, και οι AMD - V.

Η επικοινωνία των φιλοξενούμενων λειτουργικών συστημάτων, με τις φυσικές συσκευές, γίνεται μέσω του domain0. Συγκεκριμένα, όταν υπάρξει κάποια κλήση από φιλοξενούμενο λειτουργικό σύστημα, για πρόσβαση στις φυσικές συσκευές, τότε

επικοινωνεί ο front end εικονικός οδηγός συσκευής του φιλοξενούμενου λειτουργικού με τον backend εικονικό οδηγό συσκευής του domain0 (XEN0). Το XEN0 έχει πρόσβαση στις φυσικές συσκευές μέσω των native device drivers που είναι οι πραγματικοί οδηγοί (drivers) του υλικού που διαχειρίζεται το κάθε σύστημα.



Σχήμα 4.2: *Οι δακτύλιοι προνομίων για την αρχιτεκτονική x86, που διατίθενται σε προστατευμένη κατάσταση λειτουργίας (protected mode)*



Σχήμα 4.3: *XEN αρχιτεκτονική βασισμένη στους δακτυλίους ασφάλειας, όπου εκτελείται το λειτουργικό σύστημα κάθε domain.*

Το XEN, και η τεχνική της παραεικονικοποίησης που αυτό εισάγαγε, αποτέλεσαν επανάσταση στο χώρο της εικονικοποίησης. Αυτό οφείλεται στο γεγονός ότι το XEN μπορούσε να επιτύχει επιδόσεις εικονικοποίησης παρόμοιες με bare metal λειτουργικά συστήματα. Οι επιδόσεις αυτές οφείλονται στο ότι ο hypervisor, που στην ουσία εναλλάσσει και εξυπηρετεί τα domains, “τρέχει” απευθείας πάνω στο υλικό. Επίσης, όπως περιγράψαμε, για την επικοινωνία των φιλοξενούμενων λειτουργικών συστημάτων με τις φυσικές συσκευές, επικοινωνεί ο front end εικονικός οδηγός συσκευής του φιλοξενούμενου λειτουργικού με τον backend εικονικό οδηγό συσκευής του domain0 (XEN0). Άρα συνολικά το κόστος εκτέλεσης ενός εικονικού λειτουργικού συστήματος στο XEN προσεγγίζει εκείνου ενός φυσικού μηχανήματος.

Μέρος Β’

Μέθοδοι απομόνωσης διεργασιών και εκτέλεσης τους στο υλικό εικονικοποίησης

ΚΕΦΑΛΑΙΟ 5ο

Διεργασίες

5.1 Γενικά

Στο Μέρος Β’ της εργασίας αυτής θα ασχοληθούμε με διεργασίες. Συγκεκριμένα, θα μελετήσουμε μεθόδους απομόνωσης διεργασιών και εκτέλεσης τους πάνω στις επεκτάσεις του υλικού για εικονικοποίηση, θα πειραματιστούμε με δική μας υλοποίηση και θα μετρήσουμε την απόδοση κάποιων τεχνικών. Θα παρουσιαστούν επίσης κάποια χρήσιμα εργαλεία. Στις πρώτες ενότητες του κεφαλαίου αυτού παρουσιάζονται γενικά θέματα που αφορούν την εκτέλεση των διεργασιών.

Διεργασία (process) είναι ένας όρος της πληροφορικής ο οποίος περιγράφει το στιγμιότυπο ενός προγράμματος που εκτελείται σε έναν υπολογιστή. Σε αντιδιαστολή με την έννοια του προγράμματος, το οποίο είναι ένα στατικό σύνολο εντολών, μια διεργασία συνιστά την εκτέλεση αυτών των εντολών. Επομένως ένα πρόγραμμα γενικώς συσχετίζεται με περισσότερες από μία διεργασίες, μία για κάθε φορά που εκτελείται. Μια διεργασία αποτελείται από το ίδιο το πρόγραμμα και από κάποιες τιμές που περιέχονται στη μνήμη και στους καταχωρητές του επεξεργαστή, δηλαδή την *κατάσταση* του συστήματος, κάθε στιγμή που το πρόγραμμα εκτελείται. Τα σύγχρονα λειτουργικά συστήματα επιτρέπουν την

ταυτόχρονη συνύπαρξη πολλαπλών διεργασιών στη μνήμη του υπολογιστή καθώς υποστηρίζουν *πολυδιεργασία*, μία μέθοδο υλοποίησης ταυτοχρονισμού με την οποία, είτε με κατάλληλη κατανομή του χρόνου του μοναδικού επεξεργαστή (*ψευδοπαράλληλη εκτέλεση*) είτε λόγω της ύπαρξης περισσοτέρων του ενός επεξεργαστών (*παράλληλη εκτέλεση*), είναι εφικτή η ταυτόχρονη εκτέλεση πολλαπλών διεργασιών.

Μία διεργασία υλοποιείται ως μία δομή δεδομένων του πυρήνα, η οποία κατασκευάζεται από το σύστημα όταν κάποιο πρόγραμμα φορτώνεται στη μνήμη για εκτέλεση και στην οποία συνήθως αποθηκεύονται τα εξής στοιχεία:

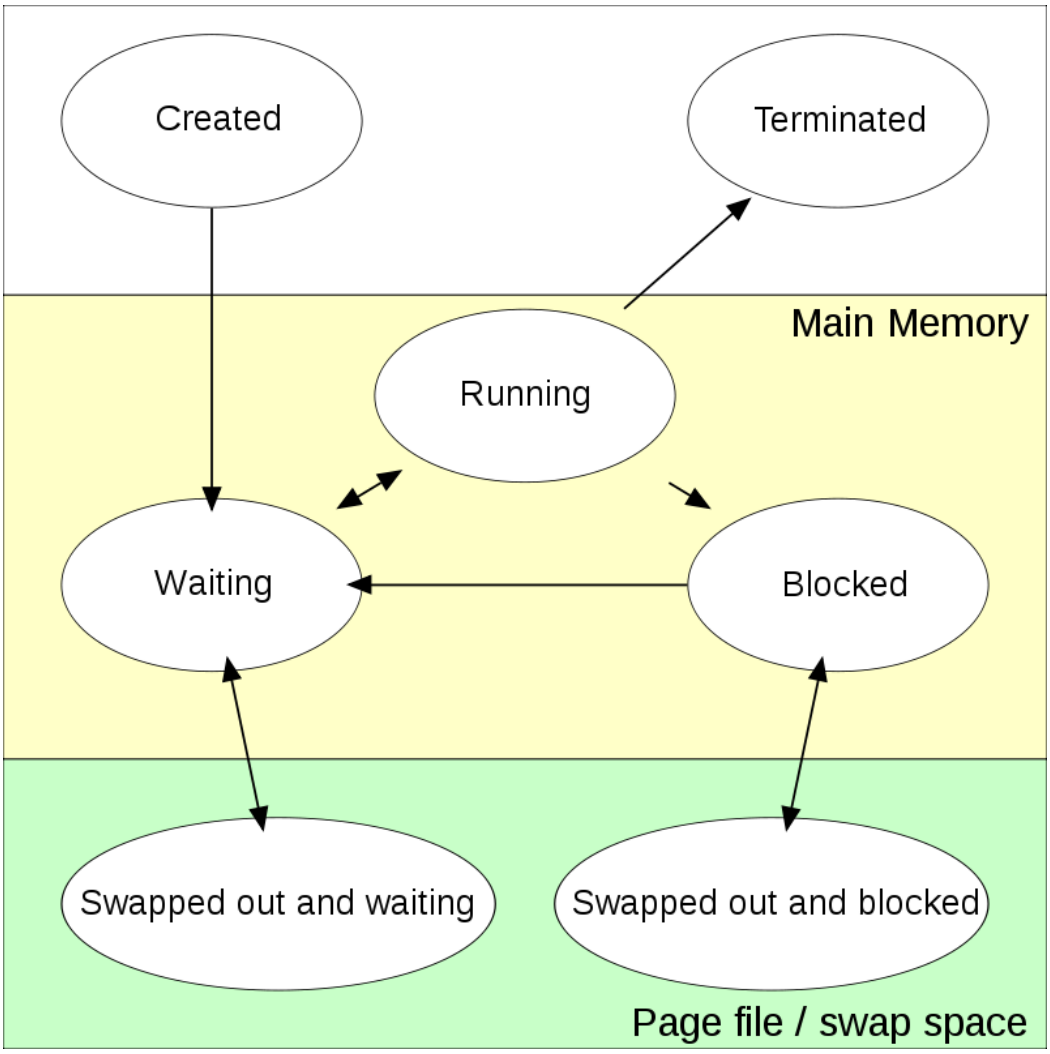
- Ένας ακέραιος αναγνωριστικός αριθμός
- Ο κώδικας του εκτελούμενου προγράμματος
- Τα καθολικά δεδομένα του προγράμματος, δηλαδή αυτά που είναι προσπελάσιμα από όλες τις διαδικασίες (procedures) του
- Τα μη αρχικοποιημένα (δεν τους δίνεται ρητά τιμή από τον προγραμματιστή κατά τη δήλωσή τους στον κώδικα) καθολικά δεδομένα του προγράμματος, τα οποία κατά την έναρξη εκτέλεσης της διεργασίας ο πυρήνας μηδενίζει ώστε να έχουν αρχική τιμή
- Η στοίβα του εκτελούμενου προγράμματος, όπου αποθηκεύονται οι τοπικές μεταβλητές κάθε διαδικασίας και η ιεραρχία των καλούμενων υποπρογραμμάτων
- Ο σωρός, από όπου γίνονται δυναμικές (κατά τον χρόνο εκτέλεσης) δεσμεύσεις μνήμης για δομές δεδομένων του προγράμματος
- Ο *πίνακας σελίδων*, ο οποίος είναι μία εσωτερική δομή δεδομένων κάθε διεργασίας απαραίτητη για τη λειτουργία του μηχανισμού της εικονικής μνήμης
- Πόροι που το πρόγραμμα χρησιμοποιεί κάθε στιγμή (π.χ. ανοιχτά αρχεία)
- Η τρέχουσα τιμή του μετρητή προγράμματος, ενός ειδικού καταχωρητή του επεξεργαστή ο οποίος δηλώνει ποια είναι η επόμενη προς εκτέλεση εντολή του προγράμματος
- Τα περιεχόμενα των άλλων καταχωρητών του επεξεργαστή κατά την εκτέλεση του προγράμματος

Η διεργασία δηλαδή δεν είναι μόνο ο δυαδικός κώδικας, αλλά ένα πλήρες σύνολο πληροφοριών για το πρόγραμμα και την κατάστασή του, κάθε στιγμή που αυτό εκτελείται.

5.2 Χρονοπρογραμματισμός διεργασιών

Στην πληροφορική **χρονοπρογραμματισμός ΚΜΕ** (αγγλ.: *scheduling*) ονομάζεται η χαρακτηριστική δυνατότητα των λειτουργικών συστημάτων, υλοποιούμενη συνήθως από έναν μηχανισμό του πυρήνα ονόματι *χρονοπρογραμματιστής*, με την οποία συντονίζεται η

συνύπαρξη πολλαπλών εκτελούμενων διεργασιών στη μνήμη του υπολογιστή. Με τον χρονοπρογραμματισμό επιτυγχάνεται επομένως η **πολυδιεργασία** (αγγλ.: *multitasking*), η οποία με τη σειρά της αποτελεί έναν τρόπο πρακτικής υλοποίησης ταυτοχρονισμού καθώς, είτε με κατάλληλη κατανομή του χρόνου του μοναδικού επεξεργαστή (*ψευδοπαράλληλη εκτέλεση*) είτε λόγω της ύπαρξης περισσοτέρων του ενός επεξεργαστών (*παράλληλη εκτέλεση*), είναι εφικτή η ταυτόχρονη εκτέλεση πολλαπλών διεργασιών στον ίδιο ηλεκτρονικό υπολογιστή.



Σχήμα 5.1: *Στην εικόνα απεικονίζονται με βέλη οι διάφορες μεταβάσεις κατάστασης μίας διεργασίας.*

Στα προεκτοπιστικά λειτουργικά συστήματα (preemptive) γίνεται αυτόματη εναλλαγή διεργασιών στον επεξεργαστή κάθε λίγες διακοπές του ρολογιού (στην αρχή κάθε «χρονικού κβάντου») ώστε να επιτευχθεί η ψευδοπαράλληλη εκτέλεση πολλαπλών διεργασιών. Στην πραγματικότητα οι διεργασίες εναλλάσσονται στον επεξεργαστή με εξαιρετικά μεγάλη συχνότητα (συνήθως το κβάντο διαρκεί κάποια millisecond). Η εναλλαγή αυτή ονομάζεται **θεματική εναλλαγή** (context switch) και, προκειμένου να είναι εφικτή, πρέπει όλες οι πληροφορίες που είναι αποθηκευμένες στην τοπική μνήμη του επεξεργαστή (στους καταχωρητές) για την εκτελούμενη διεργασία, να αποθηκευτούν σε έναν χώρο κάπου στη RAM κατά τη θεματική εναλλαγή. Έτσι, όταν έρθει ξανά η σειρά αυτής της διεργασίας να εκτελεστεί, θα μπορούν να φορτωθούν πάλι πίσω στους καταχωρητές και η εκτέλεση να συνεχίσει από εκεί που σταμάτησε. Τα δύο τελευταία στοιχεία της προηγούμενης λίστας παρέχουν αυτόν ακριβώς το χώρο. Το ποια διεργασία θα εκτελείται κάθε στιγμή καθορίζεται από έναν μηχανισμό του λειτουργικού συστήματος,τον χρονοπρογραμματιστή, η συμπεριφορά του οποίου συνήθως δεν μπορεί να προβλεφθεί ή να τροποποιηθεί από τον χρήστη.

Καθώς μία διεργασία εκτελείται, αλλάζει κατάσταση (state) (Σχήμα 5.1). Η κατάσταση ορίζεται εν μέρει από την τρέχουσα δραστηριότητά της διεργασίας και από τον χρονοπρογραμματιστή. Κάθε διεργασία μπορεί να βρίσκεται σε μία από τις ακόλουθες καταστάσεις:

- Νέα (New): Η διεργασία δημιουργείται.
- Εκτελούμενη (Running): Εκτελούνται εντολές.
- Εν αναμονή (Waiting): Η διεργασία αναμένει να συμβεί κάποιο γενονός (όπως η λήψη ενός σήματος).
- Έτοιμη (Ready): Η διεργασία περιμένει να ανατεθεί σε έναν επεξεργαστή.
- Τερματισμένη (Terminated): Η εκτέλεση της διεργασίας έχει ολοκληρωθεί.

5.3 Διαμοιραζόμενη μνήμη

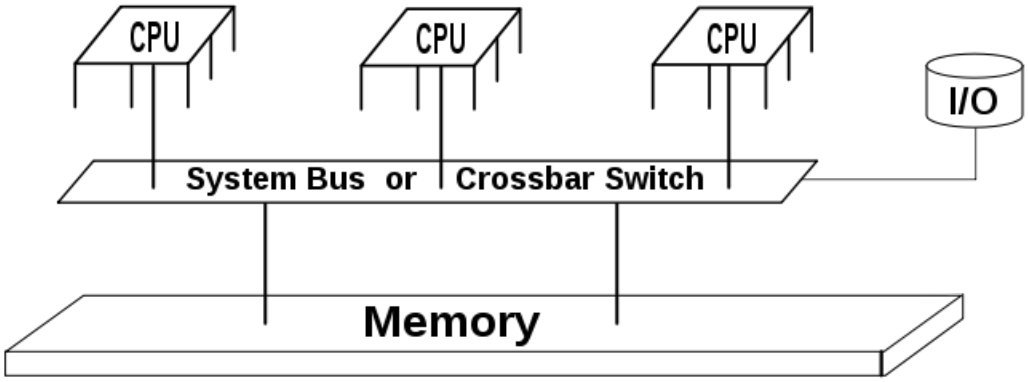
Στην επιστήμη των υπολογιστών, η κοινή μνήμη (shared memory) είναι μνήμη στην οποία μπορούν να έχουν πρόσβαση ταυτόχρονα πολλά διαφορετικά προγράμματα με πρόθεση να παρέχεται επικοινωνία μεταξύ τους ή να αποφεύγονται τα πλεονάζοντα αντίγραφα στη μνήμη (Σχήμα 5.2). Η κοινή μνήμη είναι ένα αποτελεσματικό μέσο μετάδοσης δεδομένων μεταξύ προγραμμάτων. Ανάλογα με το υπολογιστικό σύστημα, τα προγράμματα μπορούν να εκτελούνται σε έναν ενιαίο επεξεργαστή ή σε πολλούς ξεχωριστούς επεξεργαστές. Η χρήση της μνήμης για επικοινωνία μέσα σε ένα μόνο πρόγραμμα, π.χ. μεταξύ των πολλαπλών νημάτων του, αναφέρεται επίσης ως κοινή μνήμη.

Στο υλικό υπολογιστή, η κοινή μνήμη αναφέρεται σε ένα (τυπικά μεγάλο) μπλοκ μνήμης τυχαίας προσπέλασης (RAM), το οποίο, σε ένα σύστημα υπολογιστή πολλών επεξεργαστών, μπορεί να προσεγγιστεί από πολλές διαφορετικές κεντρικές μονάδες επεξεργασίας (CPU).

Στο λογισμικό των υπολογιστών, η κοινή μνήμη είναι είτε:

- Μια μέθοδος επικοινωνίας μεταξύ διαδικασιών (IPC), δηλαδή ένας τρόπος ανταλλαγής δεδομένων μεταξύ των προγραμμάτων που εκτελούνται ταυτόχρονα. Μια διεργασία θα δημιουργήσει μια περιοχή στη μνήμη RAM στην οποία μπορούν να έχουν πρόσβαση και οι άλλες διεργασίες.

- Μια μέθοδος συντήρησης του χώρου μνήμης, κατευθύνοντας τις προσβάσεις από αυτό που συνήθως θα αποτελούσε αντίγραφο ενός τμήματος δεδομένων, προς το ένα και μοναδικό τώρα στιγμιότυπο του τμήματος αυτού. Ο τρόπος που γίνεται αυτό είναι χρησιμοποιώντας εικονικές αντιστοιχίσεις μνήμης ή και με αποκλειστική-ειδική υποστήριξη του εν λόγω προγράμματος. Αυτό χρησιμοποιείται συχνότερα στις κοινές βιβλιοθήκες και για το XIP (eXecute In Place).



Σχήμα 5.2: *Μια απεικόνιση ενός συστήματος κοινής μνήμης τριών επεξεργαστών.*

5.4 Προστασία μνήμης και λειτουργία ασφαλείας

Η προστασία μνήμης είναι ένας τρόπος για τον έλεγχο των δικαιωμάτων της πρόσβασης στη μνήμη ενός υπολογιστή και αποτελεί μέρος των πιο σύγχρονων αρχιτεκτονικών συνόλου εντολών και λειτουργικών συστημάτων. Ο κύριος σκοπός της προστασίας μνήμης είναι να αποτρέψει μια διεργασία από την πρόσβαση της σε μνήμη που

δεν έχει παραχωρηθεί σε αυτήν. Αυτό αποτρέπει ένα σφάλμα ή κακόβουλο λογισμικό μέσα σε μια διεργασία από το να επηρεάσει άλλες διεργασίες ή το ίδιο το λειτουργικό σύστημα. Η προσπάθεια πρόσβασης μιας διεργασίας σε μνήμη που δεν της ανήκει έχει ως αποτέλεσμα ένα σφάλμα υλικού, που ονομάζεται σφάλμα κατάτμησης ή εξαίρεση παραβίασης της αποθήκευσης, και προκαλεί γενικά μη φυσιολογικό τερματισμό της παραβατικής διεργασίας. Για την ασφάλεια του υπολογιστή η προστασία της μνήμης περιλαμβάνει πρόσθετες τεχνικές όπως την τυχαιοποίηση της διάταξης του χώρου διευθύνσεων και την προστασία εκτελέσιμου χώρου.

Στην πληροφορική, ο προστατευμένος τρόπος λειτουργίας, που καλείται επίσης προστατευμένος τρόπος λειτουργίας εικονικής διεύθυνσης, είναι ένας τρόπος λειτουργίας των x86 κεντρικών μονάδων επεξεργασίας (CPUs). Επιτρέπει στο λογισμικό του συστήματος να χρησιμοποιεί λειτουργίες όπως είναι η εικονική μνήμη, η σελιδοποίηση και η ασφαλή λειτουργία πολλαπλών εργασιών. Οι λειτουργίες αυτές έχουν σχεδιαστεί για να αυξάνουν τον έλεγχο του λειτουργικού συστήματος πάνω στο λογισμικό εφαρμογών.

5.4.1 Μέθοδοι

-Κατάτμηση (Segmentation)

Η κατάτμηση αναφέρεται στη διαίρεση της μνήμης ενός υπολογιστή σε τμήματα. Μια αναφορά σε μια θέση μνήμης περιλαμβάνει μια τιμή που προσδιορίζει ένα τμήμα και μια σταθερά μετατόπισης εντός αυτού του τμήματος.

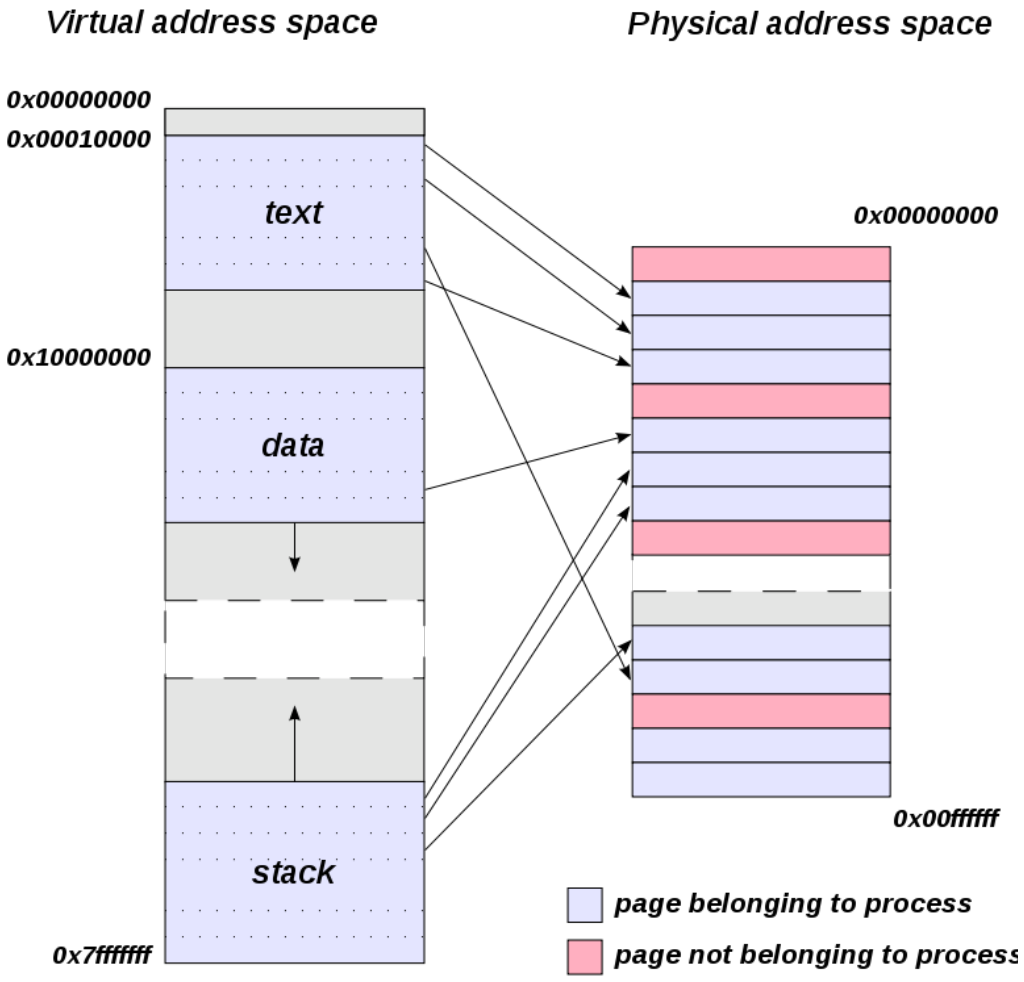
Η αρχιτεκτονική x86 διαθέτει πολλές λειτουργίες τμηματοποίησης, οι οποίες είναι χρήσιμες για τη λειτουργία προστατευμένης μνήμης σε αυτήν την αρχιτεκτονική. Στην αρχιτεκτονική αυτή, για την αναφορά τμημάτων στη μνήμη του υπολογιστή, χρησιμοποιούνται ο παγκόσμιος πίνακας περιγραφών (gdt) και οι τοπικοί πίνακες περιγραφών (ldt). Οι δείκτες προς τα τμήματα μνήμης στους x86 επεξεργαστές αποθηκεύονται στους καταχωρητές τμημάτων του επεξεργαστή. Αρχικά οι επεξεργαστές x86 είχαν 4 καταχωρητές τμημάτων, CS (τμήμα κωδικών), SS (τμήμα στοίβας), DS (τμήμα δεδομένων) και ES (έξτρα τμήμα). Αργότερα προστέθηκαν άλλοι δύο καταχωρητές τμημάτων - FS και GS.

-Σελιδοποιημένη Εικονική μνήμη (Paged virtual memory)

Κατά την σελιδοποίηση, ο χώρος διεύθυνσης μνήμης χωρίζεται σε μπλοκ ίσου μεγέθους που ονομάζονται σελίδες. Χρησιμοποιώντας υλικό εικονικής μνήμης, κάθε σελίδα μπορεί να βρίσκεται σε οποιαδήποτε θέση της φυσικής μνήμης του υπολογιστή ή να επισημαίνεται ως προστατευμένη. Η εικονική μνήμη καθιστά δυνατή την ύπαρξη ενός γραμμικού εικονικού χώρου διευθύνσεων μνήμης και τη χρήση του για πρόσβαση σε

κατακερματισμένα μπλοκ στο χώρο διεύθυνσης φυσικής μνήμης. Οι περισσότερες αρχιτεκτονικές υπολογιστών που υποστηρίζουν την σελιδοποίηση χρησιμοποιούν επίσης τις σελίδες ως βάση για την προστασία της μνήμης.

Ένας πίνακας σελίδων αντιστοιχεί την εικονική μνήμη στη φυσική μνήμη. Ο πίνακας σελίδων είναι συνήθως αόρατος στη διεργασία. Οι πίνακες σελίδων διευκολύνουν την εκχώρηση πρόσθετης μνήμης, καθώς κάθε νέα σελίδα μπορεί να διατεθεί από οπουδήποτε στη φυσική μνήμη.



Σχήμα 5.3: *Σχέση μεταξύ σελίδων που διευθυνσιοδοτούνται από εικονικές διευθύνσεις και σελίδων στη φυσική μνήμη, μέσα σε ένα απλό σχήμα χώρου διευθύνσεων. Η φυσική μνήμη μπορεί να περιέχει σελίδες που ανήκουν σε πολλές διεργασίες. Οι σελίδες μπορούν να κρατηθούν στο δίσκο αν χρησιμοποιούνται σπάνια ή εάν η φυσική μνήμη είναι πλήρης. Στο παραπάνω διάγραμμα, ορισμένες σελίδες δεν βρίσκονται στη φυσική μνήμη.*

Είναι αδύνατο για μια εφαρμογή να αποκτά πρόσβαση σε μια σελίδα που δεν έχει ρητά διατεθεί σε αυτήν, επειδή κάθε διεύθυνση μνήμης είτε δείχνει σε μια σελίδα που έχει εκχωρηθεί σε αυτήν την εφαρμογή είτε δημιουργεί μια διακοπή που ονομάζεται σφάλμα σελίδας (page fault). Οι μη καταχωρημένες σελίδες και οι σελίδες που διατίθενται σε οποιαδήποτε άλλη εφαρμογή δεν έχουν διευθύνσεις από την άποψη αυτής της εφαρμογής.

Ένα σφάλμα σελίδας μπορεί να μην σημαίνει απαραίτητα ένα σφάλμα. Οι βλάβες σελίδας δεν χρησιμοποιούνται μόνο για προστασία μνήμης. Το λειτουργικό σύστημα μπορεί να διαχειρίζεται τον πίνακα σελίδων με τέτοιο τρόπο ώστε μια αναφορά σε μια σελίδα που έχει προηγουμένως αντιγραφεί στο δίσκο να προκαλεί σφάλμα σελίδας. Το λειτουργικό σύστημα παρακολουθεί το σφάλμα της σελίδας, φορτώνει την απαιτούμενη σελίδα στη μνήμη και η εφαρμογή συνεχίζεται σαν να μην είχε συμβεί κάποιο σφάλμα. Αυτό το σχήμα, γνωστό ως εικονική μνήμη, επιτρέπει την αποθήκευση σε δίσκους και πίσω, δεδομένων εντός της μνήμης που δεν χρησιμοποιούνται προς το παρόν. Αυτό γίνεται με τρόπο καθαρό, δηλαδή χωρίς σφάλματα για τις εφαρμογές, και με τελικό σκοπό να αυξηθεί η συνολική χωρητικότητα μνήμης. Σε ορισμένα συστήματα, ο μηχανισμός σφάλματος σελίδας χρησιμοποιείται επίσης για την προστασία του εκτελέσιμου χώρου, όπως με την τεχνική W(writeable)-xor-X(executable).

5.5 Απομόνωση διεργασιών - sandboxing

Η απομόνωση διεργασιών αναφέρεται σε ένα σύνολο διαφορετικών τεχνολογιών υλικού και λογισμικού που έχουν σχεδιαστεί για να προστατεύουν κάθε διεργασία από τις άλλες διεργασίες στο λειτουργικό σύστημα. Αυτό γίνεται με την αποτροπή της διεργασίας Α από το να γράφει στο χώρο μνήμης της διεργασίας Β. Η απομόνωση διεργασίας μπορεί να υλοποιηθεί μέσω του εικονικού χώρου διευθύνσεων, όπου ο χώρος διευθύνσεων της διεργασίας Α είναι διαφορετικός από τον χώρο διεύθυνσης της διεργασίας Β – αποτρέποντας έτσι το Α από την εγγραφή στο Β. Η ασφάλεια είναι ευκολότερο να επιτευχθεί έτσι, αποκλείοντας την πρόσβαση στη μνήμη μεταξύ διεργασιών, σε αντίθεση με άλλες λιγότερο ασφαλείς αρχιτεκτονικές (όπως το DOS), στις οποίες οποιαδήποτε διεργασία μπορεί να γράψει σε οποιαδήποτε μνήμη οποιασδήποτε άλλης διεργασίας.

Στην ασφάλεια υπολογιστών, ένα sandbox είναι ένας μηχανισμός ασφαλείας για τον διαχωρισμό τρεχόντων προγραμμάτων. Συχνά χρησιμοποιείται για την εκτέλεση μη δοκιμασμένων ή μη αξιόπιστων προγραμμάτων ή κωδικών, που προέρχονται πιθανώς από μη επαληθευμένους ή μη αξιόπιστους τρίτους, προμηθευτές, χρήστες ή ιστοτόπους. Σκοπός του είναι να μην διακινδυνεύεται κάποια βλάβη στο μηχάνημα ή στο λειτουργικό σύστημα. Ένα sandbox τυπικά παρέχει ένα αυστηρά ελεγχόμενο σύνολο πόρων, για την εκτέλεση των “φιλοξενούμενων” προγραμμάτων, όπως “πρόχειρο”-scratch χώρο στο δίσκο και τη μνήμη.

Η πρόσβαση στο δίκτυο, η δυνατότητα επιθεώρησης του host συστήματος και η ανάγνωση από συσκευές εισόδου συνήθως δεν επιτρέπονται ή περιορίζονται σε μεγάλο βαθμό.

Με την έννοια της παροχής ενός εξαιρετικά ελεγχόμενου περιβάλλοντος, τα sandboxes μπορεί να θεωρηθούν ως ένα συγκεκριμένο παράδειγμα εικονικοποίησης. Το sandboxing χρησιμοποιείται συχνά για τη δοκιμή μη επαληθευμένων προγραμμάτων που ενδέχεται να περιέχουν ιό ή άλλο κακόβουλο κώδικα, ώστε να μην επιτρέπεται στο λογισμικό αυτό να βλάψει τη συσκευή του οικοδεσπότη.

5.5.1 Τεχνικές απομόνωσης διεργασιών

Από τις υπάρχουσες τεχνικές απομόνωσης διεργασιών μας ενδιαφέρουν οι παρακάτω:

-Απομόνωση σε επίπεδο host (sandboxing on native hosts)

Οι ερευνητές της ασφάλειας βασίζονται σε μεγάλο βαθμό στις τεχνολογίες sandboxing για την ανάλυση της συμπεριφοράς κακόβουλου λογισμικού. Δημιουργώντας ένα περιβάλλον που μιμείται ή αναπαράγει τους επιλεγμένους επιτραπέζιους υπολογιστές, οι ερευνητές μπορούν να αξιολογήσουν τον τρόπο με τον οποίο ο κακόβουλος κώδικας μολύνει και θέτει σε κίνδυνο έναν host στόχο. Πολλές υπηρεσίες ανάλυσης κακόβουλου λογισμικού βασίζονται στην τεχνολογία sandboxing.

-Πλήρες εικονικό περιβάλλον εκτέλεσης

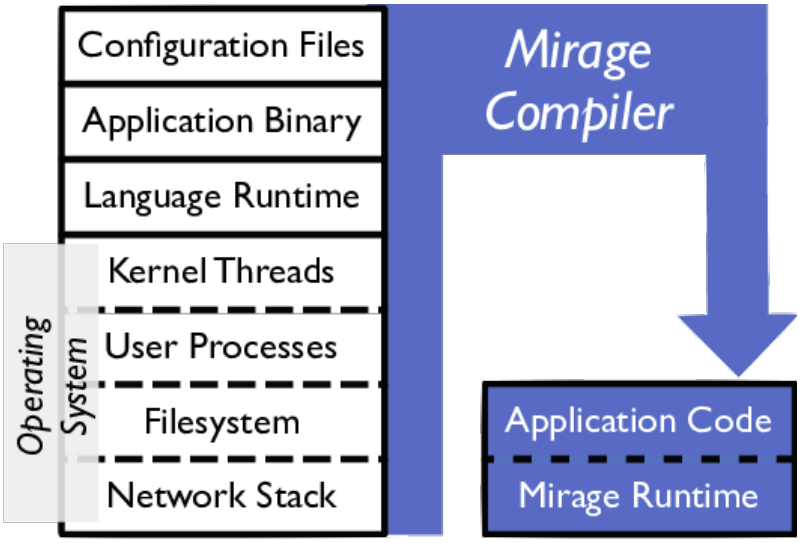
Οι εικονικές μηχανές προσομοιώνουν έναν πλήρη κεντρικό υπολογιστή, στον οποίο ένα συμβατικό λειτουργικό σύστημα μπορεί να εκκινήσει και να τρέξει όπως στο πραγματικό υλικό. Το λειτουργικό σύστημα του επισκέπτη τρέχει sandboxed με την έννοια ότι δεν λειτουργεί εγγενώς στον κεντρικό υπολογιστή και μπορεί να έχει πρόσβαση στους πόρους του οικοδεσπότη μόνο μέσω του εξομοιωτή ή του επόπτη.

-Εικονική εκτέλεση διεργασίας πάνω στις επεκτάσεις εικονικοποίησης του υλικού

Εδώ η διεργασία εκτελείται αποκλειστικά μόνη αυτή μέσα σε μία εικονική μηχανή που τρέχει πάνω στις επεκτάσεις εικονικοποίησης του υλικού. Δεν υπάρχει ολοκληρωμενο περιβάλλον εκτέλεσης δηλαδή δεν γίνεται προσομοίωση ολόκληρης εικονικής μηχανής και λειτουργικού συστήματος επισκέπτη. Η τεχνική αυτή χρησιμοποιείται για λόγους ασφάλειας αλλά και επίδοσης. Τέτοια υλοποίηση είναι το KVMSandbox του Andrew Ayer και το Dune του Πανεπιστημίου του Stanford που θα αναλυθεί περισσότερο παρακάτω.

-Unikernels, λειτουργικά συστήματα ειδικού σκοπού

Τα unikernels είναι εξειδικευμένες, ενιαίες εικόνες μηχανής, μονού χώρου διευθύνσεων, που έχουν κατασκευαστεί χρησιμοποιώντας λειτουργικά συστήματα βιβλιοθήκης. Ο προγραμματιστής επιλέγει, από μια αρθρωτή στοίβα, το ελάχιστο σύνολο βιβλιοθηκών, που αντιστοιχούν στις δομές του λετουργικού συστήματος, οι οποίες απαιτούνται για την εκτέλεση της επιθυμητής εφαρμογής. Οι βιβλιοθήκες αυτές μεταγλωττίζονται μαζί με τον κώδικα της εφαρμογής και με τον configuration code, ώστε να δημιουργήσουν “σφραγισμένες” εικόνες σταθερού σκοπού (unikernels), οι οποίες τρέχουν απευθείας σε έναν hypervisor, ή πάνω στο υλικό, χωρίς παρεμβαλλόμενο λειτουργικό σύστημα όπως το Linux ή τα Windows (Σχήμα 5.4).



Σχήμα 5.4: *Παράδειγμα αρχιτεκτονικής unikernel (MirageOS), σε σύγκριση με μια παραδοσιακή στοίβα λειτουργικού συστήματος.*

Καθώς οι hypervisors στηρίζουν το μεγαλύτερο μέρος της δημόσιας υποδομής του υπολογιστικού νέφους, οι unikernels κάνουν τις αντίστοιχες παρεχόμενες υπηρεσίες να λειτουργούν πιο φτηνά, πιο ασφαλείς και με λεπτότερο έλεγχο σε σύγκριση με μια πλήρη στοίβα λογισμικού. Τα unikernels παρέχουν πολλά οφέλη σε σύγκριση με ένα παραδοσιακό λειτουργικό σύστημα, συμπεριλαμβανομένης της βελτιωμένης ασφάλειας, της μικρότερης επεμβατικότητας στο σύστημα, της μεγαλύτερης βελτιστοποίησης και των ταχύτερων χρόνων εκκίνησης.

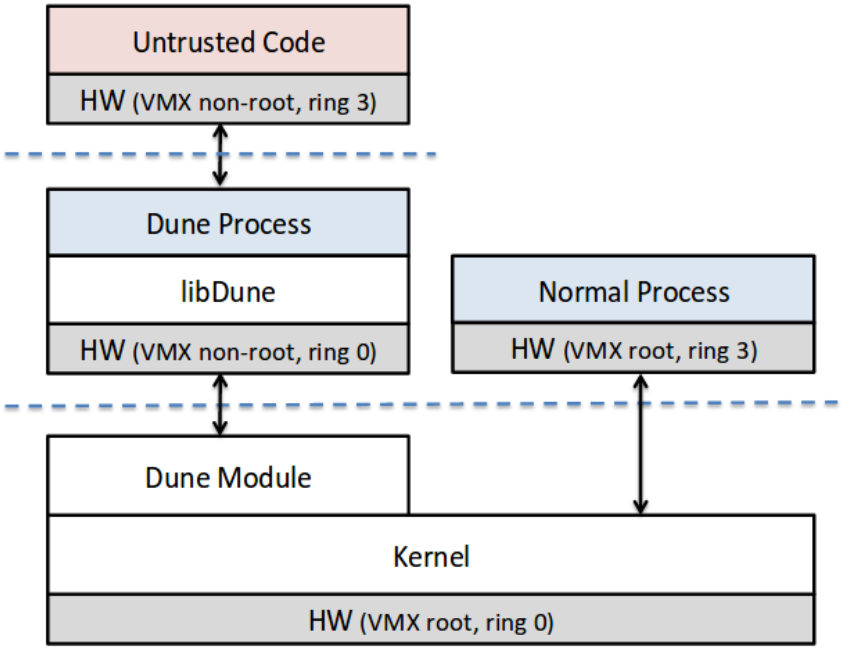
5.6 Εργαλεία

5.6.1 kvmtool: Το εγγενές εργαλείο KVM του Linux

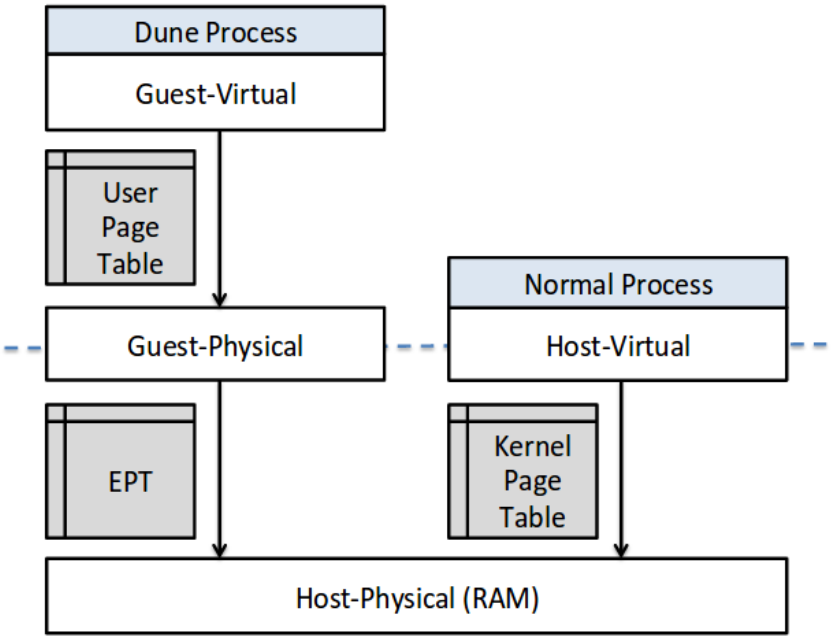
Ο στόχος αυτού του εργαλείου είναι να παρέχει μια καθαρή, από το μηδέν, και ελαφριά υλοποίηση, (όχι μεγάλη και σύνθετη όπως το QEMU), ενός host εργαλείου για το KVM, το οποίο να μπορεί να εκκινεί εικόνες Linux επισκεπτών, χωρίς εξαρτήσεις από το BIOS, και με ελάχιστη ποσότητα εξομοίωσης συσκευών παλαιού τύπου. Είναι ένα ωραίο εργαλείο εκμάθησης για την εικονικοποίηση καθώς είναι μόνο 5 χιλιάδες γραμμές καθαρού κώδικα C. Εκκινεί μια εικόνα Linux του επισκέπτη και παρέχει έξοδο στο terminal του οικοδεσπότη μέσω μιας σειριακής κονσόλας. Μπορούμε να το χρησιμοποιησουμε για να εκκινήσουμε μια εικόνα Linux guest, στο terminal, ή πάνω από ssh, και να πραγματοποιήσουμε σύνδεση στον επισκέπτη χωρίς πολλές απαιτούμενες εργασίες εγκατάστασης και από τις δύο πλευρές. Το εργαλείο έχει γραφτεί από τους Pekka Enberg, Cyrill Gorcunov, Asias He, Sasha Levin και Prasad Joshi.

5.6.2 dune: Ασφαλής πρόσβαση χρήστη σε προνομιακές λειτουργίες CPU

Το Dune είναι ένα σύστημα που παρέχει στις εφαρμογές άμεση αλλά ασφαλή πρόσβαση σε λειτουργίες υλικού, όπως είναι η προστασία δακτυλίων, οι πίνακες σελίδων και οι TLBs με ετικέτες (Σχήμα 5.7). Παράλληλα διατηρεί την υπάρχουσα διεπαφή του λειτουργικού συστήματος για τις διεργασίες. Το Dune χρησιμοποιεί το υλικό εικονικοποίησης στους σύγχρονους επεξεργαστές για να παρέχει μια αφαίρεση διεργασίας και όχι μηχανής (Σχήμα 5.5, 5.6). Αποτελείται από ένα μικρό δομοστοιχείο λογισμικού (module) πυρήνα που προετοιμάζει το υλικό εικονικοποίησης και μεσολαβεί στις αλληλεπιδράσεις με τον πυρήνα. Επίσης αποτελείται από μια βιβλιοθήκη σε επίπεδο χρήστη που βοηθάει τις εφαρμογές να διαχειρίζονται τις προνομιακές λειτουργίες του υλικού. Το Dune υλοποιήθηκε αρχικά για 64-bit x86 Linux και χρησιμοποιήθηκε για την υλοποίηση τριών εφαρμογών σε επίπεδο χρήστη, οι οποίες μπορούν να επωφεληθούν από την πρόσβαση σε προνομιακό υλικό: ένα sandbox για μη αξιόπιστο κώδικα, μια υποδομή διαχωρισμού προνομίων και ένα συλλέκτη απορριμμάτων. Η χρήση του Dune απλοποιεί σημαντικά την υλοποίηση αυτών των εφαρμογών και παρέχει σημαντικά πλεονεκτήματα απόδοσης. Το Dune υλοποιήθηκε στο Πανεπιστήμιο του Stanford από τους Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazi`eres και Christos Kozyrakis. Υπάρχει και port για αρχιτεκτονική ARM από τον Yu Chen του Πανεπιστημίου Tsinghua του Πεκίνου της Κίνας .



Σχήμα 5.5: *Η αρχιτεκτονική του συστήματος Dune*



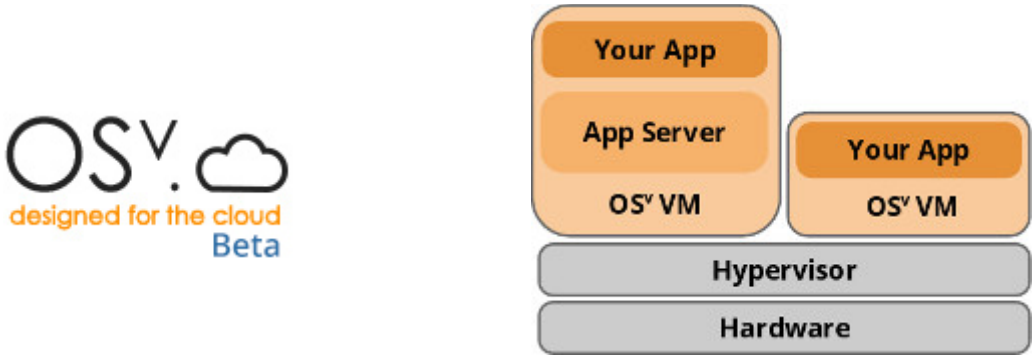
Σχήμα 5.6: *Η εικονική μνήμη στο Dune*

Mechanism	Privileged Instructions
Exceptions	LIDT, LTR, IRET, STI, CLI
Virtual Memory	MOV CRn, INVLPG, INVPCID
Privilege Modes	SYSRET, SYSEXIT, IRET
Segmentation	LGDT, LLDT

Σχήμα 5.7: *Χαρακτηριστικά υλικού που εκτίθενται από το Dune και οι αντίστοιχες προνομιούχες x86 οδηγίες.*

5.6.3 OSv: Ο unikernel για το cloud

Το OSv (Σχήμα 5.8) είναι ένα νέο λειτουργικό σύστημα που σχεδιάστηκε ειδικά για cloud εικονικές μηχανές από την Cloudius Systems. Ικανό να εκκινήσει σε λιγότερο από ένα δευτερόλεπτο, το OSv σχεδιάστηκε από το μηδέν για να εκτελέσει μια μοναδική εφαρμογή επάνω σε οποιονδήποτε hypervisor. Αποτέλεσμα είναι η καλύτερη απόδοση, η ταχύτητα και η εύκολη διαχείριση. Η υποστήριξη είναι διαθέσιμη για τις στοίβες εφαρμογών C, JVM, Ruby και Node.js.



(α) (β)

Σχήμα 5.8: *(α) Το λογότυπο του OSv unikernel. (β) Το περιβάλλον χρόνου εκτέλεσης μιας γλώσσας, το λειτουργικό σύστημα και ο hypervisor παρέχουν προστασία και αφαίρεση. Το OSv ελαχιστοποιεί (αφαιρεί) τον πλεονασμό σε αυτά τα επίπεδα, απλοποιώντας το λειτουργικό σύστημα.*

ΚΕΦΑΛΑΙΟ 6ο

Πειραματική υλοποίηση

6.1 Εκτελέσιμο binary-kvmtool

Κατά την προσέγγιση αυτή επιχειρήσαμε στο εργαστήριο C.A.R.V του I.T.E. να τρέξουμε ένα επιθυμητό προμεταγλωτισμένο εκτελέσιμο κατευθείαν πάνω στο kvmtool, στη θέση που θα φορτωνόταν η εικόνα πυρήνα Linux του επισκέπτη. Έτσι θεωρήσαμε ότι ο κώδικας της διεργασίας που θέλουμε θα τρέχει απευθείας πάνω στις επεκτάσεις εικονικοποίησης του επεξεργαστή, καθώς θα είναι ο μόνος κώδικας που θα τρέχει μέσα στην εικονική μηχανή του kvmtool, στη θέση του λειτουργικού συστήματος. Δεν χρειάστηκε να γίνουν αλλάγές στον κώδικα του kvmtool για να το πετύχουμε. Το πείραμα δούλεψε, οι τιμές των καταχωρητών άλλαζαν σωστά, και άρα θα μπορούσαμε να μιλήσουμε για μια μέθοδο sandboxing διεργασιών. Όμως το περιβάλλον της εικονικής μηχανής, για παράδειγμα οι segment registers, δεν ήταν αρχικοποιημένο, επειδή δεν είχε τρέξει τον αρχικό του κώδικα το λειτουργικό σύστημα, που θα έτρεχε κατά το boot του, μιας και δεν κάναμε ποτέ boot λειτουργικό σύστημα. Στη θέση του kernel βάλαμε το εκτελέσιμο που θέλουμε να τρέξει πάνω στις επεκτάσεις εικονικοποίησης του υλικού. Έτσι δεν μπορέσαμε ποτέ να λάβουμε ολοκληρωμένα αποτελέσματα ή να χειριστούμε τις εξόδους της εικονικής μηχανής στο χώρο του host χρήστη, για να διαχειριστούμε κάποιο system call, παρόλο που το process μέσα στην εικονική μηχανή συνέχιζε να τρέχει. Για να λύσουμε το παραπάνω πρόβλημα θα έπρεπε να αρχικοποιήσουμε από μόνοι μας το περιβάλλον εκτέλεσης της εικονικής μηχανής, κάτι που μας οδήγησε αργότερα να σκεφτούμε την λύση των unikernels που στην ουσία θα έκαναν αυτό που ακριβώς χρειαζόμασταν. Ο κώδικας ενός παραδείγματος και τα αποτελέσματα του βρίσκονται παρακάτω. Εκεί φαίνεται ότι οι πράξεις μεταξύ των καταχωρητών γίνονται σωστά, το αποτέλεσμα τυπώνεται στην οθόνη,

αλλά η εικονική μηχανή δεν αντιδράει στο εσωτερικό interrupt (κλήση συστήματος) ενώ σε άλλες περιπτώσεις δεν κάνει halt.

```
#include <stdio.h>
#include <stdlib.h>
void main ()
{
      asm volatile(     "mov $0x3f8, %rdx\n"
                  "mov $2, %rbx\n"
                  "mov $2, %rcx\n"
                  "add %rcx, %rbx\n"
                  "add $'0', %rbx\n"
                  "mov %rbx, %rax\n"
                  "out %al, (%dx)\n" //out “4”
                  "mov $'\n', %rax\n"
                  "out %al, (%dx)\n"
                  "int $0x80\n" //cd 80h
                  "hlt\n"
                  );
      exit(1);
}
```

Πάνω: Απλό παράδειγμα σε x86 assembly – Κάτω: Αποτελέσματα

```
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/d/linux-
kvm/tools/kvm$ ./lkvm run --disk disks/linux-0.2.img     --kernel
examples/ex.o
main
handle_kvm_command
handle_command
kvm_get_command
kvm_cmd_run
kvm_cmd_run_init
  # lkvm run -k examples/ex.o -m 448 -c 4 --name guest-10845
KVM_API_VERSION:12
kvm__load_kernel
load_bzimage
  Warning: examples/ex.o is not a bzImage. Trying to load it as a flat
binary...
weak load_elf_binary
load_flat_binary
0
kvm__set_thread_name
kvm__set_thread_name
kvm__set_thread_name
kvm__set_thread_name
kvm__set_thread_name
kvm__set_thread_name
kvm_cmd_run_work
kvm__set_thread_name
kvm_cpu_thread
kvm__set_thread_name
kvm_cpu_thread
kvm__set_thread_name
kvm_cpu_thread
```

```
kvm__set_thread_name
kvm_cpu_thread
kvm__set_thread_name
4
 Registers:
 ----------
 rip: 0000000000000064   rsp: 0000000000007fff flags: 0000000000000002
 rax: 0000000000000034   rbx: 0000000000000034   rcx: 0000000000000002
 rdx: 00000000000003f8   rsi: 0000000000000001   rdi: 0000000000000000
 rbp: 0000000000007fff    r8: 0000000000000000    r9: 0000000000000000
 r10: 0000000000000000   r11: 0000000000000000   r12: 0000000000000000
 r13: 0000000000000000   r14: 0000000000000000   r15: 0000000000000000
 cr0: 0000000060000010   cr2: 0000000000000000   cr3: 0000000000000000
 cr4: 0000000000000000   cr8: 0000000000000000

 Segment registers:
 ------------------
 register  selector  base              limit     type  p dpl db s l g avl
 cs        1000      0000000000010000  0000ffff  0b    1 0   0  1 0 0 0
 ss        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
 ds        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
 es        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
 fs        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
 gs        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
 tr        0000      0000000000000000  0000ffff  0b    1 0   0  0 0 0 0
 ldt       0000      0000000000000000  0000ffff  02    1 0   0  0 0 0 0
 gdt                 0000000000000000  0000ffff
 idt                 0000000000000000  0000ffff

 APIC:
 -----
 efer: 0000000000000000  apic base: 00000000fee00900  nmi: enabled

 Interrupt bitmap:
 -----------------
 0000000000000000 0000000000000000 0000000000000000 0000000000000000

 Page Tables:
 ------
 Not in protected mode

 Code:
 -----
 rip: [<0000000000000064>] <unknown>

 00 40 00 0c 00 09 00 55 48 89 e5 48 c7 c2 f8 03 00 00 48 c7 c3 02 00 00
00 48 c7 c1 02 00 00 00 48 01 cb 48 83 c3 30 48 89 d8 ee <48> c7 c0 0a 00
00 00 ee cd 80 f4 bf 01 00 00 00 e8 00 00 00 00

 Stack:
 ------
 rsp: [<0000000000007fff>]
 0x00007fff: 00 00 00 00  00 00 00 00
 0x00008007: 00 00 00 00  00 00 00 00
 0x0000800f: 00 00 00 00  00 00 00 00
```

```
0x00008017: 00 00 00 00  00 00 00 00


Registers:
----------
rip: 000000000000006c   rsp: 0000000000007fff flags: 0000000000000006
rax: 000000000000000a   rbx: 0000000000000034   rcx: 0000000000000002
rdx: 00000000000003f8   rsi: 0000000000000001   rdi: 0000000000000000
rbp: 0000000000007fff    r8: 0000000000000000    r9: 0000000000000000
r10: 0000000000000000   r11: 0000000000000000   r12: 0000000000000000
r13: 0000000000000000   r14: 0000000000000000   r15: 0000000000000000
cr0: 0000000060000010   cr2: 0000000000000000   cr3: 0000000000000000
cr4: 0000000000000000   cr8: 0000000000000000

Segment registers:
------------------
register  selector  base              limit     type  p dpl db s l g avl
cs        1000      0000000000010000  0000ffff  0b    1 0   0  1 0 0 0
ss        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
ds        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
es        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
fs        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
gs        1000      0000000000010000  0000ffff  03    1 0   0  1 0 0 0
tr        0000      0000000000000000  0000ffff  0b    1 0   0  0 0 0 0
ldt       0000      0000000000000000  0000ffff  02    1 0   0  0 0 0 0
gdt                 0000000000000000  0000ffff
idt                 0000000000000000  0000ffff

APIC:
-----
efer: 0000000000000000  apic base: 00000000fee00900  nmi: enabled

Interrupt bitmap:
-----------------
0000000000000000 0000000000000000 0000000000000000 0000000000000000

Page Tables:
------
Not in protected mode

Code:
-----
rip: [<000000000000006c>] <unknown>

 48 89 e5 48 c7 c2 f8 03 00 00 48 c7 c3 02 00 00 00 48 c7 c1 02 00 00 00
48 01 cb 48 83 c3 30 48 89 d8 ee 48 c7 c0 0a 00 00 00 ee <cd> 80 f4 bf 01
00 00 00 e8 00 00 00 00 00 47 43 43 3a 20 28 55

Stack:
------
rsp: [<0000000000007fff>]
0x00007fff: 00 00 00 00  00 00 00 00
0x00008007: 00 00 00 00  00 00 00 00
0x0000800f: 00 00 00 00  00 00 00 00
0x00008017: 00 00 00 00  00 00 00 00
```

6.2 libdune-KVM

Κατά την προσέγγιση αυτή προσπαθήσαμε να συνδέσουμε την βιβλιοθήκη χρήστη του Dune με το KVM module του πυρήνα. Έτσι θα μπορούσαμε να χρησιμοποιήσουμε την εφαρμογή sandbox που έχει γραφεί χρησιμοποιώντας την βιβλιοθήκη libdune, και παρέχεται σαν παράδειγμα μέσα στο Dune project, ώστε να φορτώσουμε ένα εκτελέσιμο, και να το τρέξουμε μέσα από KVM πάνω στις επεκτάσεις εικονικοποίησης του υλικού. Αυτό γίνεται μέσω μιας ελάχιστης εικονικής μηχανής που κατασκευάζουμε εμείς με βάση το KVM, αντικαθιστώντας αυτή που κατασκευάζει το Dune module. Τελικός σκοπός είναι να πετύχουμε απομόνωση, ασφάλεια και καλύτερη απόδοση λόγω έλλειψης ολοκληρωμένης εικονικής μηχανής. Η προσέγγιση αυτή ήταν πολύ ελπιδοφόρος καθώς θα εξασφάλιζε και φορητότητα μέσω του KVM που είναι υλοποιημένο για αρκετές αρχιτεκτονικές εν αντιθέση με το Dune που είναι υλοποιημένο κυρίως για Intel x86.

Για το σκοπό της παραπάνω προσέγγισης έπρεπε να ρυθμιστούν όλες οι παράμετροι της εικονικής μηχανής KVM ακριβώς όπως τις ρυθμίζουν το Dune module και η libdune στη δική τους εικονική μηχανή που κατασκευάζουν. Δηλαδή έπρεπε να αντιγραφούν οι τιμές των καταχωρητών τις εικονικής μηχανής Dune, σε αυτους της εικονικής μηχανής που στήναμε εμείς με το KVM, και επίσης να είναι η φυσική μνήμη της εικονικής μηχανής Dune ταυτόσημη με αυτής του KVM. Για το σκοπό αυτό αντιγράψαμε όλες τις περιοχές μνήμης που δεσμέυει η libdune ως φυσικη μνήμη του επισκέπτη, στις ίδιες περιοχές μνήμης της εικονικής μηχανής του KVM. Το ίδιο έγινε και για τους καταχωρητές. Αρχικοποιήθηκαν στην εικονική μηχανή KVM όπως αρχικοποιούνταν από το Dune module και τη libdune. Οι παραπάνω ενέργειες έγιναν με τη βοήθεια συγκεκριμένων ioctls του KVM module, τα οποία έχουν περιγραφεί παραπάνω.

Παρά το γεγονός ότι τηρήθηκε κάθε λεπτομέρεια στις διευθύνσεις, στο περιεχόμενο των θέσεων μνήμης που αντιγράφησαν (στην ουσία γίνονται mmap() ώστε να ακολουθούν και τις αλλαγές η μία της άλλης) και στο περιεχόμενο των καταχωρητών, το εκτελέσιμο ELF που φορτώθηκε σε συγκεκριμένη θέση μνήμης στην εικονική μηχανή KVM δεν εκτελέστηκε. Το σφάλμα που προκύπτει από το KVM είναι ότι δεν αναγνωρίζεται ο κώδικας της πρώτης θέσης μνήμης του εκτελέσιμου. Εκτιμούμε ότι αυτό το σφάλμα ίσως οφείλεται στα εξής : α) ελλειπής ρύθμιση της μνήμης του guest από το KVM. Ίσως δεν είναι εκτελέσιμη αν και όλες οι περιοχές που έγιναν mmaped() προς αυτήν ήταν, το block της ήταν ενιαίο και οι σχετικόι καταχωρητές gdt, idt και tss σεταρισμένοι, β) μη ρύθμιση των msr καταχωρητών στον guest καθώς δεν θεωρήθηκαν υψίστου προτεραιότητας και σημασίας εξ’αρχής και επίσης υπήρχε ελλιπής πληροφόρηση για το πως μπορούν αυτοί να ρυθμιστούν στο KVM, γ) κάποια εσωτερική ρύθμιση του KVM.

Να σημειώσουμε ότι οι κλήσεις συστήματος ήταν δυνατό να εκτελεστούν από τη διεργασία του εκτελέσιμου, χωρίς να επέμβουμε με περαιτέρω κώδικα για να πάρουμε τον έλεγχο στον host, καθώς είχαν ήδη αντιγραφεί οι δομές στη μνήμη της εικονικής μηχανής

KVM, οι οποίες εξυπηρετούσαν (στην ουσία μέσω software interrupt και hardware) τις κλήσεις συστήματος. Τις ίδιες δομές αυτές τις κατασκεύαζε η libdune, για την εικονική μηχανή του Dune module.

Παρακάτω φαίνεται ο κώδικας των συναρτήσεων που υλοποιήθηκαν, και ενσωματώθηκαν σε συγκεκριμένα σημεία στη libdune για τη προσέγγιση αυτή. Βασίστηκε στα κυριότερα ioctls του KVM που προαναφέρθηκαν στο κεφάλαιο 4:

Συναρτήσεις libdune-KVM:

```
//Astr
long aligned_page_size(long len)
{//printf("%s\n",__func__);
      //Astr
      int fixed_page_num = (len / PGSIZE); //memory alignment
      int fixed_page_rem = (len % PGSIZE); //remainder of the division:
(memory size)/PGSIZE
      if (fixed_page_rem>0){
            fixed_page_num++;
      }

      return fixed_page_num * PGSIZE;
}

void dump_vcpu_registers()
{//printf("%s\n",__func__);
            int ret;
            //Astr regs
            ret = ioctl(vcpufd, KVM_GET_REGS, &regs);
            if (ret == -1)
                  err(1, "KVM_GET_REGS");
            printf("KVM vcpu REGS:\n");
            printf("rax: %lx rbx: %lx rcx: %lx rdx: %lx\n",  regs.rax,
regs.rbx, regs.rcx, regs.rdx);
            printf("rsi: %lx rdi: %lx rsp: %lx rbp: %lx\n",  regs.rsi,
regs.rdi, regs.rsp, regs.rbp);
            printf("r8: %lx r9: %lx r10: %lx r11: %lx\n",  regs.r8,
regs.r9, regs.r10, regs.r11);
            printf("rip: %lx rflags: %lx\n",  regs.rip, regs.rflags);

            //Astr sregs
            ret = ioctl(vcpufd, KVM_GET_SREGS, &sregs);
            if (ret == -1)
                  err(1, "KVM_GET_SREGS");
            printf("KVM vcpu SREGS:\n");
            printf("CS:\n");
printf("cs.base: %lx cs.limit: %lx cs.selector: %lx cs.type:
%lx\n",sregs.cs.base, sregs.cs.limit, sregs.cs.selector, sregs.cs.type);
printf("cs.present: %lx cs.dpl: %lx cs.db: %lx cs.s: %lx cs.l: %lx\n",
sregs.cs.present, sregs.cs.dpl, sregs.cs.db, sregs.cs.s,sregs.cs.l);
printf("cs.g: %lx cs.avl: %lx cs.unusable: %lx cs.padding: %lx\n",
sregs.cs.g, sregs.cs.avl, sregs.cs.unusable, sregs.cs.padding);
printf("DS:\n");
printf("ds.base: %lx ds.limit: %lx ds.selector: %lx ds.type:
```

```
%lx\n",sregs.ds.base, sregs.ds.limit, sregs.ds.selector, sregs.ds.type);
printf("ds.present: %lx ds.dpl: %lx ds.db: %lx ds.s: %lx ds.l: %lx\n",
sregs.ds.present, sregs.ds.dpl, sregs.ds.db, sregs.ds.s,sregs.ds.l);
printf("ds.g: %lx ds.avl: %lx ds.unusable: %lx ds.padding: %lx\n",
sregs.ds.g, sregs.ds.avl, sregs.ds.unusable, sregs.ds.padding);
printf("CR:\n");
printf("cr0: %lx cr2: %lx cr3: %lx cr4: %lx\n",sregs.cr0, sregs.cr2,
sregs.cr3, sregs.cr4);
printf("GDT:\n");
printf("gdt.base: %lx gdt.limit: %lx gdt.padding: %lx\n",sregs.gdt.base,
sregs.gdt.limit, sregs.gdt.padding);
printf("IDT:\n");
printf("idt.base: %lx idt.limit: %lx idt.padding: %lx\n",sregs.idt.base,
sregs.idt.limit, sregs.idt.padding);
}

void set_vm_cr3(uint64_t cr3){
      int ret;
      sregs.cr3=cr3;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
            err(1, "KVM_SET_SREGS");
}

void set_vm_cr0(uint64_t cr0){
      int ret;
      sregs.cr0=cr0;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
            err(1, "KVM_SET_SREGS");
}

void set_vm_ip(uint64_t ip){
      int ret;
      regs.rip=ip;
      ret = ioctl(vcpufd, KVM_SET_REGS, &regs);
      if (ret == -1)
            err(1, "KVM_SET_REGS");
}

void set_vm_sp(uint64_t sp){
      int ret;
      regs.rsp=sp;
      ret = ioctl(vcpufd, KVM_SET_REGS, &regs);
      if (ret == -1)
                  err(1, "KVM_SET_REGS");
}
void set_gdt(uint16_t limit, uint64_t base){
      int ret;
      sregs.gdt.limit=limit;
      sregs.gdt.base=base;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
```

```
void set_idt(uint16_t limit, uint64_t base){
      int ret;
      sregs.idt.limit=limit;
      sregs.idt.base=base;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}

void set_ds(uint64_t ds){
      int ret;
      sregs.ds.selector=ds;
      sregs.ds.g = 1;
      sregs.ds.l = 1;
      //sregs.ds.attribute=;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
void set_es(uint64_t es){
      int ret;
      sregs.es.selector=es;
      sregs.es.g = 1;
      sregs.es.l = 1;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
void set_ss(uint64_t ss){
      int ret;
      sregs.ss.selector=ss;
      sregs.ss.g = 1;
      sregs.ss.l = 1;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
void set_cs(uint64_t cs){
      int ret;
      sregs.cs.selector=cs;
      sregs.cs.g = 1;
      sregs.cs.l = 1;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
void set_fs_base(uint64_t fs_base){
      int ret;
      sregs.fs.base=fs_base;
      sregs.fs.g = 1;
      sregs.fs.l = 1;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
```

```
void set_gs_base(uint64_t gs_base){
      int ret;
      sregs.gs.base=gs_base;
      sregs.gs.g = 1;
      sregs.gs.l = 1;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
void set_tss(uint64_t tss){
      int ret;
      sregs.tr.selector=tss;
      sregs.tr.limit=0xff;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
                  err(1, "KVM_SET_SREGS");
}
void set_dr7(uint64_t dr7){
      int ret;
      debugregs.dr7=dr7;
      ret = ioctl(vcpufd, KVM_SET_DEBUGREGS, &debugregs);
      if (ret == -1)
            err(1, "KVM_SET_DEBUGREGS");
}

int init_kvm()
{//printf("%s\n",__func__);
      int ret ;

      //open kvm
      printf("CALL_KVM\n");
      kvm = open("/dev/kvm", O_RDWR | O_CLOEXEC);

      //get kvm api version
      ret = ioctl(kvm, KVM_GET_API_VERSION, NULL);
      if (ret == -1)
            err(1, "KVM_GET_API_VERSION");
      if (ret != 12)
            errx(1, "KVM_GET_API_VERSION %d, expected 12", ret);

      printf("KVM_API_VERSION: %d \n", ret);

      //check kvm required extension
      ret = ioctl(kvm, KVM_CHECK_EXTENSION, KVM_CAP_USER_MEMORY);
      if (ret == -1)
            err(1, "KVM_CHECK_EXTENSION");
      if (!ret)
            errx(1, "Required extension KVM_CAP_USER_MEM not available");
      printf("KVM_CHECK_EXTENSION %d \n", ret);

      //create virtual machine
      vmfd = ioctl(kvm, KVM_CREATE_VM, (unsigned long)0);
      if (vmfd == -1)
            err(1, "KVM_CREATE_VM");
      //set memory slots index
```

```
	slot_num=-1;

	return ret;

}


int set_kvm_mem(void* addr,  void* mem, long length)
{//printf("%s\n",__func__);
	int ret ;


	//iterate slot_num
	slot_num++;

	/* Map mem to the second page frame (to avoid the real-mode IDT at
0). */
	struct kvm_userspace_memory_region region = {
		.slot = slot_num,
		.flags = (1UL << 0),//KVM_MEM_LOG_DIRTY_PAGES
		.guest_phys_addr = (uint64_t) addr,
		.memory_size = length,
		.userspace_addr = (uint64_t) addr,
	};

	//set kvm user memory region
	ret = ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &region);
	if (ret == -1)
		err(1, "KVM_SET_USER_MEMORY_REGION");

	//printf("KVM Memory region %d \n", slot_num);
	printf(".slot = %d \n", slot_num);
	//printf(".memory_size = %ld \n", length);
	//printf(".userspace_addr = %lx \n", (uint64_t)mem);
	printf(".guest_phys_addr = %lx \n", addr);
	printf(".region_end_addr = %lx \n", addr+length-1);

	return 0;
}

void kvm_mem()
{
	int ret ;
	struct kvm_userspace_memory_region region = {
		.slot = 0,
		.flags = 1,
		.guest_phys_addr = (uint64_t) 0x10000,
		.memory_size = 0x40000000, //4Gb
		.userspace_addr = (uint64_t) 0x10000
	};
	ret = ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &region);
	if (ret == -1)
		err(1, "KVM_SET_USER_MEMORY_REGION");

}
```

```
void create_kvm_vcpu(uint64_t rip, uint64_t rsp, uint64_t cr3, uint64_t
cr0 , uint64_t cr4)
{//printf("%s\n",__func__);

      int ret ;


      //create vcpu
      vcpufd = ioctl(vmfd, KVM_CREATE_VCPU, (unsigned long)0);
      if (vcpufd == -1)
            err(1, "KVM_CREATE_VCPU");

      /* Map the shared kvm_run structure and following data. */
      ret = ioctl(kvm, KVM_GET_VCPU_MMAP_SIZE, NULL);
      if (ret == -1)
            err(1, "KVM_GET_VCPU_MMAP_SIZE");
      mmap_size = ret;
      if (mmap_size < sizeof(*run))
                  errx(1, "KVM_GET_VCPU_MMAP_SIZE unexpectedly small");
      run = mmap(NULL, mmap_size, PROT_READ | PROT_WRITE, MAP_SHARED,
vcpufd, 0);
            if (!run)
            err(1, "mmap vcpu");

       /* Initialize CS to point at 0, via a read-modify-write of sregs.
*/
      ret = ioctl(vcpufd, KVM_GET_SREGS, &sregs);
      if (ret == -1)
            err(1, "KVM_GET_SREGS");
      sregs.cs.base = 0;
      sregs.cs.selector = 0;
      sregs.cr3 = (physaddr_t) cr3;
      sregs.cr0 = (physaddr_t) cr0;
      sregs.cr4 = (physaddr_t) cr4;
      sregs.efer = (physaddr_t) 0x4501;
      sregs.ldt.limit=0;
      ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
      if (ret == -1)
            err(1, "KVM_SET_SREGS");
      /* set debug registers */
      set_dr7(0);
      /* Initialize registers: instruction pointer for our code, addends,
and
      * initial flags required by x86 architecture. */
      regs.rip =rip;
      regs.rsp = rsp,
      regs.rflags = 0x2;

      ret = ioctl(vcpufd, KVM_SET_REGS, &regs);
      if (ret == -1)
            err(1, "KVM_SET_REGS");


}
```

```
int call_kvm()
{
      int ret ;

            printf("%lx\n",regs.rip);
            /* Repeatedly run code and handle VM exits. */
            while (1) {
                  ret = ioctl(vcpufd, KVM_RUN, NULL);
                  dump_vcpu_registers();
                  if (ret == -1)
                      err(1, "KVM_RUN");
                  switch (run->exit_reason) {
                  case KVM_EXIT_HLT:
                      puts("KVM_EXIT_HLT");
                      return 0;
                  case KVM_EXIT_IO:
                      if (run->io.direction == KVM_EXIT_IO_OUT && run-
>io.size == 1 && run->io.port == 0x3f8 && run->io.count == 1)
                        putchar(*(((char *)run) + run->io.data_offset));
                      else
                        errx(1, "unhandled KVM_EXIT_IO");
                      break;
                  case KVM_EXIT_FAIL_ENTRY:
                      errx(1, "KVM_EXIT_FAIL_ENTRY:
hardware_entry_failure_reason = 0x%llx",
                         (unsigned long long)run-
>fail_entry.hardware_entry_failure_reason);
                  case KVM_EXIT_INTERNAL_ERROR:
                        errx(1, "KVM_EXIT_INTERNAL_ERROR: suberror = 0x
%x", run->internal.suberror);
                  default:
                      errx(1, "exit_reason = 0x%x", run->exit_reason);
                  }
            }

            return 0;

}
```

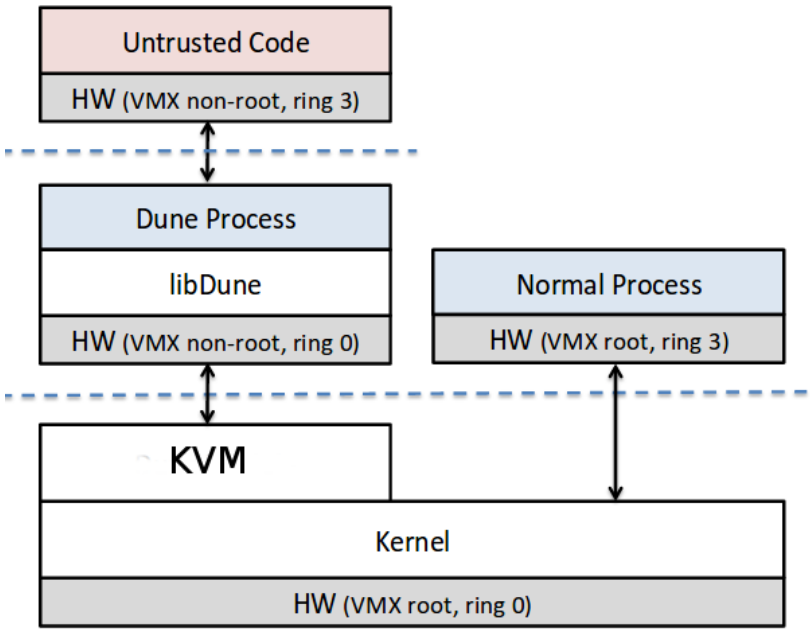
Και το αποτέλεσμα της οθόνης:

```
.
.
.
.
sp = 6fffff680
data.entry = 400890
400890
KVM vcpu REGS:
rax: 0 rbx: 0 rcx: 0 rdx: 0
rsi: 0 rdi: 0 rsp: 6fffff680 rbp: 0
r8: 0 r9: 0 r10: 0 r11: 0
rip: 400890 rflags: 2
```

```
KVM vcpu SREGS:
CS:
cs.base: 0 cs.limit: ffff cs.selector: 10 cs.type: b
cs.present: 1 cs.dpl: 0 cs.db: 0 cs.s: 1 cs.l: 1
cs.g: 1 cs.avl: 0 cs.unusable: 0 cs.padding: 0
DS:
ds.base: 0 ds.limit: ffff ds.selector: 18 ds.type: 3
ds.present: 1 ds.dpl: 0 ds.db: 0 ds.s: 1 ds.l: 1
ds.g: 1 ds.avl: 0 ds.unusable: 0 ds.padding: 0
CR:
cr0: 80010033 cr2: 0 cr3: 70495000 cr4: 206a0
GDT:
gdt.base: 7f9b02dde088 gdt.limit: 47 gdt.padding: 70214a0a
IDT:
idt.base: 70213080 idt.limit: fff idt.padding: 70214a1a
sandbox: KVM_EXIT_FAIL_ENTRY: hardware_entry_failure_reason = 0x80000021
```

Το script που γράψαμε για να τρέξουμε το libDune-KVM μέσα στο φάκελο του dune ήταν το εξής:

```
#!/bin/sh
sudo make clean
sudo make
sudo insmod kern/dune.ko
sudo ./apps/sandbox/sandbox ex2
sudo rmmod kern/dune.ko
sudo make clean
```



Σχήμα 6.1: *Η αρχιτεκτονική του συστήματος libDune-KVM (το σχήμα για τη μνήμη στο σύστημα αυτό είναι ίδιο με του Dune)*

6.2.1 Λεπτομέρειες υλοποίησης

Αναλυτικότερα, η ιδέα είναι ότι η εικονική μηχανή που θέλουμε να φτιάξουμε στο KVM είναι καθρέπτης της εικονικής μηχανής που φτιάχνει μόνο του το Dune. Για το σκοπό αυτό αντιγράφουμε στην εικονική μηχανή του KVM τη μνήμη και τους καταχωρητές ακριβώς όπως είναι στην εικονική μηχανή του Dune. Αφήνουμε δηλαδή την libdune να αρχικοποιήσει το περιβάλλον εκτέλεσης του εικονικού χρήστη. Εκεί έχουν ρυθμιστεί οι τιμές των καταχωρητών τμημάτων, στοίβας και των υπολοιπων ώστε να δείχνουν στις σωστές διευθύνσεις μνήμης καθώς και η ίδια η μνήμη ώστε αυτά να είναι έτοιμα για να χρησιμοποιηθούν από την εικονική μηχανή που υλοποιεί και τρέχει το dune module. Εμείς ακριβώς εκεί κόβουμε τη ροη της εκτέλεσης, αντιγράφουμε τις αρχικοποιήσεις αυτές στην εικονική μηχανή του KVM και δίνουμε την εκτέλεση στο KVM (Σχήμα 6.1). Έτσι το εκτελέσιμο ELF που φορτώνεται και αναλύεται αρχικά από την sandbox και από τις συναρτήσεις της libdune που αυτή καλεί, θα έχει φορτώθεί μαζί με όλες τις δομές που χρειάζονται για την εκτελεσή του στη μνήμη της εικονικής μηχανής του KVM, οι καταχωρητές της εικονικής μηχανής του KVM θα αντιστοιχούν σε αυτές τις σωστές θέσεις μνήμης, όπως είχαν αρχικοποιηθεί και για το dune module, και το εκτελέσιμο θα μπορεί να εκτελεστεί απόμονωμένο-sandboxed μέσα στην εικονική μηχανή του KVM. Η ροή των συναρτήσεων από τη libdune στο KVM περιγράφεται παρακάτω.

Αρχικά από τον κώδικα του sandbox στη boxer_main του main.c στο φάκελο dune/apps/sandbox καλούνται οι συναρτήσεις dune_init, και dune_enter (dune/libdune/entry.c). Στην dune_init γίνεται αρχικοποίηση του kvm module και δημιουργία της εικονικής μηχανής του KVM από εμάς μέσω της συνάρτησης init_kvm():

```
//init KVM - create vm machine
ret = init_kvm();
```

Στην dune_init γίνεται κλήση του dune module και κυρίως αρχικοποίηση της μνήμης. Αυτή περιλαμβάνει δέσμευση μνήμης για δομές που θα χρειαστούν για την εκτέλεση της εικονικής μηχανής του dune, για παράδειγμα του πίνακα σελίδων του επισκέπτη, δομές για τις κλήσεις συστήματος, δέσμευση του χώρου χρήστη όπου θα αποθηκευτεί και ο sandboxed εκτελέσιμος κώδικας, δομές χρήσιμες για τη λειτουργία των εικονικών cpu κ.α. Έχουμε μετατρέψει τον κώδικα της libdune ώστε να μη γίνεται η κλήση του dune module. Επίσης σε επίπεδο dune module αρχικοποιούνται όλοι οι καταχωρητές της εικονικής μηχανής. Τόσο τη μνήμη όσο και τους καταχωρητές αυτούς στο KVM θα τους αρχικοποιήσουμε αργότερα στη ροή του προγράμματος, αφού αρχικοποιηθεί η στοίβα και φορτωθεί και αναλυθεί το εκτελέσιμο ELF του sandbox και αφού υπολογιστούν οι διευθύνσεις data_entry και sp που μας χρειάζονται για την εκτέλεσή του. Στην dune_enter γίνεται αρχικοποίηση των κρίσιμων καταχωρητών της εικονικής μηχανής όπως ο cr3, και οι υπόλοιποι control registers, οι segment registers και αρχικοποιούνται και τα idt, ldt, gdt και

tss. Εκεί ακριβώς επεμβαίνουμε εμείς με κώδικα ώστε να αρχικοποιήσουμε τους καταχωρητές αυτούς στην εικονική μηχανή του KVM:

```
create_kvm_vcpu(conf.rip , conf.rsp , conf.cr3, 0x80010033, 0x000206a0 );
```

Δηλαδή μέσα στη συνάρτηση do_dune_enter που καλεί η dune_enter φτιάχνουμε ένα vcpu για την εικονική μηχανή του KVM με τις τιμές των καταχωρητών rip, rsp, cr3, cr0, και cr4 όπως τις έχει αρχικοποιήσει το dune. Για να μάθουμε τις τιμές των control registers cr0, cr4 απλά τις διαβάσαμε κατά την εκτέλεση της εικονικής μηχανής του dune και είναι σταθερές. Για τους καταχωρητές rip, rsp, cr3 το dune έχει κάνει πιο πριν μέσα στην do_dune_enter:

```
conf.rip = (__u64) &__dune_ret;
conf.rsp = 0;
conf.cr3 = (physaddr_t) pgroot;
```

και αυτές είναι η τιμές που χρησιμοποιούμε, αρχικά, γιατί λίγο πριν εκκινήσουμε το KVM θα ανανεωθουν οι rip και rsp. Το dune θέλει να μεταβεί η εκτέλεση σε εικονικό περιβάλλον από τώρα, γι’ αυτό ρυθμίζει από τώρα τους καταχωρητές αυτούς. Εμάς μας ενδιαφέρει να τρέξει σε εικονικό περιβάλλον μόνο το εκτελέσιμο ELF.

Πιο κάτω μέσα στην dune_boot κάνουμε:

```
//Astr set kvm's special registers
set_gdt(_gdtr.limit, _gdtr.base);
set_idt(_idtr.limit, _idtr.base);
set_cs(GD_KT);
set_ds(GD_KD);
set_es(GD_KD);
set_ss(GD_KD);
set_fs_base(percpu->kfs_base);
set_gs_base((unsigned long) percpu);
set_tss(GD_TSS);
```

που αντιστοιχούν στις τιμές που έχει ρυθμίσει πριν η dune_boot με τις συναρτήσεις:

```
struct tptr _idtr, _gdtr;
setup_gdt(percpu);

_gdtr.base  = (uint64_t) &percpu->gdt;
_gdtr.limit = sizeof(percpu->gdt) - 1;

_idtr.base = (uint64_t) &idt;
_idtr.limit = sizeof(idt) - 1;
```

και μέσω του assembly κώδικα παρακάτω κατευθείαν στο υλικό:

```
/*asm volatile (
      // STEP 1: load the new GDT
      "lgdt %0\n"
```

```
// STEP 2: initialize data segements
"mov $" __str(GD_KD) ", %%ax\n"
"mov %%ax, %%ds\n"
"mov %%ax, %%es\n"
"mov %%ax, %%ss\n"

// STEP 3: long jump into the new code segment
// initialize cs
"mov $" __str(GD_KT) ", %%rax\n"
"pushq %%rax\n"
"pushq $1f\n"
"lretq\n"
"1:\n"
"nop\n"

// STEP 4: load the task register (for safe stack switching)
"mov $" __str(GD_TSS) ", %%ax\n"
"ltr %%ax\n"

// STEP 5: load the new IDT and enable interrupts
"lidt %1\n"
"sti\n"

: : "m" (_gdtr), "m" (_idtr) : "rax");*/
```

και επίσης μέσω των συναρτήσεων:

```
// STEP 6: FS and GS require special initialization on 64-bit
wrmsrl(MSR_FS_BASE, percpu->kfs_base);
wrmsrl(MSR_GS_BASE, (unsigned long) percpu);
```

Έπειτα μέσα στη συναρτηση boxer_main (που καλείται από τη main) του sandbox καλούνται συναρτήσεις της libdune που φορτώνουν, αναλύουν και αποθηκεύουν το εκτελέσιμο ELF στη μνήμη, και φτιάχνουν τους trap handlers και τη στοίβα. Αυτά τα αφήνουμε να εκτελεστούν κανονικά. Δεν αφήνουμε να εκτελεστεί κώδικας ο οποίος εκτελείται σε προνομιούχο κατάσταση μέσω του dune και αφορά αλλαγές του dune σε επίπεδο hardware για να φτιάξει το περιβάλλον του. Εμείς αυτές τις αλλαγές τις ενσωματώνουμε στην εικονική μηχανή του KVM αντιγράφωντας σε αυτή, μέσω των ioctl του KVM, την κατάσταση του εικονικού επεξεργαστή του dune.

Έπειτα αφού εκτελεστούν οι προηγούμενες συναρτήσεις ρυθμίζουμε τους καταχωρητές ip, sp με τις σωστές τιμές μετά και από την αποθήκευση του ELF στη μνήμη και την αρχικοποίηση της στοίβας, ώστε ο ip να δείχνει στο start_address του ELF και ο sp στην αρχή της στοίβας. Επίσης, καθώς έχει ολοκληρωθεί η αρχικοποίηση της μνήμης από τη libdune, την αντιγράφουμε σε ένα μόνο μπλοκ, ενιαία, (για την αποφυγή σφαλμάτων) στο KVM:

```
//KVM VM set registers
set_vm_ip(data.entry);
```

```
set_vm_sp(sp);
//KVM VM set memory
kvm_mem();
```

Τέλος, αφού έχουν αρχικοποιηθεί όλοι οι καταχωρητές της εικονικής μηχανής του KVM όπως αρχικοποιούνται από το dune-module και τη libdune, και αφού έχει αντιγραφεί η μνήμη που φτιάχνει η libdune σε αυτήν της εικονικής μηχανής του KVM, δίνουμε την εκτέλεση στον vcpu του KVM:

```
ret = call_kvm();
printf("success\n");
```

Οι διευθύνσεις μνήμης της libdune είναι host virtual αλλα γίνονται στην ουσία guest physical της εικονικής μηχανής του dune προσαρμόζοντας τον πίνακα σελίδων και τους αντίστοιχους καταχωρητές αυτής.

ΚΕΦΑΛΑΙΟ 7ο

Μετρήσεις

7.1 Redis benchmark

Για να διαπιστώσουμε την απόδοση των λύσεων των προηγούμενων κεφαλαίων έπρεπε να δοκιμάσουμε να τρέξουμε κάποια εφαρμογή σε αυτές. Επιλέξαμε το Redis. Το Redis είναι ένα έργο βάσης δεδομένων “μέσα στη μνήμη”, ανοιχτού κώδικα, το οποίο υποστηρίζεται από την Redis Labs. Υποστηρίζει μεταφορές δεδομένων μεσω δικτύου, είναι βάση δεδομένων τοποθετημένη στη μνήμη, ώστε να βελτιώνεται η απόδοση, και αποθηκεύει κλειδιά με προαιρετική ανθεκτικότητα. Για το Redis υπάρχουν δύο εκτελέσιμα. Ένας redis server, που είναι και το κύριο εκτελέσιμο, ως το πρόγραμμα που υποστηρίζει τη βάση δεδομενων, και ένας redis client που συνδέεται στον server μέσω δικτύου, και στέλνει αιτήσεις πληροφορίας προς αυτόν. Για τους σκοπούς της εργασίας αυτής, ώστε να μπορέσουμε να μετρήσουμε την απόδοση των λύσεων απομόνωσης, που αναφέρθηκαν στα προηγούμενα κεφάλαια, τοποθετήσαμε τον redis server να εκτελείται μέσα στο “εικονικό”, ή όχι ακριβώς “εικονικό”, περιβάλλον απομόνωσης, και τον client τον εκτελέσαμε στον host, από όπου και στέλναμε τις αιτήσεις προς τον server. Στα επόμενα παρουσιάζονται τα αποτελέσματα της εκτέλεσης και οι μετρήσεις για κάθε διαφορετική περίπτωση. Για την πραγματοποίηση των μετρήσεων υλοποιήθηκαν δύο προγράμματα ics-ping.c και ics-set.c, χρησιμοποιώντας ως πρότυπο τον κώδικα του hiredis, τα οποία φαίνονται παρακάτω.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <hiredis.h>

int main(int argc, char **argv) {
    unsigned int n,j;
    redisContext *c;
    redisReply *reply;
    const char *hostname = "127.0.0.1";   // Host
    //const char *hostname = "10.1.1.1"; // Qemu VM
    //const char *hostname = "192.168.122.76"; // OSv VM
    int port = 6379;


    struct timeval timeout = { 1, 500000 }; // 1.5 seconds
    c = redisConnectWithTimeout(hostname, port, timeout);
    if (c == NULL || c->err) {
        if (c) {
            printf("Connection error: %s\n", c->errstr);
            redisFree(c);
        } else {
            printf("Connection error: can't allocate redis context\n");
        }
        exit(1);
    }


    /* number of commands */
    n =  (argc > 1) ? atoi(argv[1]) : 1000;
    printf("n : %d\n",n);
    for (j = 0; j < n; j++) {
      /* PING server */
      reply = redisCommand(c,"PING");
      //printf("PING: %s\n", reply->str);
      freeReplyObject(reply);
    }


    /* Disconnects and frees the context */
    redisFree(c);

    return 0;
}
```

ics-ping.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <hiredis.h>
```

```
int main(int argc, char **argv) {
    unsigned int n,j;
    redisContext *c;
    redisReply *reply;
    const char *hostname = "127.0.0.1";   // Host
    //const char *hostname = "10.1.1.1"; // Qemu VM
    //const char *hostname = "192.168.122.76"; // OSv VM
    int port = 6379;


    struct timeval timeout = { 1, 500000 }; // 1.5 seconds
    c = redisConnectWithTimeout(hostname, port, timeout);
    if (c == NULL || c->err) {
        if (c) {
            printf("Connection error: %s\n", c->errstr);
            redisFree(c);
        } else {
            printf("Connection error: can't allocate redis context\n");
        }
        exit(1);
    }


    /* number of commands */
    n =  (argc > 1) ? atoi(argv[1]) : 1000;
    printf("n : %d\n",n);
    for (j = 0; j < n; j++) {
      /* SET a[j]=j */
      reply = redisCommand(c,"SET a[%d] %d",j,j);
      //printf("SET: %s\n", reply->str);
      freeReplyObject(reply);
    }


    /* Disconnects and frees the context */
    redisFree(c);

    return 0;
}
```

ics-set.c

7.2 Redis - server στον host

Σε αυτή την περίπτωση ο redis server εκτελείται στον οικοδεσπότη, ώστε τα αποτελέσματα των μετρήσεων να αποτελούν μέτρο σύγκρισης για τις επόμενες περιπτώσεις, όπου ο redis server θα εκτελείται σε εικονικό-απομονωμένο περιβάλλον. Τα αποτελέσματα φαίνονται παρακάτω:

```
SERVER:
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-3.2.8/src$ ./redis-
server ../redis.conf
```

```
CLIENT:
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-ping 1000000
[sudo] password for astr:
n : 1000000

real  0m22.344s
user  0m3.268s
sys   0m10.412s

astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 10000

real  0m0.285s
user  0m0.052s
sys   0m0.132s

astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 100000

real  0m2.362s
user  0m0.456s
sys   0m1.088s

astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 1000000

real  0m21.349s
user  0m3.928s
sys   0m10.064s

astr@astr-HP-Pavilion-dv6-Notebook-PC:~$ cat /proc/3653/status
Name: redis-server
State:      R (running)
Tgid: 3653
Ngid: 0
Pid:  3653
PPid: 3652
TracerPid:  0
Uid:  0     0     0     0
Gid:  0     0     0     0
FDSize:     64
Groups:     0
NStgid:     3653
NSpid:      3653
NSpgid:     3652
NSsid:      3429
VmPeak:        99912 kB
VmSize:        99912 kB
VmLck:             0 kB
VmPin:             0 kB
VmHWM:         73244 kB
VmRSS:         73244 kB
```

7.3 Redis - server σε πλήρες VM

Σε αυτή την περίπτωση ο redis server εκτελείται μέσα σε μία πλήρη εικονική μηχανή QEMU/KVM:

```
sudo qemu-system-x86_64 -nographic -enable-kvm\
     -m 1024 -kernel ./amd64_kernel.img -initrd ./amd64_initrd.img \
     -append "root=/dev/vda  console=ttyS0 rootfstype=ext4 rw" -drive
file=./amd64.img,if=none,id=blk \
     -device virtio-blk-pci,scsi=off,drive=blk -device virtio-net-
pci,netdev=net0,mac=52:54:00:00:00:01 \
     -netdev tap,id=net0,ifname=tap0
sudo brctl addbr kvmbr
sudo ifconfig kvmbr 10.1.1.200/24 up
sudo brctl addif kvmbr tap0
sudo nano /proc/sys/net/ipv4/ip_forward (βάζουμ? 1)
sudo nano /proc/sys/net/ipv4/conf/kvmbr/proxy_arp (βάζουμ? 1)
sudo iptables -t nat -A POSTROUTING -s 10.1.1.1 -j MASQUERADE

SERVER:
root@kvmamd64:~/redis-3.2.8/src# ./redis-server ../redis.conf

CLIENT:
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-ping 1000000
[sudo] password for astr:
n : 1000000

real  1m2.639s
user  0m4.296s
sys   0m14.104s
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 10000
n : 10000

real  0m0.710s
user  0m0.104s
sys   0m0.128s
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 100000
n : 100000

real  0m6.403s
user  0m0.684s
sys   0m1.388s
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 1000000
n : 1000000

real  1m7.668s
user  0m5.964s
sys   0m14.304s
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-3.2.8/deps/hiredis$
cat /proc/4766/status
```

```
Name: qemu-system-x86
State:      R (running)
Tgid: 4766
Ngid: 0
Pid:  4766
PPid: 4765
TracerPid:  0
Uid:  0     0     0     0
Gid:  0     0     0     0
FDSize:     64
Groups:     0
NStgid:     4766
NSpid:      4766
NSpgid:     4765
NSsid:      3500
VmPeak:      4212808 kB
VmSize:      3942340 kB
VmLck:             0 kB
VmPin:             0 kB
VmHWM:       1156600 kB
VmRSS:       1142064 kB

DAEMONIZED SERVER (backround):
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 1000000
[sudo] password for astr:
n : 1000000

real  1m7.961s
user  0m6.272s
sys   0m13.628s
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time sudo ./ics-set 1000000
n : 1000000

real  1m5.943s
user  0m6.080s
sys   0m14.292s
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-3.2.8/src$ cat
/proc/4766/status
Name: qemu-system-x86
State:      R (running)
Tgid: 4766
Ngid: 0
Pid:  4766
PPid: 4765
TracerPid:  0
Uid:  0     0     0     0
Gid:  0     0     0     0
FDSize:     64
Groups:     0
NStgid:     4766
NSpid:      4766
NSpgid:     4765
NSsid:      3500
```

```
VmPeak:      4212808 kB
VmSize:      3917752 kB
VmLck:             0 kB
VmPin:             0 kB
VmHWM:       1156600 kB
VmRSS:       1141992 kB
```

7.4 Redis - server OSv appliance

Στην λύση αυτή επιλέξαμε το redis OSv appliance (unikernel-application image), το οποίο το τρέξαμε πάνω στις επεκτάσεις εικονικοποίησης του υλικού, σαν να ήταν μία κανονική εικονική μηχανή:

```
SERVER:
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/osv/osv$ sudo qemu-
system-x86_64 -nographic -m 512 -smp 4 -device virtio-blk-
pci,id=blk0,bootindex=0,drive=hd0 -drive
file=/home/astr/.capstan/instances/qemu/cloudius-osv-redis-
memonly/disk.qcow2,if=none,id=hd0,aio=native,cache=none -device virtio-
rng-pci -chardev stdio,mux=on,id=stdio,signal=off -device isa-
serial,chardev=stdio -netdev tap,id=hn0,script=scripts/qemu-
ifup.sh,vhost=on -device virtio-net-pci,netdev=hn0,id=nic0 -chardev
socket,id=charmonitor,path=/home/astr/.capstan/instances/qemu/cloudius-
osv-redis-memonly/osv.monitor,server,nowait -mon
chardev=charmonitor,id=monitor,mode=control -enable-kvm -cpu host,+x2apic

CLIENT:
astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time ./ics-ping 1000000
n : 1000000

real  0m39.733s
user  0m3.416s
sys   0m11.892s

astr@astr-HP-Pavilion-dv6-Notebook-PC:~/Desktop/redis-
3.2.8/deps/hiredis/ics-bench$ time ./ics-set 1000000
n : 1000000

real  0m44.075s
user  0m5.172s
sys   0m12.100s

astr@astr-HP-Pavilion-dv6-Notebook-PC:/$ cat proc/3491/status
Name: qemu-system-x86
State:      T (stopped)
Tgid: 3491
Ngid: 0
Pid:  3491
PPid: 3490
```

```
TracerPid:  0
Uid:  0     0     0     0
Gid:  0     0     0     0
FDSize:     64
Groups:     0
NStgid:     3491
NSpid:      3491
NSpgid:     3490
NSsid:      2831
VmPeak:      1123176 kB
VmSize:      1057668 kB
VmLck:             0 kB
VmPin:             0 kB
VmHWM:        105180 kB
VmRSS:        105180 kB
```

7.5 Redis - server σε kvmtool, dune – προβλήματα

Εδώ σημειώνουμε τα εξής:

Τον redis – server δεν καταφέραμε να τον τρέξουμε στο Dune. Παρουσιαζόταν μήνυμα σφάλματος “unhalded EPT violation” στην νεώτερη έκδοση του Dune, ενώ σε παλαιότερη έκδοση, η εκτέλεση του redis – server έκανε όλο το σύστημα να “κολλήσει”. Αυτό οφείλεται, ίσως, στο γεγονός, ότι το εύρος των κλήσεων συστήματος, που υποστηρίζεται από το Dune, είναι πιθανότατα περιορισμένο.

Στο kvmtool καταφέραμε να εκτελέσουμε τον redis – server. Όμως δεν καταφέραμε να συνδέσουμε τον client με τον server, καθώς το kvmtool δεν μπορούσε να εκτελεστεί με ενεργοποιημένο το virtio, κατά την εκκίνηση του. Επίσης δεν έβγαζε έξω στο host, την άκρη κάποιου network interface, από την εικονική μηχανή που εκκινούσε μέσα σε αυτό, ώστε να μπορεί έπειτα ο host, να φτιάξει μία γέφυρα προς αυτό το interface, και να συνδεθεί στον guest.

Έτσι οι λύσεις αυτές δεν μπόρεσαν να μετρηθούν.

7.6 Συμπεράσματα - Μελλοντική δουλειά - Προκλήσεις

Από τα παραπάνω φαίνεται ότι η λύση του OSv unikernel, για την απομόνωση διεργασιών, και την εκτέλεση τους “εικονικά”, δηλαδή πάνω στις επεκτάσεις εικονικοποίησης του υλικού, είναι η αποδοτικότερη, γεγονός που ήταν αναμενόμενο, καθώς η επιβάρυνση της εκτέλεσης του προγράμματος προς μέτρηση, εξαιτίας του περιβάλλοντος εκτέλεσης, είναι μικρότερη σε σχέση με αυτήν μιας πλήρους εικονικής μηχανής. Τα συγκριτικά αποτελέσματα για κάθε περίπτωση φαίνονται στον παρακάτω πίνακα:

	host	πλήρες VM	**OSv**
1000000 PINGs	0m22.344s	1m2.639s	**0m39.733s**
1000000 SETs	0m21.349s	1m5.943s	**0m44.075s**
Peak Memory Kilobytes	99912 kB	4212808 kB	**1123176 kB**
Memory RSS (Resident Set Size)	73244 kB	1141992 kB	**105180 kB**

Πίνακας 7.1: *Συγκριτικά αποτελέσματα απόδοσης λύσεων απομόνωσης διεργασιών. Αν χρησιμοποιηθεί το εργαλείο capstan για την εκτέλεση του OSv image η μέγιστη μνήμη της εκτελέσιμης διεργασίας πέφτει στην τιμή των 155888 kB, όμως ο χρόνος ανεβαίνει αισθητά στα 1m13.242s ενω και η μνήμη του qemu-system-x86_64 που καλείται από το capstan ανεβαίνει στα 2Gbyte.*

Μελλοντική δουλειά πάνω σε αυτό το αντικείμενο είναι, αρχικά, να λυθούν τα προβλήματα στις custom προσεγγίσεις του κεφαλαίου 6. Πραγματικά, ειδικά η λύση libdune-KVM φαντάζει ιδανική και πρωτότυπη, κυρίως για τους λόγους της φορητότητας εξαιτίας του KVM, αλλά και της ευχρηστιας που παρέχει η βιβλιοθήκη χώρου χρήστη libdune. Η λύση εκτελέσιμο binary – kvmtool είναι επίσης πολύ ενδιαφέρουσα, καθώς αποτελεί σίγουρα απαραίτητη αναβάθμιση του εργαλείου kvmtool, ενώ έχει πολλά να προσφέρει και από εκπαιδευτική άποψη σε αυτόν που θα ασχοληθεί.

Έπειτα, θα πρέπει να λυθούν τα προβλήματα που μας εμπόδισαν να μετρήσουμε κάποιες προσεγγίσεις. Κάποια από τα προβλήματα αυτά είναι εγγενή των εργαλείων που χρησιμοποιήθηκαν, αλλά σίγουρα μπορούν να γίνουν βελτιώσεις των υπαρχουσών εκδόσεων.

Επίσης θα μπορουσαν να γίνουν μετρήσεις ώστε να μελετηθεί η επίδοση των τεχνικών υβριδικής εικονικοποίησης.

Τέλος ενδιαφέρουσα πρόκληση θα ήταν να μεταφερθούν όλες οι παραπάνω μέθοδοι απομόνωσης και σε άλλες αρχιτεκτονικές, στην πράξη. Δηλαδή να μεταφερθούν, να εγκαταστηθούν, να εκτελεστούν, και να μετρηθούν σε αυτές. Πιο συγκεκριμένα θα μας ενδιέφερε να εγκαταστηθούν τα εργαλεία KVM, dune, kvmtool, OSv, και Redis Server σε ARM, και να γίνει η μετάβαση της προσέγγισης libdune-KVM σε ARM αρχιτεκτονική.

Όποιες και να είναι οι ιδέες που θα προκύψουν, η ενασχόληση με τις τεχνολογίες εικονικοποίησης, και με τις τεχνικές απομόνωσης διεργασιών είναι, το λιγότερο, συναρπαστική!

Βιβλιογραφία:

1)https://el.wikipedia.org/wiki/Ηλεκτρονικός_υπολογιστής κεφάλαιο 1.1,

2)https://el.wikipedia.org/wiki/Διαδίκτυο κεφάλαιο 1.2,

3)https://en.wikipedia.org/wiki/Converged_infrastructure κεφάλαιο 1.3 ,μετάφραση

4)https://en.wikipedia.org/wiki/Cloud_computing κεφάλαιο 1.4, μετάφραση

5)https://en.wikipedia.org/wiki/Virtual_machine κεφάλαιο 1.5, μετάφραση

6)https://el.wikipedia.org/wiki/Εικονικοποίηση κεφάλαιο 2, (εκτός παραγράφου 2.2.4)

7)https://en.wikipedia.org/wiki/Hardware-assisted_virtualization παράγραφος 2.2.4, μετάφραση

8)https://en.wikipedia.org/wiki/X86_virtualization “Intel virtualization (VT-x)” και ”AMD virtualization (AMD-V)” παράγραφος 3.1, μετάφραση και παράγραφος 3.2, μετάφραση

9)http://www.hardwaresecrets.com/everything-you-need-to-know-about-the-intel-virtualization-technology/2/ παράγραφος 3.1, μετάφραση

10)http://www.amd.com/Documents/AMD_WP_Virtualizing_Server_Workloads-PID.pdf “VIRTUALIZING SERVER WORKLOADS ”, AMD white paper, chapter: “AMD Virtualization (AMD-V) technology” παράγραφος 3.2, μετάφραση

11)https://www.arm.com/files/pdf/System-MMU-Whitepaper-v8.0.pdf “Virtualization is Coming to a Platform Near You”, ARM white paper, Roberto Mijat, Andy Nightingale, chapter “ARM Virtualization Extensions” κεφάλαιο 3.3 μετάφραση.

12)https://en.wikipedia.org/wiki/QEMU κεφάλαιο 4.1 μετάφραση.

13)http://www.qemu.org/ κεφάλαιο 4

14)https://www.linux-kvm.org/page/Main_Page κεφάλαιο 4

15)https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine κεφάλαιο 4.2 (εκτός παραγράφου 4.2.4) μετάφραση

16)https://www.kernel.org/doc/Documentation/virtual/kvm/api.txt κεφάλαιο 4.2.4 μετάφραση

17)https://el.wikipedia.org/wiki/XEN κεφάλαιο 4.3

18)https://el.wikipedia.org/wiki/Διεργασία_(υπολογιστές) κεφαλαια 5.1, 5.2

19)https://el.wikipedia.org/wiki/Χρονοπρογραμματισμός_ΚΜΕ κεφάλαιο 5.2

20)https://en.wikipedia.org/wiki/Shared_memory κεφάλαιο 5.3 μεταφραση

21)https://en.wikipedia.org/wiki/Memory_protection κεφάλαιο 5.4 μετάφραση

22)https://en.wikipedia.org/wiki/Protected_mode κεφάλαιο 5.4 μετάφραση

23)https://en.wikipedia.org/wiki/Sandbox_(computer_security) κεφάλαιο 5.5 μετάφραση

24)https://en.wikipedia.org/wiki/Process_isolation κεφάλαιο 5.5 μετάφραση

25)https://cs.brown.edu/research/pubs/theses/masters/2012/ayer.pdf “KVMSandbox: Application-Level Sandboxing with x86 Hardware Virtualization and KVM” , Andrew Ayer, Brown University, Brown Computer Science, 2012 κεφάλαιο 5

26)http://unikernel.org/, http://unikernel.org/projects/ κεφάλαιο 5.5.1 μετάφραση, κεφάλαιο 5.6.3 μετάφραση

27)https://en.wikipedia.org/wiki/Unikernel κεφάλαιο 5.5.1 μετάφραση

28)https://github.com/penberg/linux-kvm/tree/master/tools/kvm κεφάλαιο 5.6.1 μετάφραση

29)http://dune.scs.stanford.edu/belay:dune.pdf “Dune: Safe User-level Access to Privileged CPU Features” Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazi`eres, Christos Kozyrakis, Stanford University, 2012, κεφάλαιο 5.6.2 μετάφραση

30)https://github.com/ramonza/dune Dune code κεφάλαιο 5

31)http://osv.io/ κεφάλαιο 5.6.3 μετάφραση

32)https://github.com/chyyuu/dune-arm, Dune for ARM, Yu Chen, Tsinghua University, Beijing China κεφάλαιο 5

33)https://lwn.net/Articles/658511/, https://lwn.net/Articles/658512/ “Using the KVM API”, Josh Triplett, κεφάλαιο 6.2

34)https://redis.io/ κεφάλαιο 7

35)https://en.wikipedia.org/wiki/Redis κεφάλαιο 7.1

36)https://github.com/redis/hiredis κεφάλαιο 7

37)www.cslab.ntua.gr/

38)http://artemis-new.cslab.ece.ntua.gr:8080/jspui/bitstream/123456789/7101/1/DT2014-0237.pdf “V4vsockets: Μηχανισμός Αποδοτικής Ενδο-επικοινωνίας Εικονικών Μηχανών Χαμηλής Επιβάρυνσης”, Αλιφιεράκη Ιωάννα–Μαρία, Κοζύρης Νεκτάριος, Εργαστήριο Υπολογιστικών Συστημάτων Ε.Μ.Π. , Ιούλιος 2014

39)https://lwn.net/Articles/557132/ “Supporting KVM on the ARM architecture.” lwn.net July 3, 2013 by Christoffer Dall and Jason Nieh

40)https://en.wikipedia.org/wiki/Single-root_input/output_virtualization κεφάλαιο 3.4

41)https://translate.google.gr/ , Μετάφραση Google

42)https://repo.cslab.ece.ntua.gr/ics-forth-diploma-on-virtualization Project code

43)https://www.linaro.org/blog/core-dump/on-the-performance-of-arm-virtualization/ “On the Performance of ARM Virtualization”, www.linaro.org, by Christoffer Dall, June 16, 2016, in Core Dump