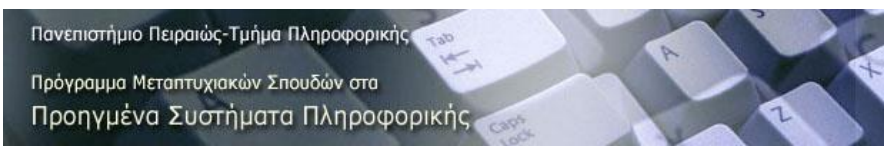
Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εικονικοποίηση ενσωματωμένων συστημάτων: Xen σε επεξεργαστή ARM** **Embedded Systems Virtualization: Xen on ARM processor**
Ονοματεπώνυμο Φοιτητή	**Κοκκίνης Αθανάσιος του Κωνσταντίνου**
Αριθμός Μητρώου	**ΜΠΣΠ14036**
Κατεύθυνση	**Τεχνολογίες Διαχείρισης Ασφάλειας**
Επιβλέπων	**Μιχαήλ Ψαράκης, Επίκουρος Καθηγητής**



Ημερομηνία Παράδοσης **Σεπτέμβριος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ψαράκης Μιχαήλ	Δουληγέρης Χρήστος	Κοτζανικολάου Παναγιώτης
Επίκουρος Καθηγητής	Καθηγητής	Επίκουρος Καθηγητής

Περίληψη

Η εισαγωγή της τεχνολογίας εικονικών μηχανών στο τομέα των ενσωματωμένων υπολογιστικών συστημάτων αποτελεί μια σπουδαία εξέλιξη. Η απομόνωση των κρίσιμων εφαρμογών από το υπόλοιπο περιβάλλον, σε συνδυασμό με την δυνατότητα χρήσης διαφορετικών λειτουργικών συστημάτων, αποτελούν μια καινοτομία για τον χώρο των ενσωματωμένων. Όμως η παραπάνω τεχνολογία έρχεται με κόστος καθώς εισάγει καθυστέρηση στο σύστημα. Σε αυτήν τη μεταπτυχιακή διατριβή μελετάται η νέα τεχνολογία εικονικοποίησης των επεξεργαστών της ARM σε περιβάλλον Xen. Πιο συγκεκριμένα γίνεται ανάλυση της γενιάς ARMv7, που εμπεριέχει σημαντικές καινοτομίες για την μείωση της καθυστέρησης που προκαλείται σε εικονικές μηχανές. Στόχος είναι η ανάλυση της καθυστέρησης καθώς και ο εντοπισμός της αιτίας που την προκαλεί. Εκτός από την θεωρητική μελέτη της αρχιτεκτονικής του ARMv7 και του Xen, έγινε και η πειραματική αξιολόγηση των τεχνολογιών με χρήση διαφόρων μετροπρογραμμάτων αξιολόγησης.

Abstract

The introduction of virtual machine technology in the field of embedded system is an significant evolution. The isolation of the critical applications from the rest computing system in addition with the ability of using different operating systems is a novel step in the area of embedded systems. However, the above technology comes at a cost as it adversely affects the system performance. In this thesis we study a new generation of ARM processors with enhanced features for virtualization support in combination with the Xen hypervisor. More specifically, we study the virtualization of the Armv7 processors, that contain architectural enhancements for the elimination of the performance degradation caused by virtualization. The goal of this master thesis is the analysis and the identification of the components which degrade the system performance. To assess the theoretical study of the ARMv7 and Xen hypervisor, we also performed an analytical set of experiments using various software benchamrks.

Περιεχόμενα

Περιεχόμενα πινάκων

Περιεχόμενα εικόνων

Εισαγωγή

Η ανάγκη για ασφάλεια και ευελιξία στα ενσωματωμένα υπολογιστικά συστήματα καθίσταται ολοένα και πιο επιτακτική. Έτσι, προσπαθώντας να καλύψουμε αυτήν την ανάγκη μεταφέρουμε στα ενσωματωμένα συστήματα τεχνικές που χρησιμοποιούνται με επιτυχία για χρόνια στα κοινά υπολογιστικά συστήματα.

Η παρούσα διατριβή έχει ως στόχο την μελέτη, εφαρμογή και ανάλυση της απόδοσης της τεχνολογίας της εικονικοποίησης (virtualization) σε ενσωματωμένα συστήματα. Η εικονικοποίηση δίνει την δυνατότητα για ταυτόχρονη εκτέλεση πολλαπλών εικονικών μηχανών (virtual machines) σε ένα φυσικό υπολογιστικό σύστημα. Βασικό πλεονέκτημα αυτής της τεχνολογίας στα ενσωματωμένα είναι η δυνατότητα απομόνωσης και προστασίας των κρίσιμων εφαρμογών του συστήματος (critical applications). Επίσης, δίνεται η δυνατότητα εκτέλεσης πολλαπλών λειτουργικών συστημάτων μέσα στο ίδιο φυσικό υπολογιστικό σύστημα.

Για τον σκοπό αυτό έχουν αναπτυχθεί διάφορα μοντέλα υπερεπόπτη (hypervisor) που έχουν σαν στόχο την βέλτιστη διαχείριση του υλικού του συστήματος. Υπάρχουν δυο βασικές προσεγγίσεις πάνω στην τεχνολογία της εικονικοποίησης. Ο πρώτος τύπος υπερεπόπτη τρέχει απευθείας στο υλικό του συστήματος και αντικαθιστά το κλασσικό λειτουργικό σύστημα. Ο δεύτερος τύπος εκτελείται μέσα από ένα παραδοσιακό λειτουργικό σύστημα. Στην συγκεκριμένη διατριβή ασχολούμαστε με τον Xen ο οποίος είναι υπερεπόπτης τύπου ένα.

Ο υπερεπόπτης Xen είναι απλός και σταθερός και μπορεί να εκτελεστεί σε μεγάλο εύρος διαφορετικών επεξεργαστών. Επιπρόσθετα υποστηρίζει μια μεγάλη ποικιλομορφία λειτουργικών συστημάτων καθώς και διαφορετικές τεχνικές για την επίτευξη της βέλτιστης απόδοσης της εικονικοποίησης. Όπως κάθε υπερεπόπτης έτσι και ο Xen έχει σαν στόχο την απομόνωση των εκάστοτε εικονικών μηχανημάτων ώστε οι αλλαγές που πραγματοποιούν να μην επηρεάζουν άμεσα το υπόλοιπο σύστημα. Για τον λόγο αυτό υπάρχει ιεραρχία δικαιωμάτων σε κάθε συστατικό μέρος του συστήματος.

Κατά την εκκίνηση του συστήματος ο Xen αναλαμβάνει την δημιουργία της πρώτης εικονικής μηχανής (virtual machine) που την ονομάζει Domain-0. Η μηχανή Domain-0 είναι η μοναδική εικονική μηχανή που έχει άμεση πρόσβαση στο υλικό. Μέσα από το Domain-0 ο Xen επιτρέπει στον χρήστη την διαχείριση του συστήματος και για τον λόγο αυτό το Domain-0 έχει αυξημένα δικαιώματα σε σχέση με τα υπόλοιπα εικονικά μηχανήματα. Από την κονσόλα του Domain-0 ο χρήστης είναι σε θέση να δημιουργεί νέα εικονικά μηχανήματα τα οποία ονομάζονται Domain-U. Τα Domain-U έχουν μειωμένα δικαιώματα σε σχέση με το Domain-0 και επιπρόσθετα δεν έχουν άμεση πρόσβαση στο υλικό του συστήματος. Για τον λόγο αυτό ο Domain-0 διαθέτει ειδικούς οδηγούς(drivers) ώστε να εξυπηρετεί τα αιτήματα για προσπέλαση υλικού που αποστέλλουν τα Domain-U.

Η έλευση των νέων επεξεργαστών της ARM της γενιάς ARMv7 και ARMv8 που υποστηρίζουν εικονικοποίηση ήρθε και άλλαξε ριζικά την δομή του Xen για ARM επεξεργαστές. Η βασική προσθήκη αυτής της νέα αυτής αρχιτεκτονικής επεξεργαστών είναι η εισαγωγή της εντολής HVC (HyperVisor Call) η οποία επιτρέπει την αποδοτικότερη επικοινωνία του πυρήνα των εικονικών μηχανών με τον υπερεπόπτη.

Εξαιτίας αυτής της εξέλιξης δημιουργήθηκε μια νέα έκδοση του Xen που εκμεταλλεύεται στο έπακρο τις δυνατότητες των νέων αυτών επεξεργαστών. Η νέα αυτή προσέγγιση είναι η Xen-on-Arm ο οποίος έχει σαν στόχο την επίλυση των προβλημάτων που είχαν δημιουργηθεί από την μακρόχρονη ανάπτυξη του Xen για x86. Η συγκεκριμένη τεχνική χωρίζει τα δικαιώματα σε τρία επίπεδα. Στο επίπεδο 0 εκτελείται ο Xen και έχει τα περισσότερα δικαιώματα . Στο επίπεδο ένα έχουμε την εκτέλεση του πυρήνα των εικονικών μηχανών. Τέλος στο επίπεδο δυο έχουμε την εκτέλεση των εφαρμογών των εικονικών μηχανών. Με αυτό τον τρόπο είναι εφικτή η απομόνωση των εικονικών μηχανών. Όμως η απομόνωση των εικονικών μηχανών έρχεται με κόστος καθώς εισάγει καθυστέρηση στο σύστημα.

Στα πλαίσια αυτής της μεταπτυχιακής διατριβής μετρήθηκε πειραματικά η καθυστέρηση που εισάγει ο Xen σε ένα ενσωματωμένο σύστημα με επεξεργαστή ARM. Η ενσωματωμένη πλατφόρμα που χρησιμοποιήθηκε είναι η πλακέτα cubieboard2 το οποίο περιέχει δυο πυρήνες

ARM Cortex-A7. Για την αξιολόγηση του συστήματος έγινε χρήση των μετροπρογραμμάτων (benchmarks) LMBench, που χρησιμοποιείται για την αξιολόγηση της απόδοσης λειτουργικών συστημάτων και miBench, που χρησιμοποιείται για την αξιολόγηση της απόδοσης ενσωματωμένων συστημάτων καθώς και του DMTCP, μιας εφαρμογής checkpointing.

Τα αποτελέσματα των πειραμάτων έδειξαν ότι οι μηχανισμοί διαχείρισης τόσο του επεξεργαστή όσο και της μνήμης έχουν μειώσει σε μεγάλο βαθμό την καθυστέρηση που εισάγει ο Xen on ARM. Όμως δεν ισχύει το ίδιο στην διαχείριση των συσκευών εισόδου/εξόδου (I/O devices). Στα πλαίσια των πειραμάτων παρατηρήθηκε ότι ο Xen on ARM εισάγει σημαντική καθυστέρηση στην προσπέλαση των περιφερειακών συσκευών.

Η εργασία εκτός της εισαγωγής, περιλαμβάνει τα παρακάτω κεφάλαια: στο Κεφ.1 περιγράφεται η ιδέα της εικονικοποίησης και οι βασικές της τεχνικές. Στα Κεφ.2 και 3 περιγράφεται αναλυτικά ο υπερεπόπτης Xen, και η έκδοσή του για επεξεργαστές ARM (Xen-on-ARM), αντίστοιχα. Στα Κεφ.4 και 5 περιγράφεται το περιβάλλον εκτέλεσης των πειραμάτων και παρουσιάζονται και αναλύονται τα πειραματικά αποτελέσματα, ενώ τέλος στο Κεφ.6 εξάγονται συμπεράσματα για την νέα αυτή τεχνολογία.

1 Εικονικοποίηση

Στην επιστήμη της πληροφορικής η εικονικοποίηση αναφέρεται στην δημιουργία μιας εικονικής έκδοσης των διαφόρων συνθετικών ενός υπολογιστικού συστήματος όπως το υλικό και το λειτουργικό. Η εικονικοποίηση ξεκίνησε την δεκαετία του ‘60, ως ένας λογικός καταμερισμός των πόρων των υπερυπολογιστών ανάμεσα στις διάφορες διεργασίες. Την εικονικοποίηση του συστήματος αναλαμβάνει ένας υπερεπόπτης. Παρακάτω ακολουθεί μια σύντομη ιστορική αναδρομή και στην συνέχεια γίνεται αναφορά τόσο στις τεχνικές εικονικοποίησης όσο και στους τύπους των υπερεποπτών.

1.1 Ιστορική αναδρομή

Ο όρος εικονικοποίηση γεννήθηκε την δεκαετία του ΄60 από την IBM στο ερευνητικό κέντρο του Cambridge. Το 1964 το ίδιο ερευνητικό κέντρο ξεκινά την δημιουργία του CP-40 που αποτελεί το πρώτο σύστημα που κάνει χρήση εικονικοποίησης. Την αμέσως επόμενη χρονιά η IBM ανακοινώνει την δημιουργία του IBM System/360-67 ο οποίος είναι ο πρώτος επεξεργαστής 32-bit με εικονική μνήμη. Η επίσημη κυκλοφορία του πρώτου συστήματος με τον συγκεκριμένου επεξεργαστή γίνεται τον Ιούνιο του 1966. Παράλληλα με την κυκλοφορία του συστήματος γίνεται και η ανάπτυξη του CP-67 ώστε να γίνει αντικατάσταση το CP-40.

Το 1970 η IBM ανακοινώνει ένα νέο σύστημα το System/370 και προχωρά στην δημιουργία μιας νέας πλατφόρμας της CP-370 ώστε να γίνει αντικατάσταση της CP-67. Η κυκλοφορία του συγκεκριμένου συστήματος γίνεται με επιτυχία την επόμενη χρόνια. Το 1973 ανακοινώνεται το VM/370 το οποίο αποτελεί το πρώτο σύστημα με την ικανότητα να εκτελεί εικονικές μηχανές μέσα σε εικονικές μηχανές. Επίσης είναι το πρώτο σύστημα που υποστηρίζει παραεικονικοποίηση (paravirtualization) .

Την επόμενη δεκαετία και πιο συγκεκριμένα το 1985 η Intel ανακοινώνει την κυκλοφορία του Intel 80286 που είναι βασισμένος στον AT&T6300+. Ο Intel 80286 επιτρέπει την εκτέλεση εικονικής μηχανής 86-DOS μέσα σε ένα Unix V Release 2 λειτουργικό σύστημα. Η συγκεκριμένη πλατφόρμα μπορούσε να εκτελέσει οποιοδήποτε λειτουργικό σύστημα ή πρόγραμμα που ήταν γραμμένο σε εντολές 8086. Την επόμενη χρονιά γίνεται η κυκλοφορία του Merge/386 που αποτελεί την αναβάθμιση του προηγούμενου συστήματος της Intel και έχει την δυνατότητα εκτέλεσης πολλαπλών εικονικών μηχανών.

Το 1994 η Bochs προχωρά στην δημιουργία του πρώτου εικονικού Bios. Παράλληλα γίνεται ανάπτυξη τεχνικών ώστε να καταστεί δυνατή η εκτέλεση αλγορίθμων διαφορετικών αρχιτεκτονικών σε περιβάλλον x86.Με αυτόν τον τρόπο έχουμε την δημιουργία της πρώτης πλατφόρμας με δυνατότητα εκτέλεσης διαφορετικών λειτουργικών συστημάτων. Το 1998 έχουμε την κυκλοφορία του USENIX’98 μιας πλατφόρμας ικανής να εκτελεί Linux 2.0.30 και Solaris 2.7 από τον ίδιο σκληρό δίσκο. Επίσης την ίδια χρονιά έχουμε την δημιουργία της VMware που την επόμενη χρονιά δημιουργεί την πρώτη της πλατφόρμα.

Το 2003 έχουμε την κυκλοφορία του πρώτου υπερεπόπτη Xen . Το 2005 η VMware κυκλοφορεί τον πρώτο VMware Player και την επόμενη χρονιά τον VMware Server. Από το 2005 μέχρι και σήμερα ο κλάδος της εικονικοποίησης συναντά ραγδαία ανάπτυξη. Σημαντικό ρόλο σε αυτό έπαιξε η ανάπτυξη της υπολογιστικής νέφους.

1.2 Τεχνικές εικονικοποίησης

Η κυρίαρχη μορφή εικονικοποίησης σήμερα είναι η εικονικοποίηση διακομιστή. Υπάρχουν διάφορες υλοποιήσεις για όλες τις αρχιτεκτονικές των επεξεργαστών. Όμως παρά την πληθώρα υλοποιήσεων που κυκλοφορούν στην αγορά, είναι εφικτό να χωριστούν σε τρεις βασικές κατηγορίες:

- Πλήρης εικονικοποίηση (full virtualization)

- Παραεικονικοποίηση (paravirtualization)
- Εικονικοποίηση επιπέδου λειτουργικού συστήματος (OS-level virtualization)

1.2.1 Πλήρης εικονικοποίηση

Σε αυτήν την τεχνική γίνεται εικονικοποίηση επαρκούς τμήματος του πραγματικού υλικού ώστε να επιτραπεί η φιλοξενία λειτουργικών συστημάτων χωρίς την τροποποίησή τους. Το αποτέλεσμα είναι ένα σύστημα στο οποίο το λειτουργικό έχει την δυνατότητα να εκτελεστεί απευθείας στο υλικό.

Η συγκεκριμένη τεχνική παρέχει διάφορα θετικά στοιχεία. Αρχικά, όπως προαναφέρθηκε, τα περισσότερα λειτουργικά συστήματα μπορούν να εκτελεστούν χωρίς αλλαγές. Επιπρόσθετα, παρέχει σχεδόν φυσικές επιδόσεις σε επίπεδο μνήμης καθώς και σε επίπεδο κεντρικού επεξεργαστή. Τέλος, παρέχει πλήρη απομόνωση των εικονικών μηχανών.

Το λειτουργικό σύστημα δεν έχει τροποποιηθεί και για αυτό εκτελεί εντολές με αυξημένα δικαιώματα. Για να μπορέσει να καταστεί αυτό δυνατό ο υπερεπόπτης αναλαμβάνει την διαχείριση των εντολών που εκτελούνται από τις εκάστοτε εικονικές μηχανές, ώστε οι αλλαγές που θα πραγματοποιηθούν να μην επηρεάζουν το υπόλοιπο σύστημα. Ακόμη μια βασική διαδικασία που αναλαμβάνει να εκτελέσει ο υπερεπόπτης είναι οι κλήσεις συστήματος. Όταν μια εικονική μηχανή εκτελεί μια κλήση συστήματος ο υπερεπόπτης αναλαμβάνει να την διαχειριστεί με τέτοιο τρόπο ώστε οι αλλαγές που θα επιφέρει να μην επηρεάσουν το υπόλοιπο σύστημα.

Όμως η συγκεκριμένη προσέγγιση απαιτεί συγκεκριμένο υλικό και λογισμικό ώστε να είναι σε θέση να εφαρμοστεί εξαιτίας της πολύπλοκης διαχείρισης δικαιωμάτων που απαιτεί. Ένα βασικό μειονέκτημα της συγκεκριμένης τεχνικής είναι η αδυναμία εκτέλεσης της σε περιβάλλον x86. Η αιτία αυτού του προβλήματος είναι η αδυναμία εκτέλεσης συγκεκριμένων εντολών όπως οι κλήσεις συστήματος. Για τον λόγο αυτό οι διάφοροι υπερεπόπτες έχουν κατασκευάσει μηχανισμούς ώστε να αντικαθιστούν αυτές τις εντολές με εντολές που μπορούν να εκτελεστούν σε περιβάλλον x86. Επιπρόσθετα από το 2005 η Intel και η AMD έχουν κατασκευάσει ειδικές εκδώσεις επεξεργαστών x86 που διαθέτουν ειδικό υλικό ώστε να μπορούν να υποστηρίξουν πλήρη εικονικοποίηση.

Ένα βασικό πρόβλημα που προκύπτει από την απευθείας εκτέλεση του λειτουργικού στο υλικό είναι αυτό της διαχείρισης των I/O. Κάθε αλλαγή στο εκάστοτε σύστημα θα πρέπει να επηρεάζει μόνο την εικονική μηχανή που εκτελεί την αλλαγή. Για τον λόγο αυτό κάθε I/O πρέπει πρώτα να ελέγχεται και μετά να εκτελείται με αποτέλεσμα η συγκεκριμένη διαδικασία να εισάγει σημαντική καθυστέρηση στο σύστημα.

1.2.2 Παραεικονικοποίηση

Η τεχνική αυτή προσφέρει μερική εξομοίωση του υλικού. Ο υπερεπόπτης δεν προσομοιώνει επακριβώς το υλικό αλλά παρέχει στις εικονικές μηχανές ένα API, μία προγραμματιστική διασύνδεση, ώστε να επιτρέπει την εκτέλεση ενός τροποποιημένου λειτουργικού συστήματος σχεδιασμένο για εκτέλεση από τον συγκεκριμένο επόπτη. Το προαναφερθέν API ονομάζεται διασύνδεση υπερκλήσεων (hypercall connection) 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 (hypercalls) και να αναμένει απαντήσεις από τον υπερεπόπτη. Η συγκεκριμένη τεχνική έγινε γνωστή από τον υπερεπόπτη Xen και σήμερα τείνει να υποστηριχθεί από όλους τους υπερεπόπτες.

Η τεχνική αυτή είναι απλούστερη και για τον λόγο αυτό ευκολότερα υλοποιήσιμη από την πλήρη εικονικοποίηση. Η συγκεκριμένη τεχνική απαιτεί την διαμόρφωση του πυρήνα του λειτουργικού συστήματος ώστε να αντικατασταθούν οι εντολές που δεν μπορούν να εικονικοποιηθούν, όπως για παράδειγμα οι κλήσεις συστήματος. Οι εντολές που δεν είναι σε

θέση να εκτελεστούν άμεσα αντικαθίστανται από υπερκλήσεις τις οποίες αναλαμβάνει να τις εκτελέσει ο υπερεπόπτης.

Ο υπερεπόπτης παρέχει επίσης μηχανισμούς και για άλλες σημαντικές εργασίες του πυρήνα, όπως η διαχείριση της μνήμης και ο χειρισμός των διακοπών. Με αυτόν τον τρόπο μειώνεται σημαντικά η χρήση εντολών με αυξημένα δικαιώματα οι οποίες είναι υπεύθυνες κατά κύριο λόγο για τα προβλήματα επιδόσεων στην πλήρη εικονικοποίηση.

Η συγκεκριμένη τεχνική όμως έρχεται με κόστος διότι απαιτεί αρκετές τροποποιήσεις σε επίπεδο λειτουργικού συστήματος ώστε να εφαρμοστεί. Για αυτό τον λόγο δεν είναι ιδιαίτερα ευέλικτη και δεν υποστηρίζει λειτουργικά συστήματα κλειστού κώδικα όπως τα Windows.

1.2.3 Εικονικοποίηση επιπέδου λειτουργικού συστήματος

Η συγκεκριμένη μέθοδος δίνει την δυνατότητα στον πυρήνα ενός λειτουργικού συστήματος να έχει πολλαπλά και απομονωμένα περιβάλλοντα χρήσης (user space).

Η συγκεκριμένη τεχνική αποτελείται από ένα λειτουργικό σύστημα χωρίς αλλαγές και για τον λόγο αυτό είναι αρκετά αποδοτική. Εξαιτίας της φύσης της δεν εισάγει καθυστέρηση στο σύστημα. Ακόμα είναι πιο εύκολη η διαχείριση και η ενημέρωση των εικονικών μηχανών καθώς υπάρχει μόνο ένας πυρήνας. Σε όλα αυτά πρέπει να προστεθεί ότι υποστηρίζει όλα τα λειτουργικά συστήματα.

Όπως είναι κατανοητό αυτή η τεχνική δεν μπορεί να υποστηρίξει την φιλοξενία διαφορετικού τύπου λειτουργικών συστημάτων. Επιπρόσθετα, εξαιτίας της ύπαρξης ενός και μόνο πυρήνα, δεν υποστηρίζει σε ίδιο βαθμό με τις άλλες τεχνικές εικονικοποίησης, την απομόνωση και την ασφάλεια των επιμέρους εικονικών μηχανών. Τέλος είναι δύσκολη η διαχείριση των πόρων στα επιμέρους εικονικά μηχανήματα.

1.3 Τύποι υπερεπόπτη

Από την πλευρά τους οι υπερεπόπτες μπορεί να χωριστούν σε δύο τύπους: τον τύπο 1 και τον τύπο 2, όπως φαίνεται στην Εικόνα 1.

Ο υπερεπόπτης τύπου 1 είναι λογισμικό που τρέχει απευθείας στο υλικό του υπολογιστικού συστήματος χωρίς την ανάγκη ύπαρξης λειτουργικού συστήματος. Εξαιτίας της δυνατότητας του να επικοινωνεί απευθείας με το υλικό, έχει αυξημένη απόδοση έναντι του τύπου 2. Πέραν όμως της απόδοσής του, θεωρείται ότι διαθέτει μεγαλύτερη ασφάλεια, καθώς τα λειτουργικά συστήματα που φιλοξενεί δεν έχουν την δυνατότητα να τον επηρεάσουν. Μια εικονική μηχανή που σκοπίμως προσπαθεί να προσβάλει είτε τον υπερεπόπτη είτε τις άλλες εικονικές μηχανές δεν είναι σε θέση να το πραγματοποιήσει καθώς έχει περιορισμένα δικαιώματα. Τέλος, ο τύπος 1 έχει λιγότερη επιβάρυνση για το σύστημα καθώς τρέχει απευθείας στο υλικό αφήνοντας έτσι περισσότερους πόρους για τις εικονικές μηχανές

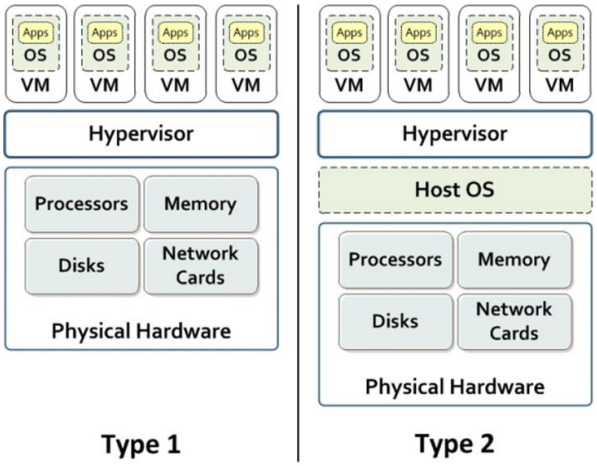
Από την άλλη πλευρά ο υπερεπόπτης τύπου 2 είναι μια εφαρμογή, που εκτελείται μέσα σε ένα παραδοσιακό λειτουργικό σύστημα. Το λειτουργικό σύστημα έχει αναλάβει την διαχείριση των πόρων και ο υπερεπόπτης αναλαμβάνει να διαμοιράσει αυτούς τους πόρους στις εκάστοτε εικονικές μηχανές. Για τον λόγο αυτό είναι ευκολότερος στην ανάπτυξη, καθώς πολλές από τις λειτουργίες διαχείρισης υλικού τις αναλαμβάνει το παραδοσιακό λειτουργικό σύστημα. Επιπρόσθετα επιτρέπει την χρήση οποιουδήποτε υλικού διότι αυτό εξαρτάται μόνο από το λειτουργικό σύστημα που χρησιμοποιείται σαν βάση. Το μοντέλο αυτό θεωρείται λιγότερο ασφαλές, διότι οποιαδήποτε ενέργεια επηρεάσει το λειτουργικό στο οποίο φιλοξενείται ο υπερεπόπτης, μπορεί να έχει αντίκτυπο στον υπερεπόπτη.

Παρακάτω ακολουθεί συγκεντρωτικός πίνακας με τους σημαντικότερους υπερεπόπτες που κυκλοφορούν στην αγορά. Στο πίνακα υπάρχουν πληροφορίες σχετικά με την αρχιτεκτονική

που υποστηρίζουν, τα λειτουργικά συστήματα που υποστηρίζουν καθώς και αν αποτελούν λογισμικό ανοιχτού κώδικα.

Υπερεπόπτης	**Τύπος**	**Ανοιχτού κώδικα**	**Αρχιτεκτονική**	**Υποστηριζόμενα λειτουργικά συστήματα**
XenServer	Τύπος 1	Ναι	IA-32,x86-64,ARM	Linux,BSD,Windows,Solaris,RTOS
Oracle VM Server for x86	Τύπος 1	Ναι	IA-32, x86-64	Windows,Linux,Solaris
Microsoft Hyper-V	Τύπος 1	Όχι	x86-64	Windows
VMware ESX/ESXi	Τύπος 1	Όχι	i386	Linux
KVM	Τύπος 1	Ναι	x86,IA-64,ARM	Linux,BSD,Windows,Solaris,RTOS
VMware Workstation	Τύπος 2	Όχι	x86-64	Linux,BSD,Windows
VMware Player	Τύπος 2	Όχι	x64	Linux,BSD,Windows
VirtualBox	Τύπος 2	Ναι	x86	Linux,Winows,Solaris,BSD
Microsoft Virtual PC	Τύπος 2	Όχι	x86-64	Windows

Πίνακας 1 Υπερεπόπτες

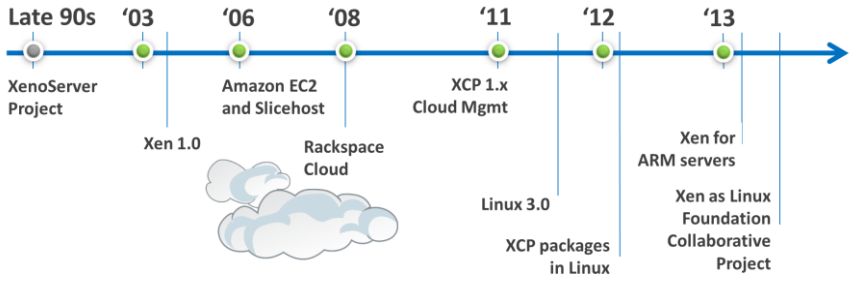


Εικόνα 1 Υπερεπόπτης Τύπου 1 και Τύπου 2

2 Υπερεπόπτης Xen

Τα σύγχρονα υπολογιστικά συστήματα έχουν αρκετή υπολογιστική ισχύ ώστε να υποστηρίξουν εικονικοποίηση. Αυτό έχει οδηγήσει στην ανάπτυξη νέων τεχνικών δημιουργίας εικονικών μηχανών. Σε αυτήν την εργασία θα ασχοληθούμε με τον Xen ο οποίος είναι υπερεπόπτης τύπου ένα.Ο Xen είναι ένα λογισμικό ανοιχτού κώδικα που παρέχει υπηρεσίες εικονικοποίησης.

Ο υπερεπόπτης Xen δημιουργήθηκε στο Πανεπιστήμιο του Cambridge στα τέλη της δεκαετίας του 90 από τον Ian Pratt και το Simon Crosby σαν κομμάτι του Xenoserver.Το 2002 έγινε λογισμικό ανοιχτού κώδικα και τον επόμενο χρόνο κυκλοφόρησε η πρώτη έκδοση του.Το 2004 έγινε η κυκλοφορία του Xen 1.0 και ακολούθησε το Xen 2.0. Την επόμενη χρόνια ο Xen υιοθετήθηκε από μεγάλες εταιρίες όπως η Red Hat,Novell και Sun επιταχύνοντας έτσι την έκδοση του Xen 3.0. Με την ραγδαία ανάπτυξη των κινητών συσκευών η Samsung Electronics το 2008 αποφασίζει την υλοποίηση του Xen ARM ώστε να γίνει εφικτή η εκτέλεση του Xen σε επεξεργαστές ARM. Σημείο κλειδί του συγκεκριμένου εγχειρήματος αποτελεί η χρονιά 2013 καθώς ο Xen ARM κυκλοφορεί επίσημα.



Εικόνα 2 Ιστορική αναδρομή Xen υπερεπόπτη

Ο Xen μπορεί να διαχωριστεί σε τρείς βασικές οντότητες [1]. Αρχικά έχουμε τον υπερεπόπτη που αντικαθιστά το κλασσικό λειτουργικό σύστημα και είναι υπεύθυνος για την διαχείριση και την κατανομή του υλικού. Στην συνέχεια έχουμε το Domain0 που συχνά το συναντούμε στην βιβλιογραφία και ως Dom0. Το Dom0 το ενεργοποιεί ο υπερεπόπτης και υποστηρίζει τα περισσότερα λειτουργικά συστήματα με εξαίρεση τα Windows.Τέλο,ς έχουμε το DomainU το οποία περιέχει το εικονικό μηχάνημα που διαχειρίζεται ο Dom0. Στην βιβλιογραφία το DomainU το συναντάμε ως DomU.

Σε αυτό το επίπεδο υπάρχει η δυνατότητα υποστήριξης όλων των λειτουργικών συστημάτων. Ο Xen υποστηρίζει δύο τεχνικές εικονικοποίησης, παραεικονικοποίηση και HVM καθώς και συνδυασμό των δύο τεχνικών για την δημιουργία υβριδίων(HVM with Paravirtualization drivers, PVHVM,PV in an HVM container) .

Στην τεχνική της παραεικονικοποίησης το λειτουργικό σύστημα γνωρίζει ότι εκτελείται μέσω υπερεπόπτη και για αυτό τον λόγο πρέπει να είναι κατάλληλα διαμορφωμένο. Ως εκ τούτου, υπάρχει περιορισμένος αριθμός λειτουργικών συστημάτων που μπορούν να υποστηριχθούν σε αυτήν την λειτουργία.

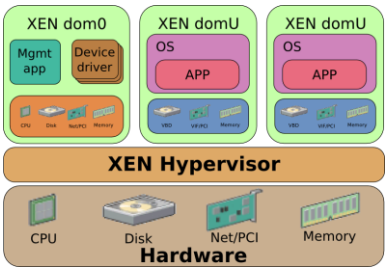
Με το HVM το λειτουργικό σύστημα που εκτελείται σαν εικονική μηχανή δεν γνωρίζει ότι βρίσκεται σε περιβάλλον που διαχειρίζεται από υπερεπόπτη. Για να γίνει εφικτό αυτό απαιτείται ειδικό υλικό που να υποστηρίζει αυτήν την λειτουργία.

2.1 Αρχιτεκτονική του Xen

Η βασική αρχή που διέπει τον Xen είναι αυτή του δακτύλιου προστασίας. Σε αυτήν την αρχιτεκτονική έχουμε διαστρωμάτωση των δικαιωμάτων του συστήματος. Όσο προχωρούμε

στα εξωτερικά στρώματα τόσο μειώνονται τα δικαιώματα. Βάση των παραπάνω ο υπερεπόπτης έχει τα περισσότερα δικαιώματα με τον Dom0 να ακολουθεί και τον DomU να είναι αυτός με τα λιγότερα. Με αυτό τον τρόπο ο Xen επιτυγχάνει την απομόνωση του εκάστοτε επιπέδου και διασφαλίζει την ακεραιότητα και την ασφάλεια των δεδομένων του συστήματος.

Η επικοινωνία των φιλοξενούμενων εικονικών μηχανών με τις συσκευές του συστήματος επιτυγχάνεται δια μέσου του Dom0.Όταν έχουμε κλήση από ένα DomU για χρήση μια φυσικής συσκευής, τότε επικοινωνεί το DomU με το Dom0 και στην συνέχεια το Dom0 με τον υπερεπόπτη ώστε να επιτευχθεί η πρόσβαση στην συσκευή. Στην Εικόνα 3 βλέπουμε την βασική δομή του Xen.



Εικόνα 3 Xen υπερεπόπτης

2.2 Δαίμονες και βιβλιοθήκες του Xen

Ο Xen παρέχει μια σειρά από δαίμονες Linux και βιβλιοθήκες για την διαχείριση των εικονικών μηχανών. Οι υπηρεσίες αυτές υπάρχουν στο Dom0 και υποστηρίζουν τον έλεγχο και την διαχείριση του συστήματος. Τα εργαλεία που περιέχει είναι τα xl, Xendm, Xenstore, Libxenctrl, Stub,Xen Virtual Firmware.

Το xl είναι ένα εργαλείο που ήρθε για να αντικαταστήσει το xm. Είναι εργαλείο γραμμής εντολών και έχει σαν σκοπό την μεταβίβαση των αιτημάτων του χρήστη στο Xend.

Ο Xend είναι μια εφαρμογή σε python και αποτελεί τον διαχειριστή για το περιβάλλον Xen.Με την βοήθεια της βιβλιοθήκης libxenctrl δημιουργεί αιτήματα στον υπερεπόπτη.

Η libxenctrl πρόκειται για μια βιβλιοθήκη σε C που παρέχει την δυνατότητα στον δαίμονα xend να επικοινωνεί με τον υπερεπόπτη μέσω του dom0.

Ο δαίμονας Xenstored είναι ένα σύστημα αποθήκευσης, το οποίο διαμοιράζεται μεταξύ των εικονικών μηχανήματών. Είναι ένα ιεραρχικό σύστημα αποθήκευσης και εκτελείται στον Dom0.

Για να μπορέσει να γίνει σωστή εκκίνηση των εικονικών μηχανημάτων απαιτείται η χρήση ενός εικονικού BIOS.Για τον λόγο αυτό ο Xen περιέχει το Xen Virtual Firmware το οποίο διασφαλίζει ότι θα παρασχεθούν τα κατάλληλα σήματα ώστε να γίνει επιτυχής εκκίνηση του εικονικού μηχανήματος.

Τέλος έχουμε το δαίμονα Stub ο οποίος είναι απαραίτητος στην τεχνική εικονικοποίησης HVM.Το Stub είναι υπεύθυνο για την διευθέτηση των αιτημάτων I/O.

2.3 Διαχείριση μνήμης

Για να μπορέσει μια εφαρμογή να εκτελεστεί απαιτεί να έχει τον δικό της χώρο στην μνήμη. Σε συστήματα που δεν υπάρχει εικονικοποίηση το λειτουργικό σύστημα δεσμεύει συνεχόμενο χώρο της φυσικής μνήμης του συστήματος για να εκτελεστεί η διεργασία. Κάτι παρόμοιο εκτελεί ο Xen για τα λειτουργικά συστήματα που τρέχουν σε αυτόν. Για να το πετύχει αυτό εισάγει ένα

μηχανισμό στον οποίο έχει πρόσβαση ο πυρήνας του κάθε DomU και ονομάζεται ψευδοφυσική μνήμη.

Κάθε διεργασία που εκτελείται σε εικονικό περιβάλλον έχει πρόσβαση στην εικονική μνήμη. Στην συνέχεια αναλαμβάνει ο πυρήνας του λειτουργικού συστήματος που έχει πρόσβαση στην ψευδοφυσική μνήμη και αντιστοιχεί τις θέσεις της εικονικής μνήμης με αυτές της ψευδοφυσικής. Τέλος ο υπερεπόπτης αναλαμβάνει να αντιστοιχήσει τις θέσεις της ψευδοφυσικής μνήμης του κάθε λειτουργικού συστήματος με θέσεις μνήμης στο φυσικό επίπεδο.

Θα ήταν δυνατό να μην υπάρχει αυτό το τριπλό επίπεδο στην προσπέλαση της μνήμης και να υπάρξει χρήση ενός διπλού επιπέδου με μηχανισμούς που απέτρεπαν την προσπέλαση στο χώρο μνήμης των άλλων πυρήνων. Όμως αυτή η τεχνική έχει απορριφθεί για δυο βασικούς λόγους.

Αρχικά τα λειτουργικά συστήματα λειτουργούν κάτω από την υπόθεση ότι κατέχουν συνεχόμενες θέσεις μνήμης στην φυσική μνήμη του συστήματος. Η φυσική μνήμη όμως υποστηρίζει κενά ανάμεσα στις διευθύνσεις έτσι ώστε να παραβλέπει τα προβληματικά τμήματα της. Για αυτό τον λόγο μια τέτοια προσέγγιση θα απαιτούσε μεγάλες αλλαγές στον πυρήνα του λειτουργικού και επιπρόσθετα θα εισήγαγε καθυστέρηση στην προσπέλαση.

Ο δεύτερος και σημαντικότερος λόγος που έχει υιοθετηθεί η τεχνολογία των στρωμάτων είναι η αποτελεσματικότερη διαχείριση της μνήμης του συστήματος. Σε πολλές περιπτώσεις ο κύκλος ζωής μια εικονικής μηχανής είναι περιορισμένος. Είναι συχνό φαινόμενο μια εικονική μηχανή να καταστέλλει την λειτουργία της μετά από μικρό χρονικό διάστημα. Όμως κατά την συνέχιση της λειτουργίας της δεν είναι βέβαιο ότι το κομμάτι της φυσικής μνήμης που καταλάμβανε είναι πλέον διαθέσιμο. Επίσης πρέπει να προστεθεί ότι πολλές φορές υπάρχει η ανάγκη μεταφοράς μιας εικονικής μηχανής σε διαφορετικό φυσικό υπολογιστή. Σε αυτήν την περίπτωση είναι σχεδόν βέβαιο ότι οι διευθύνσεις μνήμης που είχε αποθηκευμένες δεν θα είναι πλέον διαθέσιμες.

2.4 Διαχείριση επεξεργαστή

Ο Xen είναι από τα λίγα περιβάλλοντα που επιτρέπουν στον χρήστη να επιλέξει ανάμεσα σε τρείς διαφορετικούς τύπους διαχείρισης του επεξεργαστή. Οι τρεις αυτές τεχνικές είναι οι παρακάτω :
SEDF
BVT
Credit Sheduler

Ο SEDF κάνει χρήση αλγορίθμων που εκτελούνται σε πραγματικό χρόνο. Κάθε εικονική μηχανή καθορίζει τις ανάγκες της σε επεξεργαστική ισχύ μέσα από ένα διάνυσμα (s_i,p_i,x_i) με τις μεταβλητές s_i p_i να καθορίζουν την επεξεργαστική ισχύ και τον χρόνο εκτέλεσης αντίστοιχα. Η μεταβλητή x_i καθορίζει αν η εικονική μηχανή έχει την δυνατότητα να λάβει παραπάνω χρόνο στον επεξεργαστή. Η τεχνική SEDF διαμοιράζει δίκαια τον χρόνο σε κάθε εικονική μηχανή. Επίσης για κάθε εικονική μηχανή ο SEDF κρατάει ένα διάνυσμα (d_i,r_i). Το d_i(προθεσμία) είναι ο χρόνος κατά τον οποίο ολοκληρώνεται η περίοδος χρήσης του επεξεργαστή από την εκάστοτε εικονική μηχανή. Ο αλγόριθμος επιλέγει πάντα την εικονική μηχανή με το συντομότερο d_i.Το r_i(εναπομείναν) δηλώνει τον χρόνο που απομένει σε μια εικονική μηχανή ενώ εκτελείται στον επεξεργαστή.

Ο BVT είναι ένας μηχανισμός που σχεδιάστηκε με την φιλοσοφία του δίκαιου διαμοιρασμού του επεξεργαστή. Ο αλγόριθμος δίνει προτεραιότητα στις εικονικές μηχανές με μικρότερο χρόνο εκτέλεσης. Επιπρόσθετα δίνει την δυνατότητα σε εφαρμογές που είναι πραγματικού χρόνου να πάρουν προτεραιότητα ακόμα και αν ο χρόνο εκτέλεσης τους είναι μεγαλύτερος. Με αυτό τον τρόπο παρέχει χαμηλή καθυστέρηση και για αυτό τον λόγο μπορεί να υποστηρίξει εφαρμογές πραγματικού χρόνου .

Ακόμα υπάρχει ο Credit Sheduler που είναι ο νεότερος μηχανισμός διαχείρισης του επεξεργαστή. Κάθε εικονική μηχανή έχει ένα ειδικό βάρος και μια μεταβλητή c_i. Η μεταβλητή c_i δηλώνει το ποσοστό του επεξεργαστή που μπορεί να κατέχει μια εφαρμογή. Όταν η μεταβλητή γίνει μηδέν τότε μπορεί να κατέχει όλο τον επεξεργαστή κατά τον χρόνο εκτέλεσης της. Κάθε εικονική μηχανή λαμβάνει 30ms ωφέλιμου χρόνου εκτέλεσης πριν αντικατασταθεί. Ο μηχανισμός ελέγχει κάθε 10 ms την χρήση των πόρων και ανανεώνει το ειδικό βάρος κάθε εφαρμογής.

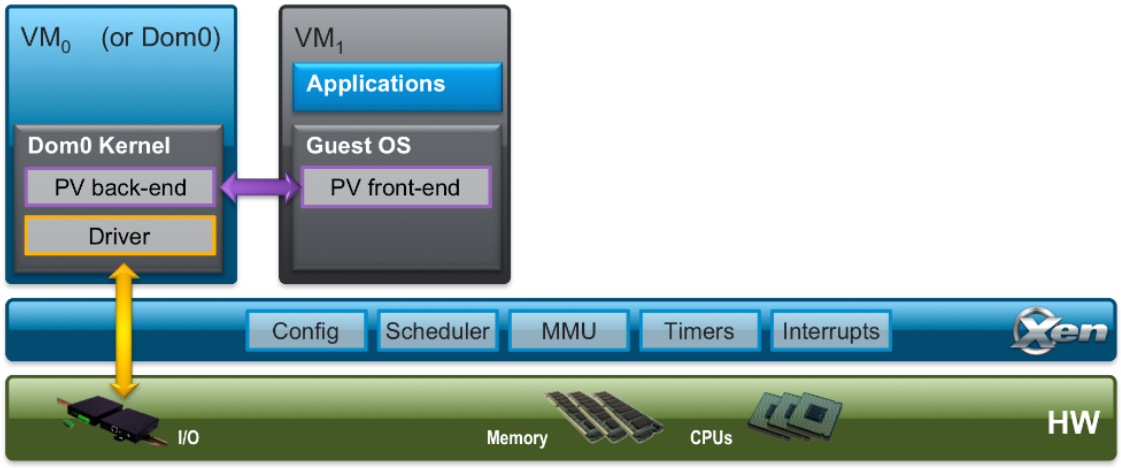
Πέραν των μηχανισμών που αναλύθηκαν παραπάνω, στα εικονικά περιβάλλοντα υπάρχει ιεραρχία δικαιωμάτων κάθε συστατικού μέρους. Για την επίτευξη αυτής της ιεραρχίας έχουμε χρήση της τεχνικής των δακτυλίων. Ο υπερεπόπτης εκτελείται με τα μέγιστα δικαιώματα και για τον λόγο αυτό βρίσκεται σε επίπεδο δακτυλίου 0.Στο επίπεδο 1 βρίσκεται ο πυρήνας του λειτουργικού συστήματος και στους εξωτερικούς δακτυλίους οι εφαρμογές. Ο αριθμός των εξωτερικών δακτυλίων διαφέρει από αρχιτεκτονική σε αρχιτεκτονική.

2.5 Διαχείριση I/O

Σε αυτό το κεφάλαιο ασχολούμαστε με την διαχείριση των I/O. Η διαχείριση των I/O διαφέρει από HVM σε παραεικονικοποίηση. Αρχικά θα γίνει ανάλυση της τεχνικής σε παραεικονικοποίηση και στην συνέχεια θα αναλυθεί σε HVM.

Όπως αναφέρθηκε και παραπάνω, ο Dom0 έχει άμεση πρόσβαση στο υλικό του συστήματος και για τον λόγο αυτό διαθέτει τους απαραίτητους οδηγούς για την πραγματοποίηση της προσπέλασης του υλικού. Σε αντίθεση με τον Dom0, ο DomU δεν έχει απευθείας πρόσβαση στο υλικό, για τον λόγο αυτό απαιτούνται ειδικοί οδηγοί.

Ένας ειδικός οδηγός αποτελείται από δύο μέρη το frontend και το backend.Ο DomU κατέχει το frontend ενώ ο Dom0 το backend. Στην Εικόνα 4 παρατηρούμε το τυπικό σχήμα αυτού του μοντέλου. Αυτό το μοντέλο παρέχει υψηλή απόδοση καθώς ο υπερεπόπτης και το λειτουργικό σύστημα δουλεύουν πιο αποδοτικά διότι δεν χρειάζεται η δημιουργία εικονικών συσκευών. Με αυτό τον τρόπο υπάρχει απλούστερη πρόσβαση στις συσκευές.



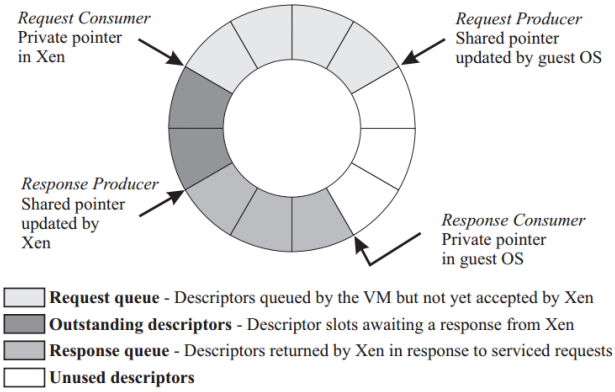
Εικόνα 4 Εικονικό I/O

Τα δεδομένα μεταφέρονται με χρήση ασύγχρονων I/O δακτυλίων όπως φαίνεται στην εικόνα 5.Ο δακτύλιος είναι μια κυκλική ουρά αναμονής. Η προσπέλαση του κάθε δακτυλίου γίνεται από ένα ζεύγος δεικτών παραγωγού-καταναλωτή. Η εικονική μηχανή τοποθετεί μια αίτηση για δεδομένα στον δακτύλιο με την χρήση ενός δείκτη παραγωγού. Ο Xen αφαιρεί αυτές τις αιτήσεις από τον δακτύλιο τοποθετώντας έναν δείκτη καταναλωτή. Σε αυτό το μοντέλο δεν υπάρχει ανάγκη οι αιτήσεις να εκπληρώνονται με την σειρά που καταφτάνουν. Για τον λόγο αυτό οι

εικονικές μηχανές τοποθετούν έναν μοναδικό αναγνωριστικό με κάθε αίτηση. Από την πλευρά του ο Xen μπορεί να αλλάξει την προτεραιότητα κάθε αιτήματος ανάλογα με τις ανάγκες του συστήματος.

Η παρουσία ενός υπερεπόπτη σημαίνει ότι υπάρχει ένα πρόσθετο επίπεδο προστασίας ανάμεσα στα εκάστοτε Dom και τις συσκευές I/O.Για τον λόγο αυτό είναι σημαντική η ύπαρξη ενός μηχανισμού μεταφοράς δεδομένων που επιτρέπει την κάθετη μεταφορά με όσο το δυνατόν μικρότερη επιβάρυνση. Δυο βασικοί παράγοντες διαμόρφωσαν τον σχεδιασμό του μηχανισμού I/O:

- Διαχείριση πόρων
- Διαχείριση συμβάντων



Εικόνα 5 Δακτύλιος εισόδου-εξόδου δεδομένων

Όταν οι εικονικές μηχανές εκτελούνται σε HVM τότε γίνεται χρήση του QEMU. Ο QEMU είναι ένα λογισμικό ανοιχτού κώδικα. Πρόκειται για έναν υπερεπόπτη που μπορεί να εκτελέσει εικονικοποίηση. Αν και έχει την δυνατότητα να εκτελέσει από μόνος του εικονικοποίηση συχνά συναντάται ενσωματωμένος σε άλλους υπερεπόπτες όπως ο KVM, VMware,Xen.

Ο QEMU τρέχει στον Dom0 και αναλαμβάνει την δημιουργία εικονικών I/O. Όταν εκτελείται μια δημιουργία εικονικού I/O ο υπερεπόπτης του Xen αναλαμβάνει να σταματήσει τις διεργασίες που εκτελούνται στους εικονικούς επεξεργαστές ώστε να ολοκληρώσει την διαδικασία δημιουργίας ο QEMU.

Πέραν όμως από την δημιουργία εικονικών I/O ο QEMU έχει την δυνατότητα να εξομοιώσει τις εντολές του ARMv7. Με αυτόν τον τρόπο υπάρχει η δυνατότητα εκτέλεσης Xen on ARM ακόμα και από επεξεργαστές που δεν είναι της οικογένειας της ARM.

2.6 Κανάλια γεγονότων

Τα κανάλια γεγονότων είναι ένας ασύγχρονος τρόπος επικοινωνίας στον Xen. Η χρήση τους διευκολύνει την επικοινωνία του frontend με το backend κομμάτι του οδηγού του υλικού για τις εικονικές μηχανές.

Η πρώτη αρχιτεκτονική που γίνεται χρήση είναι ο μηχανισμός γεγονότων(event channel). Η συγκεκριμένη τεχνική χρησιμοποιείται ώστε να καταστεί δυνατή η επικοινωνία ανάμεσα είτε του υπερεπόπτη με τις εικονικές μηχανές είτε των εικονικών μηχανών αναμεταξύ τους. Ακολουθεί την ίδια αρχιτεκτονική που συναντάται στην παραδοσιακή σηματοδοσία των λειτουργικών UNIX. Ως εκ τούτου πρόκειται για σήματα του ενός bit που ενημερώνουν για το πέρας μιας

διαδικασίας. Μια βασική διαφορά με την σηματοδοσία σε UNIX είναι ότι δεν χάνονται τα σήματα που δημιουργούνται κατά τον χρόνο που ο μηχανισμός παράδοσης είναι απενεργοποιημένος.

Σε μεγάλο βαθμό αντικαθιστούν τις διακοπές σε επίπεδο υλικού(hardware interrupts). Μια διακοπή είναι ένας ασύγχρονος τρόπος επικοινωνίας που ενημερώνει το υλικό σχετικά με τα γεγονότα που συμβαίνουν.

Επιπρόσθετα εκτός από τα κανάλια γεγονότων ο Xen προσφέρει και ένα ακόμα ασύγχρονο πρωτόκολλο επικοινωνίας την παγίδα (trap).Πρόκειται για ένα μηχανισμό που εκτελείται σε επίπεδο υλικού και σε αντίθεση με τα κανάλια γεγονότων είναι στατικός.

Όταν μια εικονική μηχανή εκτελείται σε μια κεντρική μονάδα επεξεργασίας, ο υπερεπόπτης αναλαμβάνει την ενημέρωση αυτής της μονάδας με έναν πίνακα που περιέχει πληροφορίες σχετικές με τις διακοπές. Με αυτό τον τρόπο μεταθέτει την διαχείριση των διακοπών στο εκάστοτε εικονικό μηχάνημα.

Ο βασικός λόγος που μια εικονική μηχανή κάνει χρήση αυτού του μηχανισμού είναι η αδυναμία της να εκτελέσει άμεσα κλήσεις συστήματος. Αυτό συμβαίνει διότι οι κλήσεις συστήματος απαιτούν δικαιώματα που είναι δεσμευμένα για την εκτέλεση του Xen.

Μετά τον τερματισμό τις εκάστοτε εικονικής μηχανής ο πίνακας που περιέχει τις σχετικές πληροφορίες με τις διακοπές διαγράφεται. Για αυτό τον λόγο χρησιμοποιούνται μόνο για διακοπές που μπορούν να συμβούν σε χρόνο εκτέλεσης. Ο πίνακας που περιγράφηκε παραπάνω έχει την δομή της Εικόνας 6.

```
struct trap_info {
    uint8_t        vector;  /* exception vector
                               */
    uint8_t        flags;   /* 0-3: privilege level; 4: clear
       event enable? */
    uint16_t       cs;      /* code selector
                               */
    unsigned long address; /* code offset
                               */
};
```

Εικόνα 6 Πίνακας παγίδων

Κάθε καταχώρηση περιέχει τον αναγνωριστικό αριθμό της παγίδας, το επίπεδο δικαιωμάτων που χρειάζεται ώστε να ενεργοποιηθεί καθώς και την διεύθυνση του μηχανισμού διευθέτησης της εκάστοτε παγίδας.

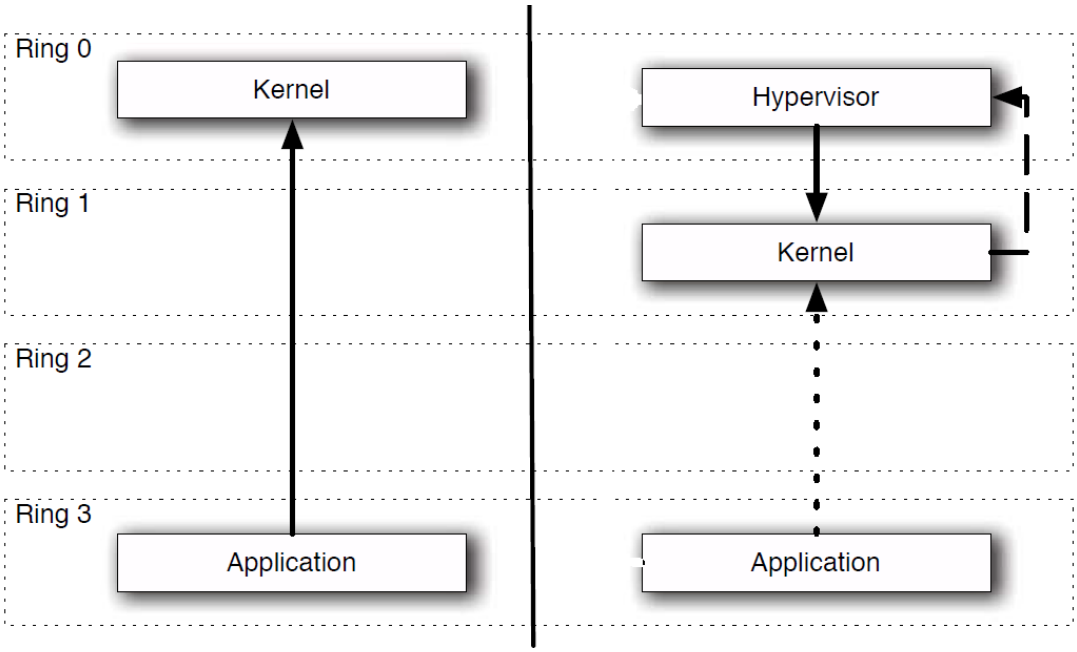
2.7 Υπερκλήσεις

Πρόκειται για έναν μηχανισμό παρόμοιο με τις κλήσεις συστήματος στα παραδοσιακά λειτουργικά συστήματα. Εξαιτίας της ιεραρχίας δικαιωμάτων οι εικονικές μηχανές εκτελούνται με μειωμένα δικαιώματα στην κεντρική μονάδα επεξεργασίας. Για αυτό τον λόγο δεν είναι σε θέση να έχουν πρόσβαση σε εντολές που απαιτούν αυξημένα δικαιώματα.

Στην εικόνα 7 βλέπουμε την διαφορά ανάμεσα στο σχήμα που ακολουθεί ο Xen και το σχήμα που ακολουθούν τα παραδοσιακά λειτουργικά συστήματα. Στο παραδοσιακό σχήμα αυτός που αναλαμβάνει την διεκπεραίωση των αιτήσεων είναι ο πυρήνας του λειτουργικού συστήματος. Αυτό συμβαίνει διότι ο πυρήνας του λειτουργικού κατέχει μέγιστα δικαιώματα στην κεντρική μονάδα επεξεργασίας.

Όμως αυτό το σχήμα διαφέρει στον Xen, διότι όπως αναφέρθηκε στο κεφάλαιο για την διαχείριση της κεντρικής μονάδας επεξεργασίας ο υπερεπόπτης τρέχει σε επίπεδο 0. Ως εκ τούτου αν μια εφαρμογή χρειαστεί αυξημένα δικαιώματα, εκτελεί στον πυρήνα του λειτουργικού συστήματος μια κλήση συστήματος και αυτός με την σειρά του εκτελεί μια υπερκλήση στον υπερεπόπτη.

Αυτό το επιπλέον επίπεδο εισάγει καθυστέρηση όμως διασφαλίζει την ομαλή λειτουργία του συστήματος. Με το σχήμα αυτό οι εφαρμογές δεν μπορούν να εκτελέσουν ενέργειες στην κεντρική μονάδα επεξεργασίας ώστε να επηρεάσουν τον υπερεπόπτη. Ο Xen παρέχει και την δυνατότητα για απευθείας κλήσης συστήματος από το επίπεδο εφαρμογής, όμως κάτι τέτοιο απαιτεί την χρήση τροποποιημένης βιβλιοθήκες libc το οποίο βρίσκεται ακόμα σε ανάπτυξη.



Εικόνα 7 υπερκλήση

2.8 Διαφορές ανάμεσα σε Dom0 και DomU

Όπως αναφέρθηκε παραπάνω ο Dom0 είναι μια εικονική μηχανή με ειδικά δικαιώματα. Χωρίς την ύπαρξη του Dom0 ο υπερεπόπτης δεν θα μπορούσε να χρησιμοποιηθεί καθώς ο Dom0 αποτελεί το βασικό λειτουργικό σύστημα που έχει σαν στόχο την διαχείριση των επιμέρους εικονικών μηχανών.

Αυτό έχει σαν αποτέλεσμα να έχει ειδικά δικαιώματα όπως η απευθείας προσπέλαση του υλικού. Ο Dom0 έχει οδηγούς για το υλικό όπως η κάρτα δικτύου και ο δίσκος και παρέχει εικονικούς οδηγούς για τα υπόλοιπα εικονικά μηχανήματα.

Όσο αφορά την διαχείριση της μνήμης και του επεξεργαστή του συστήματος η απόδοση των DomU πλησιάζει αυτήν του Dom0. Σε μεγάλο βαθμό αυτό έχει επιτευχθεί λόγω των νέων τεχνικών διαχείρισης που αναλύθηκαν σε προηγούμενο κεφάλαιο.

Παρά την ανάπτυξη των νέων αλγορίθμων διαχείρισης υλικού υπάρχουν ακόμα δυσκολίες στην διαχείριση των I/O.Για αυτό τον λόγο η ανάπτυξη εφαρμογών πραγματικού χρόνου συναντά προβλήματα.

2.9 Δυνατότητες Xen

Ο Xen έχει πολλές δυνατότητες και υποστηρίζει μεγάλο αριθμό επεξεργαστών και μνήμης. Επίσης έχει την δυνατότητα να υποστηρίξει διάφορα λειτουργικά συστήματα. Τέλος, υπάρχει η δυνατότητα εύκολης μεταφοράς των εικονικών μηχανών.

Όσο αφορά τους επεξεργαστές μπορεί να υποστηρίξει έως 4095 κεντρικές μονάδες επεξεργασίας. Κάθε μηχάνημα που κάνει χρήση παραεικονικοποίησης μπορεί να υποστηρίξει μέχρι 512 εικονικούς επεξεργαστές, ενώ κάθε HVM μέχρι 256 επεξεργαστές. Οι οικογένειες επεξεργαστών που υποστηρίζει είναι οι παρακάτω:

- Intel :IA-32,IA-64,x86-64
- PowerPC
- ARM
- MIPS XLP832

Το συνολικό μέγεθος της μνήμης που υποστηρίζει είναι 15TB. Κάθε εικονικό μηχάνημα έχει την δυνατότητα να υποστηρίξει 1TB μνήμη αν τρέχει σε HVM και 512 GB αν κάνει χρήση παραεικονικοποίησης.

Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο ο Xen έχει την δυνατότητα να εκτελέσει διάφορα λειτουργικά συστήματα. Παρακάτω ακολουθεί συγκεντρωτική λίστα με αυτά.

Dom0 λειτουργικά συστήματα:

- Alpine Linux
- Debian
- FreeBSD
- Gentoo και Arch Linux
- Mageia
- NetBSD
- OpenSolaris
- OpenSUSE
- Qebes OS
- SUSE Linux Entrepise Server
- Solaris
- Ubuntu

DomU λειτουργικά συστήματα:

- FreeBSD
- GNU/Hurd
- Linux
- MINIX
- NetBSD
- NetWare
- OpenSolaris

- OZONE
- Plan 9
- OpenBSD
- Windows

Τέλος, ο Xen υποστηρίζει την δυνατότητα μεταφοράς των εικονικών μηχανών ανάμεσα σε δυο φυσικά υπολογιστικά συστήματα που βρίσκονται στο ίδιο δίκτυο. Κατά την διάρκεια την εκτέλεσης της μεταφοράς ο Xen αντιγράφει την μνήμη της εικονικής μηχανής στο προορισμό χωρίς όμως να σταματάει την εκτελέσει της εικονικής μηχανής. Μετά το πέρας της διαδικασίας της αντιγραφεί απαιτούνται 60-300 ms ώστε να γίνει συγχρονισμός των δυο μηχανών. Με αυτό τον τρόπο δημιουργεί την ψευδαίσθηση ότι εκτελεί μετεγκατάσταση της εικονικής μηχανής χωρίς την ανάγκη διακοπής των υπηρεσιών της εικονικής μηχανής.

2.10 Απόδοση Xen σε x86

Σε αυτό το κεφάλαιο παρουσιάζεται μια πειραματική μελέτη [7] της απόδοσης του Xen σε περιβάλλον x86. Για τα πειράματα χρησιμοποιήθηκε το μετροπρόγραμμα lmbench και λειτουργικό σύστημα Linux.Το υλικό του συστήματος αποτελείται από έναν dual processor 2.4Ghz Xeon με 2GB μνήμη και 146GB Hitachi DK32EJ 10k RPM δίσκο.

Στόχος των πειραμάτων είναι η μελέτη της καθυστέρησης που εισάγει το Xen στο σύστημα. Στους πίνακες που ακολουθούν παρουσιάζονται τα αποτελέσματα της μελέτης. Στην πρώτη γραμμή των πινάκων αναφέρεται η διεργασία που εκτελείται ενώ στην δεύτερη και τρίτη γραμμή ακολουθούν οι χρόνοι εκτέλεσης σε Linux και σε Linux με χρήση Xen, αντίστοιχα. Στον πρώτο πίνακα υπάρχουν οι χρόνοι εκτέλεσης διαφόρων κλήσεων συστήματος, ενώ στον δεύτερο οι χρόνοι για εναλλαγή περιεχομένου (context switching). Η μονάδα μέτρησης και στις δυο περιπτώσεις είναι το μs.

Στον Πίνακα 2 παρατηρούμε ότι η χρήση του Xen επηρεάζει μόνο συγκεκριμένες κλήσεις συστήματος. Τα αποτελέσματα αυτά είναι αναμενόμενα, καθώς για την εκτέλεση αυτών των διεργασιών απαιτείται ανανέωση μεγάλου αριθμού πινάκων σελίδων, που με την σειρά τους πρέπει να ελεγχθούν από τον υπερεπόπτη. Αυτή η διαδικασία έχει σαν αποτέλεσμα την εισαγωγή καθυστέρησης στο σύστημα.

Config	Null call	Null I/O	Stat	Open close slct	Tcp	Sig inst	Sig hndl	Fork proc	Exec proc	Sh proc
L-UP	0.45	0.50	1.28	1.92	5.70	0.68	2.49	110	530	4k0
Xen	0.46	0.50	1.22	1.88	5.69	0.69	1.75	198	768	4k8

Πίνακας 2 Χρόνοι εκτέλεσης για το μετροπρόγραμμα lmbench

Στον Πίνακα 3 παρατηρούμε ότι το Xen εισάγει καθυστέρηση στο σύστημα η οποία κυμαίνεται από 1μs έως 3μs. Η καθυστέρηση αυτή προκύπτει από την ανάγκη εκτέλεσης υπερκλήσεων για την διευθέτηση του context switching. Παρατηρούμε ότι σε γενικές γραμμές η απόδοση του Xen είναι αρκετά καλή και πλησιάζει σε μεγάλο βαθμό την απόδοση του συστήματος χωρίς χρήση Xen. Για αυτό τον λόγο ο Xen αποτελεί μια εξαιρετική επιλογή για εικονικοποίηση.

Config	2p 0K	2p 16K	2p 64K	8p 16K	8p 64K	16p 16K	16p 64K
L-UP	0.77	0.91	1.06	1.03	24.3	3.61	37.6
Xen	1.98	2.22	2.67	3.07	28.7	7.08	39.4

Πίνακας 3 Χρόνοι context switching για το μετροπρόγραμμα lmbench

3 Xen σε ARM

Εξαιτίας των περιορισμών που έθεταν οι παλαιότερες αρχιτεκτονικές της ARM, η τεχνολογία της εικονικοποίησης ήταν αρκετά αργή. Μέχρι το 2013 ο Xen δεν υποστήριζε επίσημα τους επεξεργαστές της ARM. Αυτό ήρθε να αλλάξει με την κυκλοφορία των επεξεργαστών της γενιάς ARMv7. Τόσο η γενιά ARMv7 όσο και η ARMv8 ενσωματώνουν μηχανισμούς σε επίπεδο υλικού για την υποστήριξη της εικονικοποίησης. Η προσέγγιση αυτής της τεχνολογίας από τον Xen έχει γίνει με την δημιουργία της υβριδικής αρχιτεκτονικής εικονικοποίησης, της PVHVM(paravirtualization-on-HVM drivers). Το συγκεκριμένο μοντέλο συνδυάζει τα θετικά των δυο αρχιτεκτονικών εικονικοποίησης. Πιο συγκεκριμένα ο δίσκος, το δίκτυο και οι λοιπές συσκευές δουλεύουν σε παραεικονικοποίηση ενώ ο επεξεργαστής και η μνήμη σε XVM.

3.1 Αρχιτεκτονική Xen σε ARM

Η νέα αυτή αρχιτεκτονική επιλύει τα περισσότερα από τα προβλήματα που είχαν δημιουργηθεί από την μακρόχρονη ανάπτυξη της τεχνολογίας x86 Xen[2]. Αρχικά έκανε επίλυση των προβλημάτων επικοινωνίας των εικονικών μηχανών με το I/O. Παρά την μακρόχρονη ανάπτυξη του μηχανισμού QEMU τα προβλήματα τόσο στην απόδοση όσο και στην ασφάλεια του παραμένουν. Επιπρόσθετα είναι ένας μηχανισμός που καταλαμβάνει μεγάλο αποθηκευτικό χώρο στο σύστημα καθώς αποτελείται από χιλιάδες γραμμές κώδικα. Στην προσπάθεια για επίλυση αυτού του προβλήματος ο Xen on ARM υιοθέτησε την φιλοσοφία του μικρότερου, απλούστερου, καλύτερου. Στον Πίνακα 4 βλέπουμε την σημαντική μείωση σε γραμμές κώδικα της νέας αυτής προσέγγισης.

	Common	ARMv7	ARMv8	Total
Xen/arch/arm	11,767	3,503	1,812	17,082
C	11,587	954	813	13,354
ASM	180	2549	999	13,354
Xen/include/asm-arm	4786	984	1,050	3,728
Total ARM	165,53	4,487	2,862	23,902

X86_64	Total
Xen/arch/x86	124,615
Xen/include/asm-x86	18,530
Total x86_64	143,148

Πίνακας 4 Μέγεθος XEN

Με αυτόν τον τρόπο ο υπερεπόπτης Xen απώλεσε την ανάγκη για χρήση QEMU εκμεταλλευόμενος τους μηχανισμούς εικονικοποίησης που παρέχει η νέα γενιά επεξεργαστών της ARM. Αυτό είχε σαν αποτέλεσμα την δημιουργία μιας ταχύτερης και ασφαλέστερης πλατφόρμας για I/O.

Μια ακόμα σημαντική αλλαγή αυτής της τεχνολογίας είναι η υιοθέτηση μιας και μόνο τεχνικής εικονικοποίησης. Στον Xen μέχρι σήμερα είχαμε την δυνατότητα να επιλέξουμε ανάμεσα σε PV για λειτουργικά ανοιχτού κώδικα και HVM για λειτουργικά κλειστού κώδικα. Ο Xen on ARM θεώρησε αυτό το μοντέλο πολύπλοκο και μη αποδοτικό. Για αυτό προχώρησε στην δημιουργία του PVHVM το οποίο αποτελεί υβρίδιο των δυο τεχνικών.

Ο Xen on ARM παρέχει τρία επίπεδα εκτέλεσης EL0,EL1,EL2.Ο Xen τρέχει εξολοκλήρου σε επίπεδο μηδέν και αφήνει τα άλλα επίπεδα για τις εικονικές μηχανές. Οι εικονικές μηχανές με την σειρά τους τρέχουν το λειτουργικό τους σύστημα σε επίπεδο ένα και τις εφαρμογές τους σε επίπεδο δύο. Με αυτό τον τρόπο μειώνει σημαντικά το context switch. Επιπρόσθετα με την εισαγωγή της εντολής HVC επιτυγχάνεται η αποδοτικότερη επικοινωνία του πυρήνα με τον υπερεπόπτη.

3.2 Διαχείριση υλικού

Ο Xen δημιουργεί ένα δέντρο με τις συσκευές που υπάρχουν στο σύστημα. Ο υπερεπόπτης αναλαμβάνει να αναθέσει όλες τις συσκευές που δεν χρησιμοποιεί, στον Dom0 και έτσι δημιουργεί ένα δέντρο που περιγράφει ακριβώς το περιβάλλον που εκτελούνται οι εικονικές μηχανές.

Το δέντρο με τις συσκευές περιλαμβάνει τον αριθμό των εικονικών επεξεργαστών που έχουν δημιουργηθεί, ο οποίος σε κάποιες περιπτώσεις είναι μικρότερος από τον αριθμό των φυσικών επεξεργαστών. Επίσης περιέχει το μέγεθος της μνήμης που παρέχεται στις εικονικές μηχανές, το οποίο είναι μικρότερο από το μέγεθος της φυσικής μνήμης. Ακόμα περιέχει όλες τις συσκευές που ο υπερεπόπτης δεν κάνει χρήση . Τέλος υπάρχει ένας κόμβος που δηλώνει την ύπαρξη του Xen.

Κατά την διαδικασία εκκίνησης του Dom0 γίνεται προσπέλαση του δέντρου με το υλικό. Ο Dom0 ανακαλύπτει το διαθέσιμο υλικό και φορτώνει του κατάλληλους οδηγούς για την ορθή λειτουργία του. Κατά την προσπέλαση του δέντρου βρίσκει το κόμβο που περιέχει τον Xen και έτσι γνωρίζει ότι εκτελείται σε εικονικό περιβάλλον.

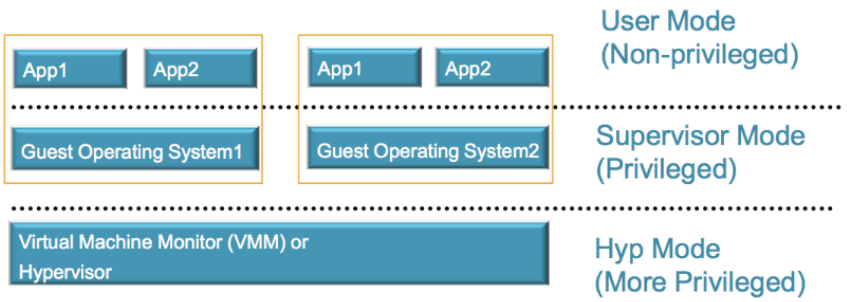
3.3 Xen on Arm σε κινητές συσκευές

Όσο αφορά τον Xen on ARM σε κινητές συσκευές εδώ η πολυπλοκότητα του προβλήματος αυξάνεται από το γεγονός ότι αυξάνονται οι συσκευές που πρέπει να διαχειριστεί το εικονικό περιβάλλον. Οι σύγχρονες κινητές συσκευές έχουν κάμερα, ποτενσιόμετρα, gps και διάφορες άλλες συσκευές.

Αν υπάρχει ανάγκη για χρήση μιας συσκευής από μόνο ένα εικονικό μηχάνημα τότε είναι εύκολη η ανάθεση αυτής της συσκευής στο συγκεκριμένο εικονικό μηχάνημα. Όμως σε περίπτωση που παραπάνω του ενός απαιτούν πρόσβαση στην συγκεκριμένη συσκευή πρέπει να γίνει χρήση ειδικών οδηγών. Παρά το γεγονός ότι υπάρχουν αρκετοί οδηγοί ανοιχτού κώδικα σε πολλές περιπτώσεις υπάρχει ανάγκη να υλοποιηθούν νέοι. Η πολυπλοκότητα της δημιουργίας τους αυξάνει με την πολυπλοκότητα των συσκευών. Για τον λόγο αυτό η συντήρηση και ανάπτυξη του Xen on ARM σε κινητές συσκευές είναι δαπανηρή.

3.4 Επεκτάσεις της αρχιτεκτονικής ARM για εικονικοποίηση

Όσο αφορά την εικονικοποίηση σε φυσικό επίπεδο είναι βασισμένη πάνω σε μια άλλη τεχνολογία της ARM το TrustZone[3]. Για να μπορέσει να καλύψει τις ανάγκες εικονικοποίησης έχει γίνει εισαγωγή ενός νέου επιπέδου δικαιωμάτων εκτέλεσης στον επεξεργαστή το “hyp mode”.Το συγκεκριμένο επίπεδο διαθέτει τους δικούς του καταχωρητές οι οποίοι δεν μπορούν να επηρεαστούν από κανένα άλλο επίπεδο δικαιωμάτων. Με την τεχνική αυτή μπορεί να υποστηριχθεί σε επίπεδο υλικού πλέον η διαστρωμάτωση του Xen (Εικόνα 8) που αναφέρθηκε στο προηγούμενο κεφάλαιο.



Εικόνα 8 Hyp mode

Ένα ακόμα βασικό χαρακτηριστικό της νέας αυτής γενιάς επεξεργαστών είναι η εισαγωγή του μηχανισμού LPAE. Ο μηχανισμός αυτός προσφέρει μετάφραση διευθύνσεων μεγέθους έως 40 bits. Με τον τρόπο αυτό κάθε εικονική μηχανή είναι σε θέση να διαχειρίζεται αποτελεσματικά την διαθέσιμη φυσική μνήμη.

Επιπρόσθετα καθώς η πολυπλοκότητα του λογισμικού αυξάνει, γίνεται ολοένα και πιο επιτακτική η ανάγκη συγκεκριμένες διεργασίες να εκτελούνται στην ίδια φυσική επεξεργαστική μονάδα. Για να μπορέσει η ARM να υποστηρίξει αυτήν την ανάγκη έχει αναπτύξει μηχανισμούς δημιουργίας εικονικών επεξεργαστών. Για την αποδοτικότερη λειτουργία του συγκεκριμένου μηχανισμού έχει γίνει συνδυασμός τόσο υλικού όσο και λογισμικού. Η ARM έχει δώσει οδηγίες στους εκάστοτε κατασκευαστές υπερεποπτών ώστε να τα λογισμικά τους να κατασκευάζονται με τέτοιο τρόπο ώστε να εκμεταλλεύονται το ειδικό υλικό του επεξεργαστή.

Τέλος ένα ακόμα βασικό χαρακτηριστικό της νέας αυτής γενιάς επεξεργαστών είναι η second level MMU. Η συγκεκριμένη μονάδα επιτρέπει την αντιστοίχηση 32-bit εικονικής μνήμης σε 40-bit φυσικής μνήμης. Με αυτό τον τρόπο υπάρχει η δυνατότητα αποδοτικότερης διαχείρισης της μνήμης. Όμως πέραν του second level MMU η ARM προσφέρει και ειδικούς μηχανισμούς για την διαχείριση των διακοπών (interrupts) των εικονικών μηχανών. Όταν φτάσει μια διακοπή στο επεξεργαστή τότε ενεργοποιείται ο μηχανισμός GIC (Generic Interrupt Controller). Ο GIC αναλαμβάνει να αξιολογήσει την προτεραιότητα κάθε διακοπής και να την προωθήσει στον υπερεπόπτη. Ο υπερεπόπτης με την σειρά του αναλαμβάνει να ανακατευθύνει στην σωστή εικονική μηχανή την εκάστοτε διακοπή.

Οι επεξεργαστές της ARM προορίζονται κατά κύριο λόγο για κινητές συσκευές. Για τον λόγο αυτό όλοι οι παραπάνω μηχανισμοί έχουν αναπτυχθεί με σκοπό τόσο της αύξησης της απόδοσης του συστήματος όσο και της μείωσης της ενεργειακής κατανάλωσης.

4 Ανάλυση περιβάλλοντος πειραμάτων

Σε αυτό το κεφάλαιο θα γίνει ανάλυση του περιβάλλοντος των πειραμάτων που εκτελέστηκαν στα πλαίσια της διατριβής. Αρχικά υπάρχει αναλυτική περιγραφή του εξοπλίσου που έγινε χρήση τόσο σε επίπεδο υλικού όσο και σε επίπεδο λογισμικού. Στην συνέχεια υπάρχει περιγραφή των βημάτων που χρειάστηκαν ώστε να το περιβάλλον να είναι έτοιμο για χρήση.

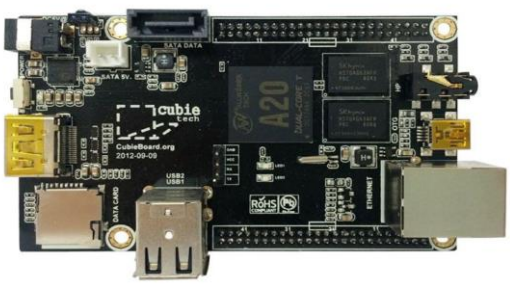
4.1 Υλικό

Στα πλαίσια της εργασίας χρησιμοποιήθηκε μια πλακέτα ολοκληρωμένου υπολογιστικού συστήματος(single-board computer).Συγκεκριμένα έγινε χρήση του Cubieboard2 το οποίο κατασκευάζεται στην Zhuhai,Guangdong,China για την cubietech. Αποτελεί την δεύτερη έκδοση της σειράς και πρωτοεμφανίστηκε στην αγορά τον Ιούνιο του 2013. Η βασική διαφορά με τον προκάτοχο του είναι η αντικατάσταση του επεξεργαστή. Η νέα έκδοση κάνει χρήση του A20 που εμπεριέχει δύο ARM Cortex-A7 πυρήνες. Η συγκεκριμένη έκδοση είναι η πρώτη της σειράς που υποστηρίζει εικονικοποίηση υλικού.

Τα τεχνικά του χαρακτηριστικά της είναι τα παρακάτω:

- AllWinnerTech SOC A20, ARM® Cortex™-A7 Dual-Core ARM® Mali400 MP2 Complies with OpenGL ES 2.0/1.1
- 1GB DDR3 @480M
- 3.4GB internal NAND flash, up to 32GB on SD slot, up to 2T on 2.5 SATA disk
- 5VDC input 2A or USB otg input
- 1x 10/100 ethernet, support usb wifi
- 2x USB 2.0 HOST, 1x mini USB 2.0 OTG, 1x micro sd
- 1x HDMI 1080P display output
- 1x IR, 1x line in, 1x line out
- 96 extend pin interface, including I2C, SPI, RGB/LVDS, CSI/TS, FM-IN, ADC, CVBS, VGA, SPDIF-OUT, R-TP, and more

Τέλος για αποθηκευτική μονάδα έγινε χρήση μιας micro sdcard της εταιρίας SanDisk.Πιο συγκεκριμένα χρησιμοποιήθηκε η microSDHC 16GB Class 10 - SanDisk Ultra.



Εικόνα 9 Cubieboard 2

4.1.1 Αρχιτεκτονική ARM® Cortex™-A7

Ο επεξεργαστής που κάνει χρήση το Cubieboard2 είναι ο Allwinner A20 ο οποίος είναι των 55nm και περιέχει 2 πυρήνες τύπου ARM Cortex-A7 με 256 KB L2 cache.

Η σειρά Cortex-A7 είναι αρχιτεκτονικής 32-bit και υποστηρίζει εικονικοποίηση σε επίπεδο υλικού. Η σωλήνωση είναι βασισμένη σε οκτώ επίπεδα και δεν υποστηρίζει Out-of-order εκτέλεση. Η ταχύτητα του κάθε πυρήνα είναι 1.9Mhz και το μέγεθος της L1 cache 32KB ανά πυρήνα. Η συγκεκριμένη σειρά της ARM θεωρείται μια από τις πιο επιτυχημένες καθώς έχει πουλήσει δισεκατομμύρια κομμάτια ανά τον κόσμο.

4.2 Λειτουργικό

Στα πλαίσια της μεταπτυχιακής διατριβής έγινε χρήση ειδικής έκδοσης του MirageOS για ARM. Η ειδική έκδοση που χρησιμοποιήθηκε αυτοματοποιεί την διαδικασία δημιουργίας περιβάλλοντος Xen στο Cubieboard2. Πιο συγκεκριμένα εμπεριέχει την έκδοση Xen 4.4.1 και κατά την εγκατάσταση του λειτουργικού για τον Dom0 κάνει χρήση του Ubuntu 14.04.3 LTS. Το ίδιο λειτουργικό σύστημα έγινε χρήση και για τους DomU.

4.3 Λογισμικό

Για τα πειράματα που εκτελέστηκαν έγινε χρήση διαφόρων εργαλείων αναφοράς καθώς και μιας εφαρμογής checkpointing. Τα εργαλεία αναφοράς που εκτελέστηκαν ήταν τα LMbench και miBench και η εφαρμογή checkpointing ήταν η DMTCP.

Το LMbench αποτελεί μια πλατφόρμα ανοιχτού κώδικα που αξιολογεί τα επιμέρους συστατικά του υπολογιστικού συστήματος. Το συγκεκριμένο λογισμικό είναι ευρέως αποδεκτό καθώς προσφέρει μια σφαιρική αξιολόγηση του υπολογιστικού συστήματος. Για τον λόγο αυτό επιλέχθηκε στην παρούσα πτυχιακή. Όμως το LMbench δεν είναι εξειδικευμένο λογισμικό για ενσωματωμένα συστήματα και για τον λόγο αυτό στα πλαίσια της εργασίας έγινε χρήση και του miBench. Το miBench εξειδικεύεται σε ενσωματωμένες συσκευές και επιπρόσθετα προσφέρει ευελιξία στα πειράματα που προσφέρει καθώς αποτελείται από πολλά μικρά προβλήματα που εκτελούνται αυτόνομα. Τέλος, έγινε χρήση του DMTCP το οποίο είναι εργαλείο checkpointing. Το checkpointing είναι μια διαδικασία που κάνει εκτεταμένη χρήση I/O και για τον λόγο αυτό επιλέχθηκε.

4.3.1 LMBench

Το LMBench[4] αποτελεί ένα λογισμικό ανοιχτού κώδικα για την σύγκριση της απόδοσης διαφόρων συστημάτων UNIX.Έχει κατασκευαστεί από τους Larry Mc Voy και Carl Staelin.Εκτελεί μια πληθώρα ελέγχων ώστε να αξιολογήσει την απόδοση του συστήματος. Οι έλεγχοι χωρίζονται σε τρείς βασικές κατηγορίες εύρους ζώνης, καθυστέρησης, επεξεργαστικής ισχύς.

Στην κατηγορία του εύρους ζώνης εμπεριέχονται έλεγχοι όπως, αντιγραφή μνήμης, εγγραφή μνήμης, προσπέλαση μνήμης, διοχέτευσης, καθώς και δικτύου TCP.

Στην κατηγορία καθυστέρησης εμπεριέχονται έλεγχοι που σχετίζονται με τον χρόνο καθυστέρησης που εισάγει το σύστημα σε εφαρμογές όπως δημιουργία και διαγραφή αρχείων συστήματος, δημιουργία διεργασίας, σηματοδοσία συστήματος, προσπέλαση μνήμης, κλήσεις συστήματος, δικτύου και context switching.

Τέλος έχουμε τον έλεγχο της απόδοσης του επεξεργαστή ο οποίος υπολογίζεται βάσει της ταχύτητας εκτέλεσης διαφόρων μαθηματικών πράξεων.

4.3.2 miBench

Το miBench[5] είναι ένα λογισμικό ανοιχτού κώδικα που εξειδικεύεται σε ενσωματωμένες συσκευές. Έχει κατασκευαστεί από το πανεπιστήμιο του Michigan από το τμήμα ηλεκτρολόγων μηχανικών και επιστήμης πληροφορικής. Σε αντίθεση με το LMBench το miBench χωρίζει τους ελέγχους σε εμπορικές κατηγορίες. Όπως φαίνεται στον πίνακα 2 υπάρχει πληθώρα διαφορετικών ελέγχων. Επίσης ακολουθεί σχετικό διάγραμμα με τις αντίστοιχες λειτουργίες που αξιοποιεί ο κάθε έλεγχος.

Στα πλαίσια αυτής της εργασίας έγινε χρήση των παρακάτω πειραμάτων:

Basic math:Ο συγκεκριμένος έλεγχος εκτελεί απλούς μαθηματικούς υπολογισμούς όπως τετραγωνική ρίζα, δύναμη τετραγώνου ακεραίου, μετατροπή μοιρών σε ακτίνα. Στα ενσωματωμένα υπολογιστικά συστήματα είναι σύνηθες να μην υπάρχει εξειδικευμένο υλικό για τον υπολογισμό αυτών των πράξεων. Τα δεδομένα εισόδου είναι σταθερά και παρέχονται στο πρόγραμμα από αρχείο δεδομένων. Ο συνολικός αριθμός των πράξεων είναι 65.459.080 για το μικρό σύνολο δεδομένων και 1.000.000.000 για το μεγάλο.

Qsort:Η δοκιμή qsort ταξινομεί μια μεγάλη σειρά δεδομένων σε αύξουσα σειρά χρησιμοποιώντας το γνωστό αλγόριθμο ταξινόμησής quick qsort. Η ταξινόμηση των δεδομένων αποτελεί μια βασική λειτουργία για τα υπολογιστικά συστήματα. Το μικρό σύνολο δεδομένων αποτελείται από λέξεις ενώ το μεγάλο από τριάδες λέξεων χωρισμένες με κόμμα. Ο συνολικός αριθμός των πράξεων είναι 43.604.903 για το μικρό σετ και 595.400.120 για το μεγάλο.

Dijktra: Ο έλεγχος αυτός κάνει χρήση του αλγορίθμου Dijkstra για τον υπολογισμό της συντομότερης διαδρομής σε ένα γράφημα. Ο συγκεκριμένος αλγόριθμος έχει γνωστό κόστος λύσης και ολοκληρώνεται σε $O(n^2)$.Το μικρό σετ δεδομένων αποτελείται από 64.927.863 πράξεις ενώ το μεγάλο 272.657.564.

Patricia: Ο συγκεκριμένος έλεγχος εκτελείται σε δέντρα δεδομένων και κάνει χρήση του αλγορίθμου Patricia που συχνά συναντάται στα δίκτυα. Ο συγκεκριμένος αλγόριθμος έχει σαν είσοδο μια λίστα από IP κίνηση σε έναν εξυπηρετητή. Το μικρό σύνολο αποτελείται από 103.923.656 πράξεις ενώ το 1.000.000.000 πράξεις.

Bitcount: Η δοκιμή αυτή έχει σαν στόχο τον έλεγχο της ικανότητας του επεξεργαστή στην καταμέτρηση των bit σε ένα πίνακα ακεραίων. Για να το επιτύχει αυτό κάνει χρήση διαφόρων αλγορίθμων. Σαν είσοδο έχει έναν πίνακα ακεραίων που αποτελείται από ίσο αριθμός 1 και 0. Για να προσδιοριστεί το μέγεθος του πίνακα ο χρήστης πρέπει να το ορίσει κατά την εκτέλεσή του. Ο συγκεκριμένο έλεγχος δεν έχει σαν είσοδο αρχείο καθώς τα δεδομένα δημιουργούνται από το ίδιο το πρόγραμμα.

BFSPEED:Ο συγκεκριμένος έλεγχος κάνει χρήση του αλγορίθμου Blowfish. Αρχικά αντιγράφει στην μνήμη ένα αρχείο ASCII με κείμενο και στην συνέχεια εκτελεί κωδικοποίηση και αποκωδικοποίηση του αρχείου με διαφορετικά μεγέθη κλειδιών. Στον συγκεκριμένο έλεγχο υπολογίζεται μόνο ο καθαρός χρόνος υπολογισμού των κλειδιών και όχι ο χρόνος που απαιτείται για την αντιγραφή του αρχείου ASCII. Ο συγκεκριμένος έλεγχος αποτελείται από 94.807.682 πράξεις για το μικρό σετ 984.691.122 για το μεγάλο τεστ.

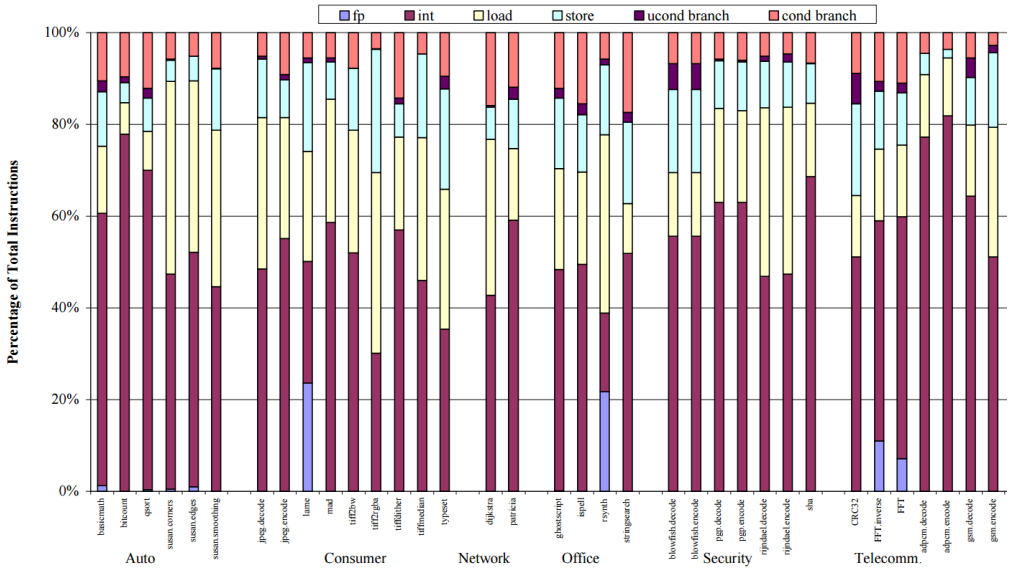
Ο λόγος που έγινε η επιλογή των συγκεκριμένων ελέγχων είναι η διαφορετικότητα στην φύση τους καθώς ο καθένας ελέγχει διαφορετική λειτουργία του συστήματος. Με τον τρόπο αυτό μπόρεσε να γίνει εκτίμηση της απόδοσης του επεξεργαστή, της μνήμης καθώς και του δίσκου. Στο διάγραμμα που ακολουθεί υπάρχουν τα ποσοστά των εκάστοτε εντολών που εκτελεί κάθε πρόγραμμα αξιολόγησης.

Παρατηρούμε ότι το πρόγραμμα Dijktra έχει μεγάλο αριθμό εντολών φόρτωσης δεδομένων καθώς και εντολών διακλάδωσης. Από την άλλη πλευρά το bitcount αποτελεί κυρίως υπολογιστικό πρόβλημα και εκτελεί κατά κύριο λόγο εντολές int. Τα προβλήματα basicmath,qsort και patricia αποτελούν προβλήματα γενικού τύπου και εκτελούν 50-65% εντολές int και το υπόλοιπο ποσοστό αποτελείται από εντολές store,load και code branch. Τέλος το μόνο πρόγραμμα που δεν επιλέχθηκε με βάση του παρακάτω διαγράμματος είναι αυτό του

blowfish καθώς στους χρόνους που δίνει σαν αποτέλεσμα αγνοεί τις εντολές load και store και λαμβάνει υπόψιν μόνο τον καθαρό χρόνο κωδικοποίηση και αποκωδικοποίησης των κλειδιών. Με τον τρόπο αυτό παρέχει μια καθαρή εικόνα για την απόδοση της κεντρικής μονάδας επεξεργασίας του συστήματος.

Auto/Industrial	Consumer	Office	Network	Security	Telecomm
basicmath	Jpeg	ghostscript	dijkstra	Blowfish enc	CRC32
bitcount	Lame	ispell	patricia	Blowfish dec	FFT
qsort	tiff2bw	rsynth	CRC32	Pgp sign	IFFT
susan (edges)	tiff2rgba	sphinx	Sha	Pgp verify	ADPCM enc
susan(corners)	tiffdither	stringsearch	blowfish	Rijndael enc	ADPCM dec
susan(smoothing)	tiffmedian			Rijndael dec	GSM enc
	typeset			sha	GSM dec
	Mad				

Πίνακας 5 MiBench Benchmarks



Διάγραμμα 1 Επί της εκατό τύπος εντολών

4.3.3 DMTCP

Η τεχνική του checkpointing εισάγει fault tolerance στα υπολογιστικά συστήματα. Κατά την εκτέλεση ενός προγράμματος η τεχνική του checkpointing αναλαμβάνει να αποθηκεύσει μια εικόνα του συστήματος. Με αυτόν τον τρόπο σε περίπτωση που προκύψει κάποιο σφάλμα κατά την εκτέλεση του προγράμματος υπάρχει η δυνατότητα επανεκκίνησης σε μια προηγούμενη κατάσταση. Η συγκεκριμένη τεχνική είναι ιδιαίτερα σημαντική καθώς προστατεύει προγράμματα με μεγάλο χρόνο εκτέλεση που τρέχουν σε επίφοβα υπολογιστικά συστήματα.

Υπάρχουν διάφορές υλοποιήσεις που εκτελούν την παραπάνω διαδικασία με τις δημοφιλέστερες να είναι οι παρακάτω:

- FTI (Fault Tolerance Interface)
- BLCR (Berkeley Lab Checkpoint/Restart)
- DMTCP
- Mementos
- Idetic

Στα πλαίσια αυτής της πτυχιακής επιλέχθηκε να γίνει χρήση του DMTCP. Το DMTCP είναι αρκετά φιλικό προς τον χρήστη και παρέχει αρκετές δυνατότητες παραμετροποίησης. Ένα από τα βασικά πλεονεκτήματα της συγκεκριμένης υλοποίησης είναι ότι υποστηρίζει τις περισσότερες γλώσσες προγραμματισμού. Τα εκάστοτε προγράμματα συνδέονται στον εξυπηρετητή του DMTCP με χρήση socket. Ένα από τα βασικά πλεονεκτήματα του DMTCP είναι ότι δεν απαιτεί μετατροπή του εκάστοτε λειτουργικού ή της διεργασίας που επιθυμεί να δημιουργήσει σημείο επαναφοράς. Τα σημεία επαναφοράς δημιουργούνται κατά τον χρόνο εκτέλεσης της διεργασίας και αποθηκεύονται στο δίσκο του συστήματος.

4.4 Περιγραφή βημάτων εγκατάστασης συστήματος

Τα βήματα για την εγκατάσταση του συστήματος χωρίζονται σε δυο βασικές κατηγορίες. Αρχικά στην δημιουργία μιας εικόνας του περιβάλλοντος και στην συνέχεια εγκατάσταση των εικονικών μηχανών.

4.4.1 Δημιουργία εικόνας

Για να μπορέσουμε να δημιουργήσουμε μια εικόνα του περιβάλλοντος πρέπει αρχικά να κατεβάσουμε τα κατάλληλα αρχεία για την συσκευή cubieboard2. Όπως προαναφέρθηκε το MirageOs παρέχει ειδική έκδοση για ARM επεξεργαστές. Αφού κατεβάσουμε την κατάλληλη έκδοση πρέπει να ορίσουμε την πλακέτα την οποία κάνουμε χρήση. Στα πλαίσια της πτυχιακής έγινε χρήση της cubieboard2 όποτε στο φάκελο του MirageOS εκτελούμε την εντολή

```
$  export Board=cubieboard2.
```

Στην συνέχεια χρειάζεται να εγκαταστήσουμε ένα πακέτο cross-toolchain. Το λειτουργικό που έγινε χρήση είναι Ubunty 14.04 οπότε εκτελώντας την εντολή :$apt-get install gcc-arm-linux-gnueabihf κατεβάζουμε τα απαραίτητα αρχεία. Στην συνέχεια ενώ βρισκόμαστε στον φάκελο ρίζα του MirageOS εκτελούμε την εντολή $make clone με αυτήν την εντολή κατεβάζουμε όλα τα απαραίτητα αρχεία που χρειάζεται η διαδικασία του compile.

Παρά το γεγονός ότι το MirageOs περιέχει στο πακέτο του έτοιμα αρχεία για την δημιουργία του U-boot χρειάστηκε να γίνουν αλλαγές σε αυτά για την ορθή εκκίνηση του συστήματος καθώς και για την διαχείριση της μνήμης του Dom0.Το πακέτο που παρέχει το MirageOs είναι σχεδιασμένο για την πλακέτα cubietruck για τον λόγο αυτό χρειάστηκαν να γίνουν οι αλλαγές στο αρχείο boot-cubietruck.cmd ώστε οι θέσεις μνήμης να ταιριάζουν με αυτές του cubieboard. Παρακάτω υπάρχει το ολοκληρωμένο αρχείο boot-cubietruck.cmd με τις σωστές θέσεις μνήμης για cubieboard.

```
# Top of RAM:        0xc0000000
# Xen relocate addr   0xbfe00000
setenv fdt_addr       0xaec00000 # 2M
# Top of RAM:        0x80000000
# Xen relocate addr   0x7fe00000
```

```
setenv fdt_addr       0x7ec00000 # 2M
setenv fdt_high       0xffffffff # Load fdt in place instead of relocating
# Put Xen and Linux in the top half of RAM, or you get
# These used to be at 0x7... but with the latest U-Boot that makes us hang when
# loading the vmlinuz (use U-Boot's "bdinfo" to see what's in the way).
setenv kernel_addr_r  0x6ee00000
setenv xen_addr_r     0x6ea00000 # 2M
# sunxi-common.h contains some more useful information:
# CONFIG_SYS_TEXT_BASE  0x4a000000 - location of U-Boot's code
# CONFIG_SYS_SDRAM_BASE 0x40000000 - start of RAM
# Load xen/xen to ${xen_addr_r}.
fatload mmc 0 ${xen_addr_r} /xen
setenv bootargs "console=dtuart dtuart=/soc@01c00000/serial@01c28000
dom0_mem=1024M,max:1024M"
# Load appropriate .dtb file to ${fdt_addr}
fatload mmc 0 ${fdt_addr} /sun7i-a20-cubieboard2.dtb
fdt addr ${fdt_addr} 0x40000
fdt resize
fdt chosen
fdt set /chosen \#address-cells <1>
fdt set /chosen \#size-cells <1>
# Load Linux arch/arm/boot/zImage to ${kernel_addr_r}
fatload mmc 0 ${kernel_addr_r} /vmlinuz
fdt mknod /chosen module@0
fdt set /chosen/module@0 compatible "xen,linux-zimage" "xen,multiboot-module"
fdt set /chosen/module@0 reg <${kernel_addr_r} 0x${filesize} >
fdt set /chosen/module@0 bootargs "console=hvc0 ro root=/dev/mmcblk0p2 rootwait
clk_ignore_unused"
bootz ${xen_addr_r} - ${fdt_addr}
```

Σε αυτό το σημείο αυτό πρέπει να γίνει δημιουργία του λειτουργικού συστήματος Linux για τον Dom0. Αφού γίνει λήψη των απαραίτητων αρχείων από το MirageOs και ενώ βρισκόμαστε στον φάκελο του λειτουργικού συστήματος εκτελούμε τις παρακάτω εντολές.

```
make ARCH=arm multi_v7_defconfig
make ARCH=arm menuconfig
```

Για την ορθή δημιουργία της εικόνας πρέπει να γίνουν οι παρακάτω ρυθμίσεις στα αρχεία παραμετροποίησης του συστήματος.

```
CONFIG_CROSS_COMPILE="/usr/bin/arm-linux-gnueabihf-"
CONFIG_XEN_DOM0=y
CONFIG_XEN=y
CONFIG_IPV6=y
CONFIG_NETFILTER=y
CONFIG_NETFILTER_ADVANCED=y
CONFIG_BRIDGE_NETFILTER=y
CONFIG_STP=y
CONFIG_BRIDGE=y
CONFIG_SYS_HYPERVISOR=y
CONFIG_XEN_BLKDEV_FRONTEND=y
CONFIG_XEN_BLKDEV_BACKEND=y
CONFIG_AHCI_SUNXI=y
CONFIG_XEN_NETDEV_FRONTEND=y
```

```
CONFIG_XEN_NETDEV_BACKEND=y
CONFIG_INPUT_AXP20X_PEK=y
CONFIG_INPUT_XEN_KBDDEV_FRONTEND=y
CONFIG_HVC_DRIVER=y
CONFIG_HVC_IRQ=y
CONFIG_HVC_XEN=y
CONFIG_HVC_XEN_FRONTEND=y
CONFIG_MFD_AXP20X=y
CONFIG_REGULATOR_AXP20X=y
CONFIG_FB_SYS_FOPS=y
CONFIG_FB_DEFERRED_IO=y
CONFIG_XEN_FBDEV_FRONTEND=y
CONFIG_MMC_SUNXI=y
CONFIG_VIRT_DRIVERS=y
CONFIG_XEN_BALLOON=y
CONFIG_XEN_SCRUB_PAGES=y
CONFIG_XEN_DEV_EVTCHN=y
CONFIG_XEN_BACKEND=y
CONFIG_XENFS=y
CONFIG_XEN_COMPAT_XENFS=y
CONFIG_XEN_SYS_HYPERVISOR=y
CONFIG_XEN_XENBUS_FRONTEND=y
CONFIG_XEN_GNTDEV=y
CONFIG_XEN_GRANT_DEV_ALLOC=y
CONFIG_SWIOTLB_XEN=y
CONFIG_XEN_PRIVCMD=y
CONFIG_PHY_SUN4I_USB
CONFIG_HAS_IOPORT=y
# LVM
CONFIG_MD=y
CONFIG_BLK_DEV_DM_BUILTIN=y
CONFIG_BLK_DEV_DM=y
CONFIG_DM_BUFIO=y
CONFIG_DM_SNAPSHOT=y
```

Τέλος, πρέπει να εκτελεστεί η δημιουργία του Xen.Για τον λόγο αυτό στον φάκελο του Xen εκτελούμε την εντολή

```
make dist-xen XEN_TARGET_ARCH=arm32 CROSS_COMPILE=$CROSS_COMPILE
CONFIG_EARLY_PRINTK=sun7i -j4
```

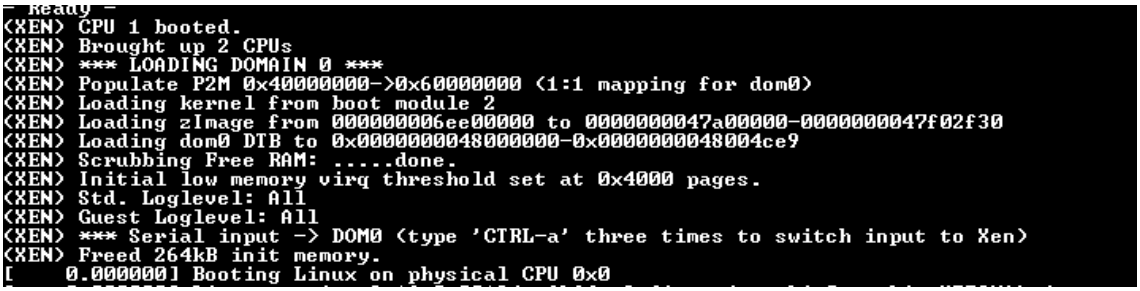
Μετά το πέρας της διαδικασίας που προαναφέρθηκε έχουν δημιουργηθεί όλα τα απαραίτητα αρχεία για την εκκίνηση του συστήματος. Το μόνο που μένει πλέον είναι η αντιγραφή των αρχείων στην sdcard που θα γίνει χρήση ως κεντρική αποθηκευτική μοναδά.

4.4.2 Εκκίνηση συστήματος

Κατά την διαδικασία εκκίνησης αρχικά γίνεται φόρτωση του U-boot το οποίο είναι υπεύθυνο για να εκκινήσει ο Xen. Το U-Boot είναι ένας ελεγκτής εκκίνησης ανοιχτού κώδικα που χρησιμοποιείται κυρίως σε ενσωματωμένα συστήματα.

Ο Xen με την σειρά του εκκινεί σε hyp mode όπως φαίνεται στην εικόνα 9. Ο Xen δημιουργεί ένα δέντρο με το υλικό της πλακέτας εδώ πρέπει να σημειωθεί ότι μια είσοδος UART δεσμεύεται από τον Xen και δεν εμφανίζεται στο δέντρο των συσκευών. Στην συνέχεια ο Xen αναλαμβάνει την εκκίνηση του Dom0 όπως φαίνεται στην εικόνα 10.Τέλος με την χρήση του Xen Virtual Firmware εκτελείται η εκκίνηση του Dom0 που δεν διαφέρει από την κανονική εκκίνηση του Ubuntu σε μη εικονικοποιημένο περιβάλλον. Στο παράρτημα Α υπάρχει η πλήρης διαδικασία εκκίνησης.

Εικόνα 10 Xen Boot



Εικόνα 11 Dom0

4.4.3 Εκκίνηση και δημιουργία DomU

Για να μπορέσουμε να έχουμε πρόσβαση στην κονσόλα του συστήματος θα πρέπει να γίνει είτε χρήση ssh είτε χρήση της σειριακής θύρας. Κατά την πρώτη εκκίνηση έγινε χρήση της σειριακής θύρας ώστε να παραμετροποιηθεί το ssh. Επίσης κατά την πρώτη εκκίνηση πρέπει να είναι συνδεδεμένες όλες οι συσκευές που υπάρχουν στο σύστημα ώστε ο Xen αυτόματα να δημιουργήσει το δέντρο συσκευών. Στα πλαίσια της εργασίας έγινε χρήση του πρωτοκόλλου ssh για σύνδεση με τον Dom0. Για την εγκατάσταση του χρειάστηκε η εκτέλεση των παρακάτω εντολών μέσω της σειριακής θύρας.

```
mount -o remount,rw /mount -t proc proc /proc
export PATH=/bin:/usr/bin:/sbin:/usr/sbin
export HOME=/root
ifup eth0
ip addr show dev eth0
apt-get install openssh-server
cd
mkdir .ssh
vi .ssh/authorized_keys
```

Η δημιουργία των DomU είναι εφικτή μέσα από την γραμμή εντολών του Dom0. Οι περισσότερες διαδικασίες που περιγράφονται παρακάτω απαιτούν δικαιώματα su.

Για την εγκατάσταση των DomU πρέπει πρώτα να δημιουργήσουμε ένα εικονικό partition. Αυτό γίνεται με την βοήθεια των εργαλείων του LVM που είναι προ εγκατεστημένα στο περιβάλλον μας. Για να το πετύχουμε αυτό εκτελούμε τις παρακάτω εντολές:

```
pvcreate /dev/mmcblk0p3
vgcreate vg0 /dev/mmcblk0p3
 lvcreate -L 4G vg0 --name linux-guest-1
```

Με αυτό τον τρόπο δημιουργούμε ένα partition 4gb που ανήκει στην ομάδα vg0 και έχει όνομα linux-guest-1.Στην συνέχεια πρέπει να γίνει διαμόρφωση αυτού του partition. Για τον λόγο αυτό εκτελούμε την εντολή $mkfs.ext4 /dev/vg0/linux-guest-1.

Έχοντας ολοκληρώσει την διαδικασία δημιουργίας εικονικού partition πρέπει να δημιουργήσουμε την εικόνα που θέλουμε να κάνουμε χρήση. Αυτό μπορούμε να καταφέρουμε με την εκτέλεση των παρακάτω τριών εντολών:

```
$mount /dev/vg0/linux-guest-1 /mnt
$debootstrap --arch armhf trusty /mnt
$chroot /mnt
```

Πλέον το λειτουργικό βρίσκεται στο εικονικό partition.Στην συνέχεια πρέπει να γίνει εγκατάσταση του δικτύου του DomU.Για να μπορέσουμε να το πετύχουμε αυτό πρέπει να γίνουν αλλαγές στα αρχεία /etc/hostname και /etc/network/interfaces.Στο πρώτο αρχείο πρέπει να προστεθεί η γραμμή auto eth0 και στο δεύτερο αρχείο την γραμμή iface eth0 inet dhcp.Πλέον το δίκτυο είναι σωστά ρυθμισμένο ώστε να κάνει χρήση DHCP.

Το επόμενο βήμα είναι η δημιουργία χρηστών για τον DomU.Για τον λόγο αυτό εκτελούμε την εντολή chroot /mnt με τις αντίστοιχες παραμέτρους που επιθυμούμε για τον νέο χρήστη. Στην συνέχεια πρέπει να ρυθμιστεί το fstab. Για τον λογο αυτό στο αρχείο /etc/fstab προσθέτουμε την γραμμή /dev/xvda / ext4 rw,relatime,data=ordered 0 1.

Ακόμα πρέπει να γίνει εγκατάσταση του openssh ώστε να υπάρχει η δυνατότητα σύνδεσης με την γραμμή εντολών του DomU.Με την εκτέλεση της εντολής $chroot /mnt apt-get install -y openssh-server avahi-daemon κάνουμε εγκατάσταση του openssh server και το avahi-daemon.

Τέλος πρέπει να κάνουμε δημιουργία του κατάλληλου αρχείου αρχικοποίησης ώστε ο Xen να γνωρίζει τις λεπτομέρειες για το εικονικό μηχάνημα. Το αρχείο πρέπει να έχει κατάληξη. conf και η δομή του είναι η ακόλουθη.

Στην πρώτη γραμμή ορίζουμε την θέση του πυρήνα που θα γίνει χρήση. Στην συνέχεια ορίζουμε το μέγεθος της μνήμης σε Mb. Στην τρίτη γραμμή ονοματίζουμε το DomU. Το όνομα κάθε DomU πρέπει να είναι μοναδικό. Στην συνέχεια ορίζουμε τον αριθμό των εικονικών επεξεργαστών. Στην πέμπτη γραμμή ορίζουμε το κομμάτι δίσκου που θα κάνει χρήση η εικονική μηχανή. Στην προτελευταία γραμμή πρέπει να δηλώσουμε το δίκτυο. Στα πλαίσια της εργασίας έγινε χρήση του interface xenbr0 που αποτελεί γέφυρα ανάμεσα στο Dom0 και στον DomU. Τέλος ορίζουμε επιπρόσθετες πληροφορίες σχετικές με την κονσόλα του εικονικού μηχανήματος και την θέση της ρίζας.

```
kernel = "/root/dom0_kernel"
memory = 256
name = "linux-guest-1"
vcpus = 2
disk = [ "phy:/dev/vg0/linux-guest-1,xvda,w" ]
vif = ["bridge=xenbr0"]
extra = "console=hvc0 xencons=tty root=/dev/xvda'"
```

Πλέον ο DomU είναι έτοιμος για εκκίνηση. Η εκκίνηση των εικονικών μηχανημάτων γίνεται με την χρήση του εργαλείου xl.Για την δημιουργία ενός εικονικού μηχανήματος πρέπει να εκτελεστεί η εντολή $xl create όνομα_αρχείου.conf.Με τον τρόπο αυτό εκκινούμε το εικονικό μηχάνημα και πλέον μπορούμε να συνδεθούμε με ssh με την χρήση της εντολής $ssh mirage@ονομα_domU.local .Η εικονική μηχανή που δημιουργήθηκε στα πλαίσια της

διατριβής κατέχει 256 mb μνήμης και κάνει χρήση 2 επεξεργαστών. Παρακάτω παρατίθεται και το script που δημιουργήθηκε για την αυτοματοποίηση της διαδικασίας που αναλύθηκε παραπάνω.

```
#!/bin/bash
set -uo errexit
echo "Creating guest partition..."
mkfs.ext4 /dev/mmcblk0p3
pvcreate /dev/mmcblk0p3
lvcreate -L 4G vg0 --name linux-guest-1
/sbin/mkfs.ext4 /dev/vg0/linux-guest-1
echo "Bootstrapping..."
mount /dev/vg0/linux-guest-1 /mnt && \
trap "umount /mnt" EXIT # umount on exit
debootstrap --arch armhf trusty /mnt
echo "Setting hostname..."
echo "linux-guest-1" > /mnt/etc/hostname
echo "Configuring networking to use DHCP..."
echo 'auto eth0
iface eth0 inet dhcp' > /mnt/etc/network/interfaces
echo "Adding mirage user"
chroot /mnt useradd -s /bin/bash -G sudo -m mirage -p mljnMhCVerQE6
chroot /mnt passwd root -l
echo "Setting up fstab"
echo "/dev/xvda       / ext4   rw,norelatime,nodiratime       0 1" > /mnt/etc/fstab
echo "Installing ssh and avahi..."
chroot /mnt apt-get install -y openssh-server avahi-daemon
echo "UseDNS no" >> /mnt/etc/ssh/sshd_config
echo "Creating linux-guest-1.conf..."
echo 'kernel = "/root/dom0_kernel"
memory = 256
name = "linux-guest-1"
vcpus = 2
serial="pty"
disk = [ "phy:/dev/vg0/linux-guest-1,xvda,w" ]
vif = ["bridge=xenbr0"]
extra = "console=hvc0 xencons=tty root=/dev/xvda"' > linux-guest-1.conf
echo "Done!"
```

```
    echo "Start and attach to guest with"
    echo -e "\txl create linux-guest-1.conf"
    echo
    echo "Connect to the guest with"
    echo -e "\tssh mirage@linux-guest-1.local"
```

4.4.3 Διαχείριση εικονικών μηχανών

Στα πλαίσια της διατριβής έγινε δημιουργία πολλαπλών εικονικών μηχανών. Η διαχείριση πολλαπλών εικονικών μηχανών με τις βασικές εντολές που παρέχει ο Xen είναι δύσκολη και για τον λόγο αυτό έγινε χρήση του Libvirt.Το Libvirt είναι ένα πακέτο εργαλείων που επιτρέπει την ευκολότερη και αποτελεσματικότερη διαχείριση των εικονικών μηχανών. Για την εγκατάσταση του ακολουθήθηκαν οι οδηγίες που παρέχει ο επίσημος σύνδεσμος του MirageOs.Παρακάτω ακολουθεί ο κώδικας για την εγκατάσταση του εργαλείου.

```
    #!/bin/bash
LIBVIRT_FILE="libvirt.tar.gz"
LIBVIRT_URL=http://libvirt.org/sources/libvirt-1.2.8.tar.gz
if [ ! -e "$LIBVIRT_FILE" ]; then
        echo "Downloading $LIBVIRT_URL (to $LIBVIRT_FILE)"
        curl $LIBVIRT_URL '-L#o' $LIBVIRT_FILE
fi
mkdir Libvirt
tar xvf $LIBVIRT_FILE -C libvirt --strip-components=1 && \
cd libvirt && \
./configure --prefix=/usr --localstatedir=/var  --sysconfdir=/etc --with-xen --with-qemu=no --with-
gnutls --with-uml=no --with-openvz=no --with-vmware=no --with-phyp=no --with-xenapi=no --
with-libxl=yes --with-vbox=no --with-lxc=no --with-esx=no  --with-hyperv=no --with-parallels=no -
-with-init-script=upstart && \
make clean && \
    make -j3 && \
sudo make install && \
cd .. && \
rm -rf Libvirt
if [ ! -e "/etc/default/libvirt-bin" ]; then
        echo 'start_libvirtd="yes"
libvirtd_opts="-d"' > /etc/default/libvirt-bin
fi
if [ ! -e "/etc/init/libvirtd.conf" ]; then
        if [ -e "/etc/event.d/libvirtd" ]; then # libvirtd 1.2.8 installs its upstart script here
                echo "libvirtd upstart config installed in /etc/event.d, moving to /etc/init"
                sudo mv -v /etc/event.d/libvirtd /etc/init/libvirtd.conf
        else
                echo "Unable to add libvirtd to upstart, startup script not found. You may have to
configure it manually."
        fi
fi
sudo start libvirtd
```

5 Παρουσίαση και ανάλυση αποτελεσμάτων

Σε αυτό το κεφάλαιο γίνεται παρουσίαση και ανάλυση των αποτελεσμάτων που προέκυψαν από τα εργαλεία κατάταξης καθώς και από το DMTCP. Μέσα από την ανάλυση των αποτελεσμάτων γίνονται κατανοητοί οι λόγοι που οδηγούν στην καθυστέρηση του συστήματος.

5.1 Παρουσίαση και ανάλυση αποτελεσμάτων lmbench

Σε αυτό το κεφάλαιο παρουσιάζονται και αναλύονται τα αποτελέσματα του lmbench. Παρακάτω ακολουθεί συγκεντρωτικός πίνακας των αποτελεσμάτων του lmbench. Η πρώτη στήλη περιγράφει το είδος του ελέγχου που εκτελείται. Ο διαχωρισμός των ελέγχων γίνεται σε τέσσερις κατηγορίες οι οποίες είναι διεργασίες συστήματος(system calls),βασικές πράξεις μικρών ακεραίων, βασικές πράξεις int64 και context switch. Στην μεσαία στήλη παρουσιάζονται τα αποτελέσματα των πειραμάτων στον Dom0. Τέλος στην αριστερή στήλη υπάρχουν τα αποτελέσματα των πειραμάτων στον DomU. Η μονάδα μέτρησης χρόνου είναι το μs.

Test	**Dom0**	**DomU**
Διεργασίες συστήματος	**Ο χρόνος είναι σε μs όσο μικρότερος τόσο καλύτερος.**	**Ο χρόνος είναι σε μs όσο μικρότερος τόσο καλύτερος.**
Null call	0.23	0.26
Null I/O	0.63	0.64
Stat	3.73	3.78
Open clos	9.06	9.06
Slct TCP	32.2	32.5
Sig inst	1.00	1.00
Sig hndl	4.07	4.10
Fork proc	1162	1448
Exec proc	3575	3761
Sh proc	7959	8099
Βασικές πράξεις ακεραίων		
Intgr bit	1.16	1.16
Add	1.12	1.12
Mul	3.58	3.58
Div	88.2	88.2
Mod	27.9	27.9
Βασικές uint64 πράξεις		
Bit	2.310	2.310
Mul	5.97	5.97
Div	357.2	357.2
Mod	217.8	217.6
Βασικές πράξεις float		
Add	4.51	4.51
Mul	4.46	4.46
Div	20.6	20.6

Context switching		
2p/0K	4.61	4.71

Πίνακας 6 Αναλυτικοί χρόνοι εκτέλεσης για το μετροπρόγραμμα lmbench

Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα αποτελέσματα στις διεργασίες fork, exec. Όπως παρατηρούμε στους συγκεντρωτικούς πίνακες ο DomU έχει σημαντική καθυστέρηση έναντι του Dom0.

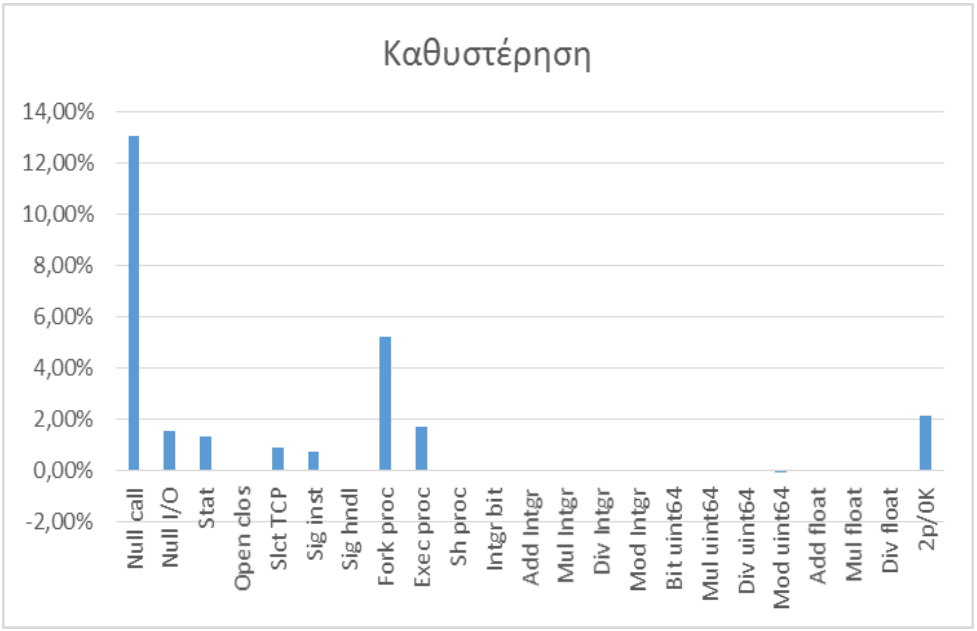
Αυτό το αποτέλεσμα ήταν αναμενόμενο καθώς αυτές οι διεργασίες απαιτούν ενημέρωση μεγάλου αριθμού από πίνακες σελίδων οι οποίοι πρέπει με την σειρά τους να επιβεβαιωθούν από τον υπερεπόπτη.

Επίσης ενδιαφέρον παρουσιάζουν τα αποτελέσματα που έχουμε στο context switch. Παρατηρούμε ότι DomU έχει μεγάλή καθυστέρηση στην εκτέλεση του καθώς πρέπει να εκτελεί υπερκλήσεις για την διευθέτηση τους.Η συγκεκριμένη τιμή δηλώνει ότι εκτελείτε context switching μεταξύ δυο διεργασιών χωρίς αυτές οι διεργασίες να εκτελούν κάποιο έργο.

Τα συγκεκριμένα αποτελέσματα ήταν αναμενόμενα λόγο της αρχιτεκτονικής που ακολουθεί ο υπερεπόπτης. Παρατηρούμε ότι η μέση καθυστέρηση του συστήματος είναι μόλις 2.23%.

Τα αποτελέσματα του Xen on ARM πλησιάζουν σε μεγάλο βαθμό τα αποτελέσματα του Xen x86 που παρουσιάστηκε στο κεφάλαιο δυο. Η μεταφορά του Xen για επεξεργαστές ARM είναι αρκετά επιτυχής και σε μεγάλο βαθμό πλησιάζει την απόδοση του Xen για αρχιτεκτονική x86.

Επίσης παρατηρούμε ότι δεν υπάρχει καθυστέρηση στις πράξεις στην κεντρική μονάδα επεξεργασίας. Το νέο επίπεδο που έχει εισαγάγει η ARM στους επεξεργαστές της λειτουργεί αποτελεσματικά και έχει μηδενίσει την καθυστέρηση στην κεντρική μονάδα επεξεργασίας. Παρακάτω στην εικόνα 11 ακολουθεί η συγκεντρωτική εικόνα του πειράματος.



Εικόνα 12 Καθυστέρηση ανά εφαρμογή

5.2 Παρουσίαση και ανάλυση αποτελεσμάτων miBench

Σε αυτήν την ενότητα θα γίνει παρουσίαση και ανάλυση των δεδομένων που προέκυψαν από το miBench. Τα πειράματα εκτελέστηκαν κάτω από τέσσερις διαφορετικές συνθήκες. Οι καταστάσεις που δοκιμάστηκε η απόδοση του συστήματος είναι οι παρακάτω:

- Dom0
- Dom0 με DMTCP
- DomU
- DomU με DMTCP

Για την ορθή εκτέλεση των πειραμάτων στο σύστημά μας χρειάστηκαν να γίνουν αλλαγές στον κώδικα του miBench στα αρχεία Basicmath large, Basicmath small,Qsort small,Qsort large,Patricia,Dijktra.Στο παράρτημα Β υπάρχει ο κώδικας των πειραμάτων. Επίσης κατά την διαδικασία του compile προστέθηκαν οι παράμετροι –WL,-Bdynamic.

Στους πίνακες που ακολουθούν φαίνονται αναλυτικά τα αποτελέσματα των πειραμάτων που εκτελέστηκαν. Είναι σημαντικό να σημειωθεί ότι σε κάθε πείραμα οι ρυθμίσεις του Dmtcp άλλαζαν ανάλογα με τις ανάγκες του προβλήματος.

Για το Basicmath large το DMTCP ρυθμίστηκε να λάβει 100 checkpoints. Για qsort large,patricia large,basic math small και Bit counter το DMTCP είχε περιοδικότητα ένα δευτερόλεπτο. Τέλος για όλα τα υπόλοιπα έλαβε ένα μόνο checkpoint.

Στους πίνακες που ακολουθούν στην πρώτη στήλη υπάρχει το είδος του πειράματος που εκτελέστηκε. Στην συνέχεια στην δεύτερη και τρίτη στήλη ακολουθούν τα αποτελέσματα του Dom0 και του Dom0 με Dmtcp αντίστοιχα. Τέλος στην τέταρτη και πέμπτη στήλη ακολουθούν τα αποτελέσματα του DomU και του DomU με Dmtcp αντίστοιχα.

Εξαίρεση αποτελεί ο πίνακας 8 που δεν περιλαμβάνει τα αποτελέσματα του Dom0 και του DomU με χρήση Dmtcp.Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο ο έλεγχος BFSPEED υπολογίζει μόνο τον καθαρό χρόνο υπολογισμού των κλειδιών. Εξαιτίας αυτής της ιδιαιτερότητας αγνοεί τον χρόνο που χρειάζεται ο DMTCP για την εκτέλεση checkpointing.

Μετροπρόγραμμα	Dom0 Os	Dom0 with Dmtcp	Xen Vm	Xen VM με DMTCP
Basicmath small	2.912	3.064	3.059	3.255
Qsort small	0.245	0.316	0.254	0.366
Dijktra small	0.076	0.151	0.074	0.183
Patricia small	0.865	1.139	0.874	1.146
Basicmath large	152.206	183.685	164.782	213.141
Qsort large	3.472	5.226	4.119	6.038
Dijktra large	0.288	0.489	0.301	0.569
Patricia large	14.062	16.691	14.167	16.988

Πίνακας 7 Χρόνοι εκτέλεσης για τα μετροπρογράμματα miBench

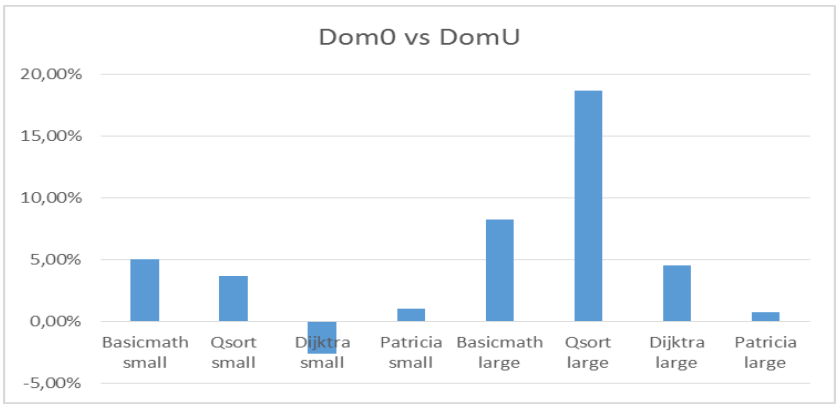
Bit counter	Dom0 times in seconds	Dom0 with Dmtcp times in seconds	Xen Vm times in seconds	Xen VM με DMTCP times in seconds	Bits
Optimized 1 bit/loop counter	1.749	1.941	1.802	2.223	534.224.958
Ratko's mystery algorithm	0.727	0.825	0.741	0.838	503.787.475
Recursive bit count by nybbles	1.701	1.928	1.624	2.067	538.082.424
Non-recursive bit count by nybbles	1.436	1.649	1.340	1.761	544.638.268
Non-recursive bit by bytes (BW)	0.728	0.978	0.743	1.074	518.870.054
Non-recursive bit count by byter (AR)	0.764	1.009	0.782	1.285	492.674.444
Shift and count bits	9.912	11.024	10.068	11.628	509.324.367

Πίνακας 8 Χρόνοι εκτέλεσης για το μετροπρόγραμμα mibench Bit counter

Μετροπρόγραμμα	Dom0 Os	Xen VM
BF SPEED	171.2uS per key	168.7uS per key

Πίνακας 9 Αποτελέσματα για το μετροπρόγραμμα mibench BFSPEED

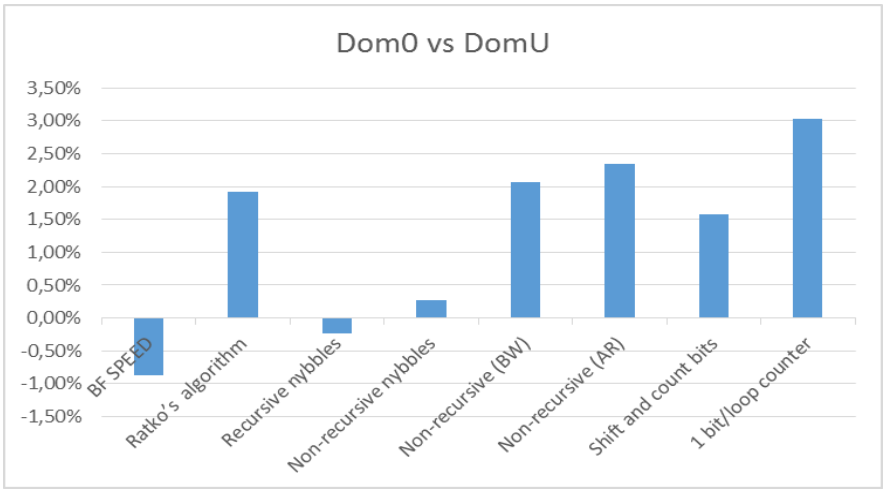
Από τα αποτελέσματα των πειραμάτων παρατηρούμε ότι εκείνα που περιλάμβαναν προσπέλαση του δίσκου έχουν σημαντική καθυστέρηση. Τα αποτελέσματα αυτά ήταν αναμενόμενα καθώς η προσπέλαση στον δίσκο απαιτεί την διαδικασία που αναλύθηκε στα προηγούμενα κεφάλαια. Η μέση καθυστέρηση που εισάγει η προσπέλαση στον δίσκο ανάμεσα σε Dom0 και DomU είναι 5%. Ενδιαφέρον παρουσιάζουν τα αποτελέσματα του Qsort καθώς παρατηρούμε ότι εμφανίζει την μεγαλύτερη καθυστέρηση. Όπως αναλύθηκε σε προηγούμενο κεφάλαιο ο Qsort κάνει εκτεταμένη χρήση του δίσκου και αυτό έχει σαν αποτέλεσμα την σημαντική επιβάρυνσή του.



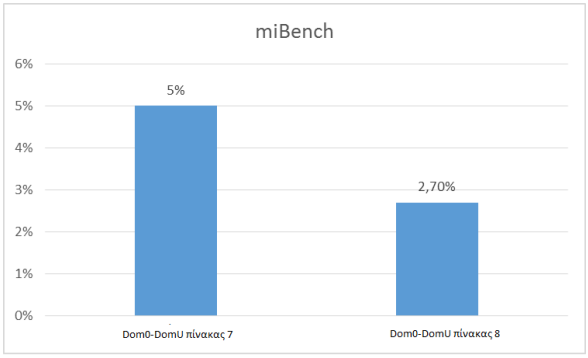
Εικόνα 13 Καθυστέρηση ανά εφαρμογή πίνακα 7 Dom0-DomU

Οι εφαρμογές του πίνακα 4 παίρνουν σαν είσοδο αρχεία που είναι αποθηκευμένα στον δίσκο και εξαιτίας αυτού παρατηρούμε χειρότερα αποτελέσματα στις περισσότερες περιπτώσεις.

Ακόμα πιο ξεκάθαρα είναι τα αποτελέσματα στους πίνακες 5 και 6.Τα συγκεκριμένα πειράματα δεν έχουν σαν είσοδο αρχεία και για τον λόγο αυτό τα αποτελέσματα σε Dom0 και DomU συγκλίνουν. Η μέση επιβάρυνση κυμαίνεται στο 2.7%.



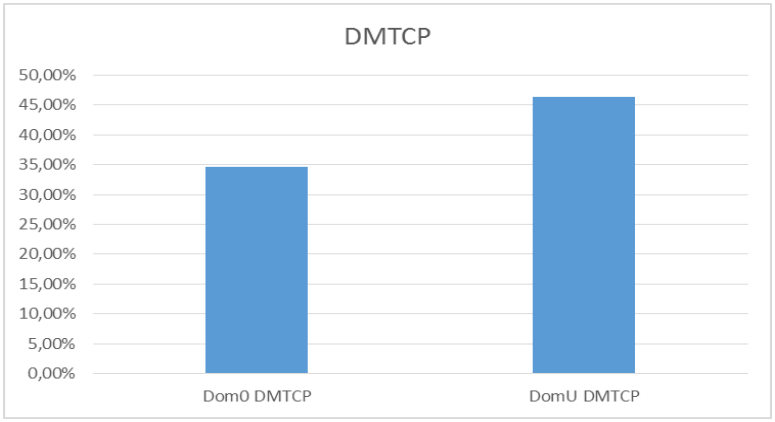
Εικόνα 14 Καθυστέρηση ανά εφαρμογή πίνακα 8-9 Dom0-DomU



Εικόνα 15 Dom0-DomU συνολικά

Τα αποτελέσματα όμως αλλάζουν ριζικά στο DMTCP καθώς όταν εκτελείται στον Dom0 εισάγει μέση επιβάρυνση 34.7% ενώ όταν εκτελείται στον DomU 46,3%. Η διαφορά ανάμεσα τα δυο συστήματα είναι σημαντική και ανέρχεται στο 11.6%. Η διαφορά αυτή είναι αποτέλεσμα δυο παραγόντων.

Αρχικά τα DMTCP κάνει συχνή χρήση κλήσεων συστήματος. Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο οι κλήσεις συστήματος είναι μια διαδικασία που εισάγει καθυστέρηση στο σύστημα. Επιπρόσθετα πέραν των κλήσεων συστήματος το DMTCP κάνει χρήση και του δίσκου καθώς το σημείο επαναφοράς που δημιουργεί το γράφει στον δίσκο. Ο συνδυασμός αυτών των δυο παραγόντων έχει σαν αποτέλεσμα την σημαντική αύξηση της καθυστέρησης.



Εικόνα 16 DMTCP

5.3 Συμπεράσματα πειραμάτων

Από τα πειράματα που παρουσιάστηκαν παραπάνω μπορεί να εξαχθούν χρήσιμα συμπεράσματα σχετικά με την συμπεριφορά των νέων επεξεργαστών της ARM σε περιβάλλον Xen. Σε πολλές περιπτώσεις παρατηρούμε ότι δεν υπάρχουν διαφορές ανάμεσα στα δυο περιβάλλοντα. Παρά το γεγονός ότι ο DomU δεν έχει απευθείας πρόσβαση στο υλικό τόσο του επεξεργαστή όσο και της μνήμης, φαίνεται ότι οι νέες τεχνολογίες διαχείρισης της εικονικής μνήμης και των εικονικών επεξεργαστών έχουν σχεδόν μηδενίσει την καθυστέρηση.

Η συμπεριφορά του συστήματος αλλάζει με την προσθήκη διεργασιών που εκτελούν κλήσεις συστήματος. Οι συγκεκριμένες διεργασίες απαιτούν αυξημένα δικαιώματα για να εκτελεστούν και για τον λόγο αυτό δημιουργούν καθυστέρηση. Η αδυναμία της εικονικής μηχανής να

εκτελέσει απευθείας τις κλήσεις συστήματος οφείλεται στην τεχνολογία των δακτυλίων που αναλύθηκε σε προηγούμενο κεφάλαιο.

Ένας ακόμα παράγοντας για την εισαγωγή καθυστέρησης στο σύστημα είναι και η προσπέλαση του δίσκου. Ο DomU δεν έχει δικαιώματα για άμεση προσπέλαση στον δίσκο. Ως εκ τούτου θα πρέπει να βασιστεί στον μηχανισμό διαχείρισης I/O ο οποίος εισάγει καθυστέρηση.

Η καθυστέρηση αυτή δεν φαίνεται να αποτελεί πρόβλημα για τις κλασσικές διεργασίες όμως σε περιπτώσεις εφαρμογών πραγματικού χρόνου πρέπει να λαμβάνεται σοβαρά υπόψη.

6 Συμπεράσματα

Η τεχνολογία της εικονικοποίησης παρέχει μια ιδανική πλατφόρμα για την ανάπτυξη διαφόρων εφαρμογών. Η ραγδαία ανάπτυξη του υλικού, σε συνδυασμό με την ολοένα και μεγαλύτερη ζήτηση για πολύπλοκα υπολογιστικά συστήματα, έχει συντελέσει στην ανάπτυξη νέων μεθόδων εικονικοποίησης. Στην παρούσα διατριβή αναλύθηκε η απόδοση του Xen που αποτελεί έναν από τους σημαντικότερους υπερεπόπτες για την δημιουργία εικονικών μηχανών. Ένα από τα βασικά εμπόδια που συναντά ο Xen είναι η ανάγκη ύπαρξης διαφόρων επιπέδων εικονικοποίησης ώστε να εξασφαλιστεί η ακεραιότητα και η ορθή λειτουργία του συστήματος.

Οι υποδομές του Xen υποστηρίζουν ένα ευρύ φάσμα τόσο υλικού όσο και λειτουργικών συστημάτων. Για την αύξηση της απόδοσης σε εικονικά περιβάλλοντα πρέπει να γίνει απαλοιφή των ενδιάμεσων επιπέδων εικονικοποίησης που προσδίδουν σημαντική επιβάρυνση στο σύστημα. Σε μεγάλο βαθμό η νέα γενιά επεξεργαστών της ARM στοχεύει στην απαλοιφή αυτών των ενδιάμεσων επιπέδων και παράλληλα διατηρεί όλα τα πλεονεκτήματα του εικονικού περιβάλλοντος, όπως η ευελιξία στην εκτέλεση των εφαρμογών, η απομόνωση και ευκολότερη διαχείρισή τους.

Οι νέοι αλγόριθμοι διαχείρισης της μνήμης και των επεξεργαστών του Xen σε συνδυασμό με το ειδικό υλικό που εισήγαγε η ARM έχουν κάνει τα εικονικά μηχανήματα του Xen on ARM να εκτελούνται σε χρόνους μη εικονικοποιημένων λειτουργικών συστημάτων.

όμως, παρά τις εξελίξεις αυτές η προσπέλαση των επιμέρους συσκευών, όπως ο δίσκος και η κάρτα δικτύου, αποτελούν ακόμα σημεία εισόδου καθυστέρησης στο σύστημα. Παρά τις προσπάθειες που έχουν γίνει τα τελευταία χρόνια για την απαλοιφή αυτών των προβλημάτων φαίνεται ότι ακόμα χρειάζονται σημαντικές βελτιώσεις ώστε να εξαλειφθούν. Για τον λόγο αυτό χρειάζεται ιδιαίτερη προσοχή στην ανάπτυξη εφαρμογών πραγματικού χρόνου ώστε να περιορίζεται η καθυστέρηση που προκαλούν τα I/O.

Επίσης, ένα σημαντικό εμπόδιο που πρέπει να ξεπεραστεί είναι η συμβατότητα των κινητών συσκευών με τον Xen, εξαιτίας της πολυπλοκότητας των περιφερειακών συσκευών που χρησιμοποιούν.

Συνοψίζοντας, η νέα τεχνολογία Xen on ARM φαίνεται ότι έχει κάνει σημαντική πρόοδο στην απόδοση των εικονικών μηχανών εξαλείφοντας σε μεγάλο βαθμό τις καθυστερήσεις του συστήματος. Επιπρόσθετα, παρά την πολυπλοκότητα της νέας αρχιτεκτονικής διατηρείται η δυνατότητα χρήσης διαφόρων λειτουργικών συστημάτων. Σε μελλοντική εργασία, θα μπορούσε να γίνει ανάλυση της συμπεριφοράς του Xen on ARM και σε λειτουργικά συστήματα πραγματικού χρόνου (Real-Time Operating Systems, RTOS). Επίσης, θα μπορούσε να γίνει ανάπτυξη εφαρμογών πραγματικού χρόνου, ώστε να μελετηθεί η βιωσιμότητα τους σε εικονικό περιβάλλον εξαιτίας των καθυστερήσεων που εισάγονται στο σύστημα.

Παράρτημα Α

```
## Executing script at 43100000
reading /xen
688912 bytes read in 58 ms (11.3 MiB/s)
reading /sun7i-a20-cubieboard2.dtb
20757 bytes read in 29 ms (698.2 KiB/s)
reading /vmlinuz
5254960 bytes read in 274 ms (18.3 MiB/s)
Kernel image @ 0x6ea00000 [ 0x000000 - 0x0ef700 ]
## Flattened Device Tree blob at 7ec00000
   Booting using the fdt blob at 0x7ec00000
   reserving fdt memory region: addr=7ec00000 size=6000
   Using Device Tree in place at 7ec00000, end 7ec08fff
Starting kernel ...
- UART enabled -
- CPU 00000000 booting -
- Xen starting in Hyp mode -
- Zero BSS -
- Setting up control registers -
- Turning on paging -
- Ready -
Checking for initrd in /chosen
RAM: 0000000040000000 - 000000007fffffff
MODULE[1]: 000000007ec00000 - 000000007ec06000
MODULE[2]: 000000006ee00000 - 000000006f302f30 console=hvc0 ro root=/dev/mmcblk0p2
     rootwait clk_ignore_unused
 RESVD[0]: 000000007ec00000 - 000000007ec06000
Command line: console=dtuart dtuart=/soc@01c00000/serial@01c28000
     dom0_mem=512M,max:512M
Placing Xen at 0x000000007fe00000-0x0000000080000000
Xen heap: 0000000076000000-000000007e000000 (32768 pages)
Dom heap: 229376 pages
Looking for UART console /soc@01c00000/serial@01c28000
 Xen 4.4.1-rc1
(XEN) Xen version 4.4.1-rc1 (sasakis@) (arm-linux-gnueabihf-gcc (Ubuntu/Linaro 4.8.2-
     16ubuntu4) 4.8.2) debug=y Tue Jan 19 17:15:23 EET 2016
(XEN) Latest ChangeSet: Fri Aug 1 18:59:35 2014 +0100 git:a63eb05
(XEN) Processor: 410fc074: "ARM Limited", variant: 0x0, part 0xc07, rev 0x4
(XEN) 32-bit Execution:
(XEN)   Processor Features: 00001131:00011011
(XEN)     Instruction Sets: AArch32 Thumb Thumb-2 ThumbEE Jazelle
(XEN)     Extensions: GenericTimer Security
(XEN)   Debug Features: 02010555
(XEN)   Auxiliary Features: 00000000
(XEN)   Memory Model Features: 10101105 40000000 01240000 02102211
```

```
(XEN)  ISA Features: 02101110 13112111 21232041 11112131 10011142 00000000
(XEN) Platform: Allwinner A20
(XEN) Using PSCI for SMP bringup
(XEN) Generic Timer IRQ: phys=30 hyp=26 virt=27
(XEN) Using generic timer at 24000 KHz
(XEN) GIC initialization:
(XEN)        gic_dist_addr=0000000001c81000
(XEN)        gic_cpu_addr=0000000001c82000
(XEN)        gic_hyp_addr=0000000001c84000
(XEN)        gic_vcpu_addr=0000000001c86000
(XEN)        gic_maintenance_irq=25
(XEN) GIC: 160 lines, 2 cpus, secure (IID 0100143b).
(XEN) Using scheduler: SMP Credit Scheduler (credit)
(XEN) Allocated console ring of 16 KiB.
(XEN) VFP implementer 0x41 architecture 2 part 0x30 variant 0x7 rev 0x4
(XEN) Bringing up CPU1
- CPU 00000001 booting -
- Xen starting in Hyp mode -
- Setting up control registers -
- Turning on paging -
- Ready -
(XEN) CPU 1 booted.
(XEN) Brought up 2 CPUs
(XEN) *** LOADING DOMAIN 0 ***
(XEN) Populate P2M 0x40000000->0x60000000 (1:1 mapping for dom0)
(XEN) Loading kernel from boot module 2
(XEN) Loading zImage from 000000006ee00000 to 0000000047a00000-0000000047f02f30
(XEN) Loading dom0 DTB to 0x0000000048000000-0x0000000048004ce9
(XEN) Scrubbing Free RAM: .....done.
(XEN) Initial low memory virq threshold set at 0x4000 pages.
(XEN) Std. Loglevel: All
(XEN) Guest Loglevel: All
(XEN) *** Serial input -> DOM0 (type 'CTRL-a' three times to switch input to Xen)
(XEN) Freed 264kB init memory.
```

Παράρτημα Β

Basicmath large

```
#include "snipmath.h"
#include <math.h>
int main(void)
{
  double  a1 = 1.0, b1 = -10.5, c1 = 32.0, d1 = -30.0;
  double  x[3];
  double X;
  int     solutions;
  int i;
  unsigWned long l = 0x3fed0169L;
  struct int_sqrt q;
  long n = 0;
  printf("********* CUBIC FUNCTIONS **********\n");
  SolveCubic(a1, b1, c1, d1, &solutions, x);
  printf("Solutions:");
  for(i=0;i<solutions;i++)
    printf(" %f",x[i]);
  printf("\n");
  a1 = 1.0; b1 = -4.5; c1 = 17.0; d1 = -30.0;
  SolveCubic(a1, b1, c1, d1, &solutions, x);
  printf("Solutions:");
  for(i=0;i<solutions;i++)
    printf(" %f",x[i]);
  printf("\n");
  a1 = 1.0; b1 = -3.5; c1 = 22.0; d1 = -31.0;
  SolveCubic(a1, b1, c1, d1, &solutions, x);
  printf("Solutions:");
  for(i=0;i<solutions;i++)
    printf(" %f",x[i]);
  printf("\n");
  a1 = 1.0; b1 = -13.7; c1 = 1.0; d1 = -35.0;
  SolveCubic(a1, b1, c1, d1, &solutions, x);
  printf("Solutions:");
  for(i=0;i<solutions;i++)
    printf(" %f",x[i]);
  printf("\n");
  a1 = 3.0; b1 = 12.34; c1 = 5.0; d1 = 12.0;
  SolveCubic(a1, b1, c1, d1, &solutions, x);
  printf("Solutions:");
  for(i=0;i<solutions;i++)
    printf(" %f",x[i]);
```

```
 printf("\n");
 a1 = -8.0; b1 = -67.89; c1 = 6.0; d1 = -23.6;
 SolveCubic(a1, b1, c1, d1, &solutions, x);
 printf("Solutions:");
 for(i=0;i<solutions;i++)
   printf(" %f",x[i]);
 printf("\n");
 a1 = 45.0; b1 = 8.67; c1 = 7.5; d1 = 34.0;
 SolveCubic(a1, b1, c1, d1, &solutions, x);
 printf("Solutions:");
 for(i=0;i<solutions;i++)
   printf(" %f",x[i]);
 printf("\n");
 a1 = -12.0; b1 = -1.7; c1 = 5.3; d1 = 16.0;
 SolveCubic(a1, b1, c1, d1, &solutions, x);
 printf("Solutions:");
 for(i=0;i<solutions;i++)
   printf(" %f",x[i]);
 printf("\n");
 for(a1=1;a1<10;a1+=1) {
   for(b1=10;b1>0;b1-=.25) {
     for(c1=5;c1<15;c1+=0.61) {
        for(d1=-1;d1>-5;d1-=.451) {
           SolveCubic(a1, b1, c1, d1, &solutions, x);
           printf("Solutions:");
           for(i=0;i<solutions;i++)
             printf(" %f",x[i]);
           printf("\n");
        }
     }
   }
 }
 printf("********* INTEGER SQR ROOTS ***********\n");
 for (i = 0; i < 100000; i+=2)
   {
     usqrt(i, &q);
    // printf("sqrt(%3d) = %2d, remainder = %2d\n",
    printf("sqrt(%3d) = %2d\n",
          i, q.sqrt);
   }
printf("\n");
 for (l = 0x3fed0169L; l < 0x3fed4169L; l++)
   {
      usqrt(l, &q);
```

```
      //printf("\nsqrt(%lX) = %X, remainder = %X\n", l, q.sqrt, q.frac);
      printf("sqrt(%lX) = %X\n", l, q.sqrt);
    }
  printf("********* ANGLE CONVERSION **********\n");
  /* convert some rads to degrees */
/*   for (X = 0.0; X <= 360.0; X += 1.0) */
  for (X = 0.0; X <= 360.0; X += .001)
    printf("%3.0f degrees = %.12f radians\n", X, deg2rad(X));
  puts("");
/*   for (X = 0.0; X <= (2 * PI + 1e-6); X += (PI / 180)) */
  for (X = 0.0; X <= (2 * PI + 1e-6); X += (PI / 5760))
    printf("%.12f radians = %3.0f degrees\n", X, rad2deg(X));
  return 0;
}
```

Basicmath small

```
#include "snipmath.h"
#include <math.h>
int main(void)
{
  double  a1 = 1.0, b1 = -10.5, c1 = 32.0, d1 = -30.0;
  double  a2 = 1.0, b2 = -4.5, c2 = 17.0, d2 = -30.0;
  double  a3 = 1.0, b3 = -3.5, c3 = 22.0, d3 = -31.0;
  double  a4 = 1.0, b4 = -13.7, c4 = 1.0, d4 = -35.0;
  double  x[3];
  double X;
  int     solutions;
  int i;
  unsigned long l = 0x3fed0169L;
  struct int_sqrt q;
  long n = 0;
  printf("********* CUBIC FUNCTIONS **********\n");
  SolveCubic(a1, b1, c1, d1, &solutions, x);
  printf("Solutions:");
  for(i=0;i<solutions;i++)
    printf(" %f",x[i]);
  printf("\n");
  SolveCubic(a2, b2, c2, d2, &solutions, x);
  printf("Solutions:");
```

```
for(i=0;i<solutions;i++)
  printf(" %f",x[i]);
printf("\n");
SolveCubic(a3, b3, c3, d3, &solutions, x);
printf("Solutions:");
for(i=0;i<solutions;i++)
  printf(" %f",x[i]);
printf("\n");
SolveCubic(a4, b4, c4, d4, &solutions, x);
printf("Solutions:");
for(i=0;i<solutions;i++)
  printf(" %f",x[i]);
printf("\n");
for(a1=1;a1<10;a1++) {
  for(b1=10;b1>0;b1--) {
    for(c1=5;c1<15;c1+=0.5) {
      for(d1=-1;d1>-11;d1--) {
        SolveCubic(a1, b1, c1, d1, &solutions, x);
        printf("Solutions:");
        for(i=0;i<solutions;i++)
          printf(" %f",x[i]);
        printf("\n");
      } } } }
printf("********* INTEGER SQR ROOTS ***********\n");
/* perform some integer square roots */
for (i = 0; i < 1001; ++i)
  {
    usqrt(i, &q);
   printf("sqrt(%3d) = %2d\n",
          i, q.sqrt);
  }
usqrt(l, &q);
//printf("\nsqrt(%lX) = %X, remainder = %X\n", l, q.sqrt, q.frac);
printf("\nsqrt(%lX) = %X\n", l, q.sqrt);
printf("********* ANGLE CONVERSION ***********\n");
for (X = 0.0; X <= 360.0; X += 1.0)
  printf("%3.0f degrees = %.12f radians\n", X, deg2rad(X));
```

```
    puts("");
    for (X = 0.0; X <= (2 * PI + 1e-6); X += (PI / 180))
      printf("%.12f radians = %3.0f degrees\n", X, rad2deg(X));
    return 0;
  }
```

Qsort large

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define UNLIMIT
#define MAXARRAY 60000 /* this number, if too large, will cause a seg. fault!! */
struct my3DVertexStruct {
  int x, y, z;
  double distance;
};
int compare(const void *elem1, const void *elem2){
  /* D = [(x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2]^(1/2) */
  /* sort based on distances from the origin... */
  double distance1, distance2;
  distance1 = (*((struct my3DVertexStruct *)elem1)).distance;
  distance2 = (*((struct my3DVertexStruct *)elem2)).distance;
  return (distance1 > distance2) ? 1 : ((distance1 == distance2) ? 0 : -1);
}
int
main(int argc, char *argv[]) {
  struct my3DVertexStruct array[MAXARRAY];
  FILE *fp;
  int i,count=0;
  int x, y, z;
  if (argc<2) {
    fprintf(stderr,"Usage: qsort_large <file>\n");
    exit(-1);
  }
  else {
    fp = fopen(argv[1],"r");
```

```
   while((fscanf(fp, "%d", &x) == 1) && (fscanf(fp, "%d", &y) == 1) && (fscanf(fp, "%d", &z) == 1)
&&  (count < MAXARRAY)) {
            array[count].x = x;
            array[count].y = y;
            array[count].z = z;
            array[count].distance = sqrt(pow(x, 2) + pow(y, 2) + pow(z, 2));
            count++;    }  }
  printf("\nSorting %d vectors based on distance from the origin.\n\n",count);
  qsort(array,count,sizeof(struct my3DVertexStruct),compare);
  for(i=0;i<count;i++)
    printf("%d %d %d\n", array[i].x, array[i].y, array[i].z);
  return 0;
}
```

Qsort small

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define UNLIMIT
#define MAXARRAY 60000 /* this number, if too large, will cause a seg. fault!! */
struct myStringStruct {
  char qstring[128];
};
int compare(const void *elem1, const void *elem2)
{
  int result;
  result = strcmp((*((struct myStringStruct *)elem1)).qstring, (*((struct myStringStruct
*)elem2)).qstring);
  return (result < 0) ? 1 : ((result == 0) ? 0 : -1);
}
int
main(int argc, char *argv[]) {
  struct myStringStruct array[MAXARRAY];
  FILE *fp;
  int i,count=0;
  if (argc<2) {
    fprintf(stderr,"Usage: qsort_small <file>\n");
```

```
    exit(-1);
  }
  else {
    fp = fopen(argv[1],"r");

    while((fscanf(fp, "%s", &array[count].qstring) == 1) && (count < MAXARRAY)) {
            count++;
    }
  }
  printf("\nSorting %d elements.\n\n",count);
  qsort(array,count,sizeof(struct myStringStruct),compare);
  for(i=0;i<count;i++)
    printf("%s\n", array[i].qstring);
  return 0;
}
```

Patricia

```
#include <stdlib.h>        /* free(), malloc() */
#include <string.h>        /* bcopy() */
#include "patricia.h"
static __inline
unsigned long
bit(int i, unsigned long key)
{
        return key & (1 << (31-i));
}
static int
pat_count(struct ptree *t, int b)
{
        int count;

        if (t->p_b <= b) return 0;
        count = t->p_mlen;
        count += pat_count(t->p_left,  t->p_b);
        count += pat_count(t->p_right, t->p_b);
        return count;
```

```
}
static struct ptree *
insertR(struct ptree *h, struct ptree *n, int d, struct ptree *p)
{
	if ((h->p_b >= d) || (h->p_b <= p->p_b)) {
		n->p_b = d;
		n->p_left = bit(d, n->p_key) ? h : n;
		n->p_right = bit(d, n->p_key) ? n : h;
		return n;
	}

	if (bit(h->p_b, n->p_key))
		h->p_right = insertR(h->p_right, n, d, h);
	else
		h->p_left = insertR(h->p_left, n, d, h);
	return h;
}
struct ptree *
pat_insert(struct ptree *n, struct ptree *head)
{
	struct ptree *t;
	struct ptree_mask *buf, *pm;
	int i, copied;

	if (!head || !n || !n->p_m)
		return 0;
	n->p_key &= n->p_m->pm_mask;
	t = head;
	do {
		i = t->p_b;
		t = bit(t->p_b, n->p_key) ? t->p_right : t->p_left;
	} while (i < t->p_b);
	if (n->p_key == t->p_key) {
		/*
		 * If we have a duplicate mask, replace the entry
		 * with the new one.
		 */
```

```
		for (i=0; i < t->p_mlen; i++) {
			if (n->p_m->pm_mask == t->p_m[i].pm_mask) {
				t->p_m[i].pm_data = n->p_m->pm_data;
				free(n->p_m);
				free(n);
				n = 0;
				return t;
			}
		}
			buf = (struct ptree_mask *)malloc(
		    sizeof(struct ptree_mask)*(t->p_mlen+1));

		copied = 0;
		for (i=0, pm=buf; i < t->p_mlen; pm++) {
			if (n->p_m->pm_mask > t->p_m[i].pm_mask) {
				bcopy(t->p_m + i, pm, sizeof(struct ptree_mask));
				i++;
			}
			else {
				bcopy(n->p_m, pm, sizeof(struct ptree_mask));
				n->p_m->pm_mask = 0xffffffff;
				copied = 1;
			}
		}
		if (!copied) {
			bcopy(n->p_m, pm, sizeof(struct ptree_mask));
		}
		free(n->p_m);
		free(n);
		n = 0;
		t->p_mlen++;
		free(t->p_m);
		t->p_m = buf;
		return t;
	}
	for (i=1; i < 32 && bit(i, n->p_key) == bit(i, t->p_key); i++);
	if (bit(head->p_b, n->p_key))
```

```
                head->p_right = insertR(head->p_right, n, i, head);
        else
                head->p_left = insertR(head->p_left, n, i, head);

        return n;
}
int
pat_remove(struct ptree *n, struct ptree *head)
{
        struct ptree *p, *g, *pt, *pp, *t;
        struct ptree_mask *buf, *pm;
        int i;
        if (!n || !n->p_m || !t)
                return 0;
        g = p = t = head;
        do {
                i = t->p_b;
                g = p;
                p = t;
                t = bit(t->p_b, n->p_key) ? t->p_right : t->p_left;
        } while (i < t->p_b);
        if (t->p_key != n->p_key)
                return 0;
        if (t->p_mlen == 1) {
        if (t->p_b == 0)
                        return 0;
                if (t->p_m->pm_mask != n->p_m->pm_mask)
                        return 0;

                pp = pt = p;
                do {
                        i = pt->p_b;
                        pp = pt;
                        pt = bit(pt->p_b, p->p_key) ? pt->p_right : pt->p_left;
                } while (i < pt->p_b);

                if (bit(pp->p_b, p->p_key))
```

```
                pp->p_right = t;
        else
                pp->p_left = t;
        if (bit(g->p_b, n->p_key))
                g->p_right = bit(p->p_b, n->p_key) ?
                        p->p_left : p->p_right;
        else
                g->p_left = bit(p->p_b, n->p_key) ?
                        p->p_left : p->p_right;

                        if (t->p_m->pm_data)
                free(t->p_m->pm_data);
        free(t->p_m);
        if (t != p) {
                t->p_key = p->p_key;
                t->p_m = p->p_m;
                t->p_mlen = p->p_mlen;
        }
        free(p);
        return 1;
}
        for (i=0; i < t->p_mlen; i++)
        if (n->p_m->pm_mask == t->p_m[i].pm_mask)
                break;
if (i >= t->p_mlen)
        return 0;

buf = (struct ptree_mask *)malloc(
     sizeof(struct ptree_mask)*(t->p_mlen-1));

for (i=0, pm=buf; i < t->p_mlen; i++) {
        if (n->p_m->pm_mask != t->p_m[i].pm_mask) {
                bcopy(t->p_m + i, pm++, sizeof(struct ptree_mask));
        }
}
t->p_mlen--;
free(t->p_m);
```

```
	t->p_m = buf;
	return 1;
}
struct ptree *
pat_search(unsigned long key, struct ptree *head)
{
	struct ptree *p = 0, *t = head;
	int i;

	if (!t)
		return 0;

	do {

		if (t->p_key == (key & t->p_m->pm_mask)) {
			p = t;
		}

		i = t->p_b;
		t = bit(t->p_b, key) ? t->p_right : t->p_left;
	} while (i < t->p_b);
	return (t->p_key == (key & t->p_m->pm_mask)) ? t : p;
}
```

Dijkstra

```
#include <stdio.h>
#define NUM_NODES		100
#define NONE			9999
struct _NODE
{
  int iDist;
  int iPrev;
};
typedef struct _NODE NODE;
struct _QITEM
{
```

```
  int iNode;
  int iDist;
  int iPrev;
  struct _QITEM *qNext;
};
typedef struct _QITEM QITEM;
QITEM *qHead = NULL;
int AdjMatrix[NUM_NODES][NUM_NODES];
int g_qCount = 0;
NODE rgnNodes[NUM_NODES];
int ch;
int iPrev, iNode;
int i, iCost, iDist;
void print_path (NODE *rgnNodes, int chNode)
{
  if (rgnNodes[chNode].iPrev != NONE)
    {
      print_path(rgnNodes, rgnNodes[chNode].iPrev);
    }
  printf (" %d", chNode);
  fflush(stdout);
}
void enqueue (int iNode, int iDist, int iPrev)
{
  QITEM *qNew = (QITEM *) malloc(sizeof(QITEM));
  QITEM *qLast = qHead;
  if (!qNew)
    {
      fprintf(stderr, "Out of memory.\n");
      exit(1);
    }
  qNew->iNode = iNode;
  qNew->iDist = iDist;
  qNew->iPrev = iPrev;
  qNew->qNext = NULL;
  if (!qLast)
    {
```

```
      qHead = qNew;
    }
  else
    {
      while (qLast->qNext) qLast = qLast->qNext;
      qLast->qNext = qNew;
    }
  g_qCount++;
  //              ASSERT(g_qCount);
}
void dequeue (int *piNode, int *piDist, int *piPrev)
{
  QITEM *qKill = qHead;
  if (qHead)
    {
      //              ASSERT(g_qCount);
      *piNode = qHead->iNode;
      *piDist = qHead->iDist;
      *piPrev = qHead->iPrev;
      qHead = qHead->qNext;
      free(qKill);
      g_qCount--;
    }
}
int qcount (void)
{
  return(g_qCount);
}
int dijkstra(int chStart, int chEnd)
{
  for (ch = 0; ch < NUM_NODES; ch++)
    {
      rgnNodes[ch].iDist = NONE;
      rgnNodes[ch].iPrev = NONE;
    }
  if (chStart == chEnd)
    {
```

```
      printf("Shortest path is 0 in cost. Just stay where you are.\n");
    }
  else
    {
      rgnNodes[chStart].iDist = 0;
      rgnNodes[chStart].iPrev = NONE;
          enqueue (chStart, 0, NONE);
     while (qcount() > 0)
           {
             dequeue (&iNode, &iDist, &iPrev);
             for (i = 0; i < NUM_NODES; i++)
              {
                if ((iCost = AdjMatrix[iNode][i]) != NONE)
                     {
                       if ((NONE == rgnNodes[i].iDist) ||
                          (rgnNodes[i].iDist > (iCost + iDist)))
                        {
                          rgnNodes[i].iDist = iDist + iCost;
                          rgnNodes[i].iPrev = iNode;
                          enqueue (i, iDist + iCost, iNode);
                        }
                     }
              }
           }
      printf("Shortest path is %d in cost. ", rgnNodes[chEnd].iDist);
      printf("Path is: ");
      print_path(rgnNodes, chEnd);
      printf("\n");
    }
}
int main(int argc, char *argv[]) {
  int i,j,k;
  FILE *fp;
  if (argc<2) {
    fprintf(stderr, "Usage: dijkstra <filename>\n");
    fprintf(stderr, "Only supports matrix size is #define'd.\n");
  }
```

```
 /* open the adjacency matrix file */
 fp = fopen (argv[1],"r");
 /* make a fully connected matrix */
 for (i=0;i<NUM_NODES;i++) {
   for (j=0;j<NUM_NODES;j++) {
     /* make it more sparce */
     fscanf(fp,"%d",&k);
                                AdjMatrix[i][j]= k;
   }
 }
 /* finds 10 shortest paths between nodes */
 for (i=0,j=NUM_NODES/2;i<100;i++,j++) {
                                j=j%NUM_NODES;
     dijkstra(i,j);
 }
 exit(0);
```

Βιβλιογραφία

[1] Xen. Xen Project Software Overview. [Ηλεκτρονικό] http://wiki.xen.org/wiki/Xen_Overview#Introduction_to_Xen_Architecture.

[2] —. Xen ARM with Virtualization Extensions whitepaper. [Ηλεκτρονικό] http://wiki.xen.org/wiki/Xen_ARM_with_Virtualization_Extensions_whitepaper.

[3] Brash, David. Extensions to the ARMv7-A Architecture. Architecture Program Manager, ARM Ltd. Hotchips : Arm, 2010.

[4] Larry McVoy, Carl Staelin. lmbench: Portable Tools for Performance Analysis. San Diego, California : USENIX Annual Technical Conference, 1996.

[5] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst,. MiBench: A free, commercially representative embedded benchmark suite. Michigan : The University of Michigan, December 2001, In IEEE 4th Annual Workshop on Workload Characterization, 2001.

[6] Jason Ansel, Kapil Arya and Gene Cooperman. DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop. Rome, Italy : 23rd IEEE International Parallel and Distributed Processing Symposium, 2009.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, et al., "Xen and the art of virtualization", *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[8] J. Hwang, S. Suh, S. Heo, C. Park, J. Ryu, S. Park, C. Kim, "Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones," In Proceedings of the 5th Annual IEEE Consumer Communications&Networking Conference, USA, January 2008.

[9] Ludmila Cherkasova , Rob Gardner, Measuring CPU overhead for I/O processing in the Xen virtual machine monitor, Proceedings of the annual conference on USENIX Annual Technical Conference,April 10-15, 2005, Anaheim, CA

[10] Roberto Mijat and Andy Nightingale, Virtualization is Coming to a Platform Near You: The ARM Architecture Virtualization Extensions and the importance of System MMU for virtualized solutions and beyond, ARM White paper, 2011

[11] John Goodacre, Hardware accelerated Virtualization in the ARM Cortex Processors, XenSummit Asia 2011

[12] David Brash, Extensions to the ARMv7-A Architecture, HotChips 2010