Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Αποδοτικός διαμοιρασμός αρχείων μεταξύ host και unikernel

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φώτιος Ζαφείρης Μ. Ξενάκης

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2020

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Αποδοτικός διαμοιρασμός αρχείων μεταξύ host και unikernel

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φώτιος Ζαφείρης Μ. Ξενάκης

Επιβλέπων: Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Οκτωβρίου 2020.

.....	.....	.....
Νεκτάριος Κοζύρης	Γεώργιος Γκούμας	Διονύσιος Πνευματικάτος
Καθηγητής Ε.Μ.Π.	Επίκουρος καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2020

.....

Φώτιος Ζαφείρης Μ. Ξενάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το cloud computing είναι η κυρίαρχη προσέγγιση προς την υπολογιστική υποδομή, βασιζόμενο στην τεχνολογία της εικονικοποίησης (virtualization). Καθώς το cloud επεκτείνεται, καθίσταται επιτακτική η αποδοτική χρήση των υπολογιστικών του πόρων από το λογισμικό. Προς αυτό τον σκοπό, μια λύση είναι τα *unikernels*, πυρήνες λειτουργικών συστημάτων εξειδικευμένοι για να τρέχουν μία μόνο εφαρμογή, εξοικονομώντας πόρους σε σχέση με έναν γενικού σκοπού πυρήνα. Η αποδοτική πρόσβαση των virtualized guests στους πόρους του host συστήματος είναι μια μεγάλη πρόκληση για την εικονικοποίηση. Σε αυτόν τον τομέα, σημαντική συνεισφορά αποτελεί το *virtio*, μια προδιαγραφή paravirtualized συσκευών για την αποδοτική χρήση των πόρων του host. Για το διαμοιρασμό αρχείων ανάμεσα σε host και guest έχει προταθεί το *virtio-fs*, μια virtio συσκευή που προσφέρει στον guest πρόσβαση σε έναν κατάλογο του συστήματος αρχείων του host, παρέχοντας υψηλές επιδόσεις και σημασιολογία τοπικού συστήματος αρχείων.

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η υλοποίηση και αξιολόγηση της χρήσης του virtio-fs σε ένα unikernel, συγκεκριμένα στο OSv. Δείχνουμε ότι ο συνδυασμός αυτών των δύο έχει σημαντικά πλεονεκτήματα, τόσο από άποψη επιδόσεων, όπου επιτυγχάνονται αποτελέσματα συγκρίσιμα με τα τοπικά συστήματα αρχείων, όσο και από διαχειριστική άποψη στο πλαίσιο του cloud. Επίσης, τα παραπάνω γίνονται σε πλήρη ενσωμάτωση με το έργο ανοιχτού λογισμικού του unikernel στο οποίο βασιστήκαμε. Έτσι, το προϊόν της εργασίας αποκτά πρακτική αξία, καθώς αποτελεί χρήσιμη συνεισφορά στο έργο, επιτυγχάνοντας έτσι έναν κεντρικό, μη τεχνικό, στόχο της. Ταυτόχρονα, εξερευνούμε τον τρόπο λειτουργίας των έργων ανοιχτού λογισμικού και των κοινοτήτων που σχηματίζονται γύρω τους, καθώς γινόμαστε ενεργά μέλη μίας από αυτές.

Λέξεις κλειδιά

εικονικοποίηση, νέφος, σύστημα αρχείων, unikernel, virtio, OSv, virtio-fs, QEMU

Abstract

Cloud computing is the dominant approach to compute infrastructure, established on the technology of virtualization. As the cloud expands, efficient utilization of its compute resources by software becomes imperative. One solution towards that are *unikernels*, operating system kernels specialized to run a single application, sparing resources compared to a general-purpose kernel. Efficient access from virtualized guests to the underlying host's resources is a substantial challenge in virtualization. In this aspect, *virtio* has been a significant contribution, as a specification of paravirtual devices enabling efficient usage of the host's resources. For host-guest file sharing, *virtio-fs* has been proposed, as a virtio device offering guest access to a file system directory on the host, providing high performance and local file system semantics.

This thesis is concerned with the implementation and evaluation of virtio-fs in the context of the OSvunikernel. We demonstrate that combining the two offers great benefits, both with regard to performance achieved, which is comparable to local file systems, and the operational aspect in a cloud context. Moreover, the above are carried out fully within the open-source project behind the unikernel we based our work on. This way, the resulting product gains practical value, being a useful contribution to the project, thus achieving a pivotal, non-technical goal. Furthermore, we explore how open-source software projects and the communities around them work, as we become active members of one.

Keywords

virtualization, cloud, file system, unikernel, virtio, OSv, virtio-fs, QEMU

Ευχαριστίες

Για την παρούσα εργασία, που σηματοδοτεί την ολοκλήρωση μίας πορείας ετών, θα ήθελα να ευχαριστήσω τα μέλη του εργαστηρίου υπολογιστικών συστημάτων, υπό την αιγίδα του οποίου πραγματοποιήθηκε. Πιο πολύ όμως θα ήθελα να τους ευχαριστήσω, όπως και άλλα μέλη της σχολής ΗΜΜΥ, για τη διδασκαλία, το γνήσιο ενδιαφέρον και την καλλιέργεια της κουλτούρας του μηχανικού που μου προσέφεραν.

Επίσης, οφείλω ένα μεγάλο ευχαριστώ στους ανθρώπους των κοινοτήτων του OSv και του virtio-fs για την υποστήριξη, τις συμβουλές και το χρόνο τους, αλλά κυρίως για την ανοικτότητα, το ήθος και το έργο τους, που ενέπνευσαν αυτή τη συνεισφορά.

Τέλος, αυτοί για τους οποίους είμαι περισσότερο ευγνώμων είναι η οικογένεια και οι φίλοι μου, που πάντα βρίσκονται στο πλευρό μου, με ανέχονται και με στηρίζουν και χωρίς τους οποίους τίποτα δεν θα μπορούσε να επιτευχθεί.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

1.1.1 Γιατί unikernel

Η τελευταία δεκαετία έχει χαρακτηριστεί από την κυριαρχία του μοντέλου του υπολογιστικού νέφους (cloud computing) στην υπολογιστική υποδομή. Αυτό έχει εξελιχθεί σε θεμέλιο λίθο σε πολλά πεδία εφαρμογών, κάνοντας πρακτικές αρχιτεκτονικές που μέχρι πρότινος ήταν αδύνατες, για ένα ολοένα αυξανόμενο πλήθος χρηστών. Επίσης, η ευρύτατη υιοθέτηση του έχει οδηγήσει σε διαχείριση υπολογιστικών πόρων πρωτοφανούς κλίμακας. Η τεχνολογία που καθιστά δυνατό το cloud στην παρούσα του μορφή είναι αυτή της εικονικοποίησης: της κατά βούληση 'σμίλευσης' πολλαπλών εικονικών μηχανών (guests) από ένα φυσικό σύστημα (host).

Παρά τις μεγάλες αλλαγές που έχει φέρει η έλευση του cloud, τα παραδοσιακά κομμάτια της στοίβας λογισμικού, με κύριο εκπρόσωπο το λειτουργικό σύστημα, έχουν επηρεαστεί ελάχιστα από αυτήν. Έτσι, σε αυτή την πλευρά κυριαρχούν τα λειτουργικά συστήματα γενικού σκοπού όπως το Linux, με σχεδιαστικές επιλογές και κληρονομιά δεκαετιών. Το γεγονός αυτό είναι αδιαμφισβήτητα ενδεικτικό της υψηλής δυσκολίας που έχει η υλοποίηση τους, αλλά και της αξίας που έχουν τα παρόντα συστήματα, κατόπιν συσσώρευσης μακράς αλυσίδας βελτιώσεων.

Ένας συνδυασμός παραγόντων όμως καθιστά πλέον εμφανές ότι τα γενικού σκοπού λειτουργικά συστήματα δεν είναι η βέλτιστη λύση για τις ανάγκες των σύγχρονων εφαρμογών. Σε αυτούς τους παράγοντες συγκαταλέγεται η μετάβαση από φυσικά σε εικονικά μηχανήματα με την ακύρωση της υπόθεσης της αποκλειστικής χρήσης των πόρων που αυτή συνεπάγεται. Επίσης, το ταχύ κλείσιμο του άλλοτε χάσματος ανάμεσα στις επιδόσεις I/O και επεξεργασίας, που καθιστά σχεδόν απαγορευτική την εμπλοκή του λειτουργικού συστήματος στο μονοπάτι δεδομένων (data path) μίας εφαρμογής που απαιτεί ύψιστες επιδόσεις, εξ΄ ου και η δημοφιλία των [3, 9]. Τέλος, η προαναφερθείσα κλίμακα των υποδομών, που ενισχύει την ανάγκη για βελτιστοποίηση της αποδοτικότητας, καθώς η υποβέλτιστη λειτουργία έχει μεγάλο κόστος.

Τα unikernels είναι μία αξιόλογη πρόταση για αντικατάσταση των συμβατικών λειτουργικών συστημάτων στους guests, όταν αυτά χρησιμοποιούνται για την εκτέλεση μίας μόνο εφαρμογής. Προκύπτουν από τη συγχώνευση μίας εφαρμογής μαζί με όσα υποστηρικτικά στοιχεία αυτή χρειάζεται (που τυπικά παρέχονται από το λειτουργικό σύστημα), υπό τη μορφή βιβλιοθηκών, σε έναν κοινό χώρο διευθύνσεων. Οι παραγόμενες εκτελέσιμες εικόνες εκτελούνται ως εικονικές μηχανές επιτυγχάνοντας υψηλότερη αποδοτικότητα, αφού έχουν μικρότερο μέγεθος (οπότε ταχύτερη μεταφορά και εκκίνηση [32] και χαμηλότερες απαιτήσεις αποθηκευτικού χώρου), χαμηλότερες απαιτήσεις μνήμης και πιο αποδοτικές λειτουργίες συστήματος λόγω πχ έλλειψης mode switches και απλουστευμένου μοντέλου ασφάλειας.

1.1.2 Γιατί κοινό σύστημα αρχείων και virtio-fs

Τυπικά, τα συστήματα αρχείων είναι τοπικά, υλοποιημένα με συσκευές block, στο πλαίσιο του πυρήνα ενός λειτουργικού συστήματος. Υπάρχουν όμως περιπτώσεις οι οποίες ωφελούνται από την κοινή χρήση του ίδιου συστήματος αρχείων από περισσότερα συστήματα. Μία τέτοια περίπτωση είναι αυτή της εικονικοποίησης, όπου host και guest μοιράζονται ένα σύστημα αρχείων (το οποίο συνήθως προϋπάρχει στον host). Ένα παράδειγμα όπου είναι σαφής η χρησιμότητα του παραπάνω είναι στην εκκίνηση εικονικών μηχανών οι οποίες (ανα)διαμορφώνονται από τον host, ενώ ένα άλλο είναι η εκτέλεση βραχύβιων εικονικών μηχανών που διαβάζουν δεδομένα εισόδου ή διαμόρφωσης και γράφουν δεδομένα εξόδου σε ένα κοινόχρηστο σύστημα αρχείων.

Το virtio-fs είναι μια πρόσφατη προσπάθεια στο πεδίο των κοινόχρηστων συστημάτων αρχείων, η πρώτη χτισμένη από την αρχή με αποκλειστικό στόχο τα περιβάλλοντα εικονικοποίησης. Αυτή η εξειδίκευση του επιτρέπει να μην έχει καμία εξάρτηση από δικτυακά πρωτόκολλα ή (εικονική) δικτυακή υποδομή, οδηγώντας αφενός σε χαμηλότερες απαιτήσεις από τον guest για τη χρήση του, αφετέρου σε καλύτερες επιδόσεις και προσφορά σημασιολογίας τοπικού συστήματος αρχείων [55]. Σημαντικό ρόλο στην επίτευξη των προηγούμενων παίζει η χρήση ενός χώρου κοινής μνήμης μεταξύ host και guest όπου απεικονίζονται τα περιεχόμενα των αρχείων, το λεγόμενο DAX (Direct Access) window. Τα παραπάνω το καθιστούν ιδανικό για χρήση σε 'lightweight' guests, συμπεριλαμβανομένων των unikernels.

1.2 Στόχοι

Αυτή η εργασία έχει σκοπό την εξερεύνηση των δυνατοτήτων που προσφέρει το virtio-fs σε ένα unikernel. Συγκεκριμένα, επιλέγουμε το OSv και επεκτείνουμε την ήδη υπάρχουσα, στοιχειώδη υλοποίηση του για το virtio-fs, προσθέτοντας μεταξύ άλλων (read-only) υποστήριξη για το DAX window καθώς και δυνατότητα εκκίνησης (boot) από ένα virtio-fs σύστημα αρχείων. Έτσι καθίσταται πρακτική η χρήση του σε πολλές περιπτώσεις. Επίσης, αξιολογούμε τις επιδόσεις της υλοποίησης μας σε σύγκριση με πλήθος άλλων συστημάτων αρχείων, σε ένα εύρος αντιπροσωπευτικών σεναρίων.

Η εργασία όμως έχει και μη τεχνικούς στόχους που έχουν να κάνουν με το μοντέλο του ελεύθερου λογισμικού / λογισμικού ανοιχτού κώδικα. Μιας και αμφότερα τα έργα που εμπλέκονται (OSv και virtio-fs) είναι έργα ελεύθερου λογισμικού, θεωρούμε ευκαιρία αλλά και υποχρέωση όλες οι επεκτάσεις μας στο OSv να προσφερθούν για συμπερίληψη σε αυτό. Με αυτόν τον τρόπο και άλλοι χρήστες του έργου μπορεί να επωφεληθούν από την εργασία μας, η οποία αποκτά αξία πέραν της ακαδημαϊκής, ενώ, τέλος, οι κοινότητες των έργων αποκτούν ένα νέο, ενεργό μέλος.

1.3 Σχετική δουλειά

Στα χρόνια από την ανακοίνωση του MirageOS [33] έχουν κάνει την εμφάνιση τους πληθώρα unikernel frameworks, σχηματίζοντας ένα πολυποίκιλο σύνολο. Μερικά, εκτός του OSv, είναι τα RumpRun [48], IncludeOS [6], HermitCore [29] και HermiTux [37], ukl [47], Toro [10] και Nanos [7].

Η κυριότερη, προϋπάρχουσα εναλλακτική του virtio-fs είναι το VirtFS [26] που βασίζεται στο δικτυακό πρωτόκολλο 9P [1]. Το virtio-fs είναι ακόμα ένα νεό έργο, υπό ενεργή ανάπτυξη, γι΄ αυτό και είναι διαθέσιμες σχετικά λίγες υλοποιήσεις του. Από τη μεριά του host, πέρα από την κύρια υλοποίηση του στο QEMU [54], υπάρχει πλήρης υλοποίηση του virtio-fs στον cloud-hypervisor [19], μία εναλλακτική υλοποίηση του virtiofsd ονόματι memfsd από την κοινότητα των nabla containers [36] καθώς και μία υλοποίηση στο firecracker [14] η οποία έχει απορριφθεί προς το παρόν. Από τη μεριά του guest, πέρα από την κύρια υλοποίηση του στο Linux [53], υπάρχει υλοποίηση στο Toro (ένα ακόμα unikernel, γραμμένο σε Pascal) [10], όπως και για Windows [56].

1.4 Οργάνωση του παρόντος

Στο δεύτερο κεφάλαιο δίνεται μία πλήρης περιγραφή όλου του τεχνικού υποβάθρου της εργασίας, από τις περιπτώσεις χρήσης του cloud έως και τις τεχνολογίες με χρήση των οποίων υλοποιείται το virtio-fs. Το κεφάλαιο τρία αναλύει τη διαδικασία και τα χαρακτηριστικά της υλοποίησης των επεκτάσεων μας στο OSv. Το τέταρτο κεφάλαιο καταπιάνεται με την αξιολόγηση της τελευταίας, περιγράφοντας τη μεθοδολογία που ακολουθήθηκε και εξετάζοντας τα αποτελέσματα, ενώ το πέμπτο και τελευταίο κεφάλαιο περιέχει μία σύντομη ανασκόπηση και κατευθύνσεις για περαιτέρω δουλειά στο μέλλον.

Κεφάλαιο 2

Υπόβαθρο

2.1 Υπολογιστικό νέφος

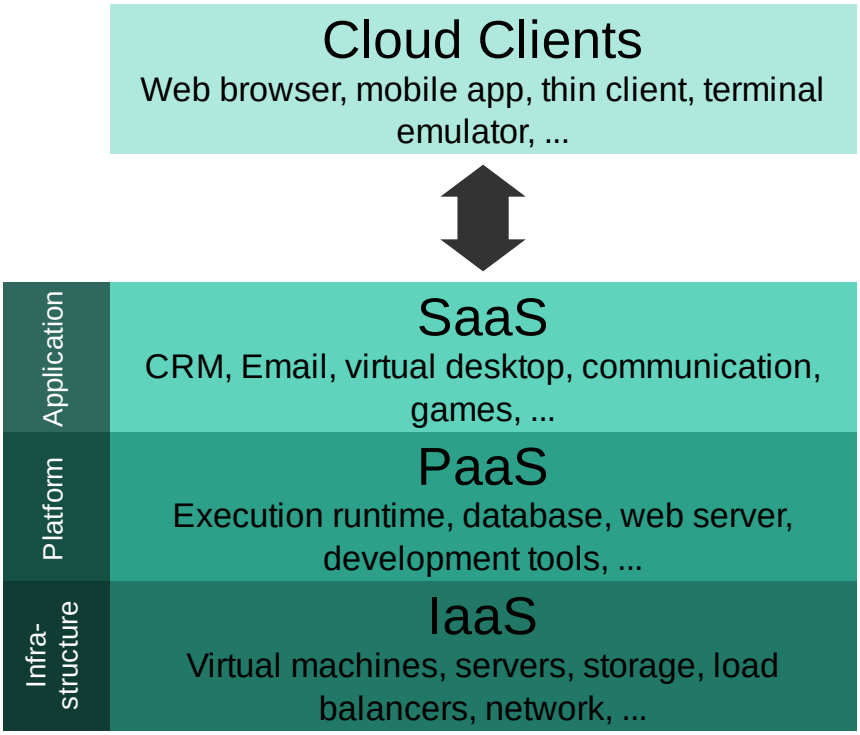
Το υπολογιστικό νέφος (cloud computing) είναι ένα μοντέλο παροχής υπολογιστικών πόρων ως υπηρεσίας το οποίο εδραιώνεται πλέον ως κυρίαρχο. Βασίζεται στην παροχή συγκεντρωμένων, κοινόχρηστων υπολογιστικών πόρων κατ΄ απαίτηση (on demand), μέσω δικτύου. Αυτοί οι φυσικοί πόροι ανήκουν και διαχειρίζονται από έναν πάροχο υπηρεσιών νέφους (cloud service provider) και γίνονται διαθέσιμοι εξ΄ αποστάσεως σε χρήστες - πελάτες, οι οποίοι έτσι επιτυγχάνουν χαμηλότερο αρχικό κόστος για υπολογιστική υποδομή, ευελιξία, κλιμακωσιμότητα, συντομότερο χρόνο deployment και απλοποιημένη διαχείριση [58].

Διακρίνονται τρία βασικά μοντέλα υπηρεσιών cloud μέσω των οποίων παρέχεται πρόσβαση στην υποκείμενη υπολογιστική υποδομή, όπως φαίνεται και στο σχήμα 2.1 [35]:

Infrastructure-as-a-Service (IaaS) όπου γίνονται διαθέσιμοι υπολογιστικοί πόροι επεξεργασίας (compute), αποθήκευσης (storage) και δικτύωσης, όπως εικονικές μηχανές (virtual machines), μονάδες αποθήκευσης, εικονικά δίκτυα και εξισορροπητές φόρτου (load balancers). Αυτό είναι το μοντέλο που προσφέρει το μικρότερο επίπεδο αφαίρεσης (abstraction) παράλληλα με την μεγαλύτερη ευελιξία στους χρήστες.

Platform-as-a-Service (PaaS) στο οποίο παρέχεται η δυνατότητα στους χρήστες να αναπτύξουν τις εφαρμογές τους λαμβάνοντας ως δεδομένο το 'περιβάλλον' στο οποίο αυτές θα εκτελεστούν. Ουσιαστικά, σε αυτό το μοντέλο τους παρέχεται μία υπηρεσία υψηλότερου επιπέδου με την έννοια ότι δεν έχουν την ευθύνη της διαχείρισης συστατικών όπως το λειτουργικό σύστημα και το σύστημα χρόνου εκτέλεσης (runtime system), αφήνοντας τους να επικεντρωθούν στην εφαρμογή.

Software-as-a-Service (SaaS) όπου πρόκειται για το πιο υψηλό επίπεδο αφαίρεσης από την υπολογιστική υποδομή. Σε αυτό δίνεται πρόσβαση στους χρήστες σε



Σχήμα 2.1: Μοντέλα παροχής υπηρεσιών cloud. Πηγή Wikimedia Commons.

συγκεκριμένες εφαρμογές, οι οποίες διαχειρίζονται πλήρως από τον πάροχο, από την υπολογιστική υποδομή μέχρι και την έκδοση και διαμόρφωση της ίδιας της εφαρμογής. Παραδείγματα αποτελούν υπηρεσίες ηλεκτρονικής αλληλογραφίας (email), διαχείρισης έργων (project management) και εφαρμογών γραφείου όπως επεξεργασίας εγγράφων και υπολογιστικών φύλλων (spreadsheets).

Ένα μοντέλο που έχει κάνει την εμφάνιση του πιο πρόσφατα και έχει ελκύσει πολλούς χρήστες είναι το λεγόμενο serverless ή Function-as-a-Service (FaaS). Αυτό ταιριάζει με το PaaS (θα μπορούσε να θεωρηθεί κομμάτι ή εξέλιξη του), καθώς όπως υποδηλώνει το όνομα του, αφαιρεί εξ΄ ολοκλήρου την ευθύνη του server από τους χρήστες της υπηρεσίας. Τα τεχνικά χαρακτηριστικά στα οποία δίνεται έμφαση είναι η αυτόματη κλιμακωσιμότητα από το μηδέν μέχρι και πολύ μεγάλη κλίμακα καθώς και η άμεση απόκριση σε διακυμάνσεις του φόρτου, μιας και η εκκίνηση και ο τερματισμός στιγμιοτύπων της εφαρμογής είναι πολύ γρήγορες διαδικασίες [5]. Ένα μειονέκτημα του serverless είναι ότι κάποιες μόνο εφαρμογές ταιριάζουν σε αυτό το μοντέλο, οπότε είναι δυνατό να επωφεληθούν από τα παραπάνω [18, 20].

2.2 Εικονικοποίηση

Η τεχνολογία που βρίσκεται κατά κύριο λόγο πίσω από το cloud είναι αυτή της εικονικοποίησης (virtualization). Με αυτήν προστίθεται ένα επίπεδο αφαίρεσης πάνω από την φυσική υπολογιστική υποδομή, αφού επιτρέπει την απεικόνιση ενός (συνήθως μεγάλου) φυσικού μηχανήματος (host) σε περισσότερα, μικρότερα εικονικά μηχανήματα (virtual machines – guests), τα οποία δημιουργούνται, τροποποιούνται, μεταφέρονται και καταργούνται δυναμικά, μέσω λογισμικού, θέτοντας τα θεμέλια για τις σύγχρονες υπηρεσίες cloud [59].

Ως προς τις απαιτήσεις από το λογισμικό που εκτελείται ως guest διακρίνονται δύο περιπτώσεις:

Πλήρης εικονικοποίηση (full virtualization) όπου ο guest δεν αντιλαμβάνεται ότι εκτελείται από ένα εικονικό μηχάνημα. Έτσι, δεν απαιτούνται αλλαγές ώστε λογισμικό που έχει δημιουργηθεί για φυσικά μηχανήματα να εκτελεστεί σε εικονικά.

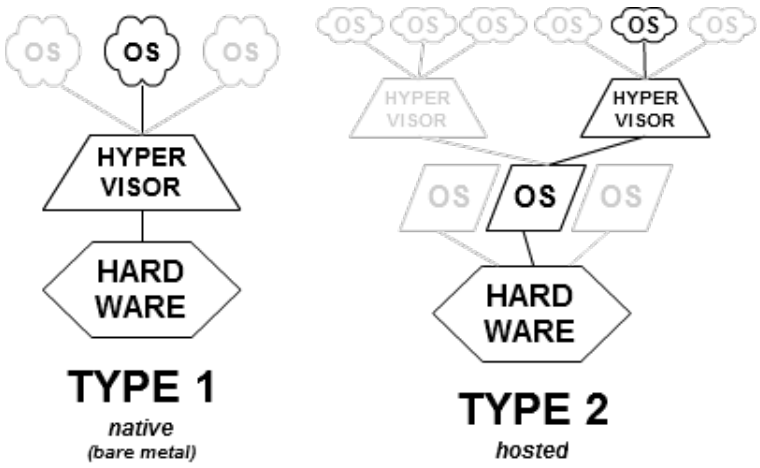
Paravirtualization όπου ο guest τροποποιείται ειδικά για την εκτέλεση του στο virtual machine.

Η εικονικοποίηση ήταν παρούσα για δεκαετίες πριν την έλευση του cloud. Καθοριστικό σημείο για την υλοποίηση αυτού ήταν η προσθήκη υποστήριξης υλικού για την εικονικοποίηση στην αρχιτεκτονική x86, που οδήγησε στην αποδοτική χρήση της, χωρίς υψηλές απαιτήσεις από το λογισμικό [13]. Μέχρι εκείνο το σημείο η εικονικοποίηση σε αυτή τη δημοφιλή αρχιτεκτονική βασιζόταν σε τεχνικές paravirtualization.

Την ευθύνη για την υλοποίηση της εικονικοποίησης έχει ο λεγόμενος hypervisor ή virtual machine monitor (VMM). Αυτός είναι συνήθως λογισμικό, εκτελούμενο στον host με αυξημένα δικαιώματα (privileges). Το έργο του περιλαμβάνει την εκκίνηση των εικονικών μηχανών και την εξομοίωση των ενεργειών που δεν επιτρέπονται σε αυτές, με κυριότερη την αλληλεπίδραση με τις περιφερειακές συσκευές του συστήματος. Το δεύτερο τυπικά υλοποιείται με την τεχνική trap and emulate, όπου συγκεκριμένες εντολές κατά την εκτέλεση της εικονικής μηχανής προκαλούν traps στον επεξεργαστή με συνέπεια τη μεταφορά στον κώδικα του hypervisor. Εκείνος αναλαμβάνει να ελέγξει και να εκτελέσει την ενέργεια που επιχείρησε ο guest, επιστρέφοντας κατόπιν τον έλεγχο στο σημείο όπου είχε σταματήσει αυτός. Ο μηχανισμός ετούτος δίνει στον hypervisor πλήρη έλεγχο στο μοντέλο (εικονικών) συσκευών (device model) του virtual machine [60].

Οι hypervisors διακρίνονται σε δύο πρωταρχικές κατηγορίες (σχήμα 2.2) [46]:

Τύπου 1 οι οποίοι εκτελούνται απευθείας πάνω από το φυσικό μηχάνημα, αναλαμβάνοντας εξ΄ ολοκλήρου τη διαχείριση του, πέρα από τη διαχείριση των εικονικών μηχανών. Τυπικά αυτοί οι hypervisors εξαρτώνται από έναν guest ειδικού σκοπού, ο οποίος έχει αυξημένα προνόμια σε σχέση με τους υπόλοιπους, ώστε να διαχειρίζεται τις συσκευές του μηχανήματος, διαθέτοντας τους κατάλληλους οδηγούς (drivers). Ο πιο γνωστός εκπρόσωπος ελεύθερου λογισμικού αυτής της κατηγορίας είναι ο Xen [16].



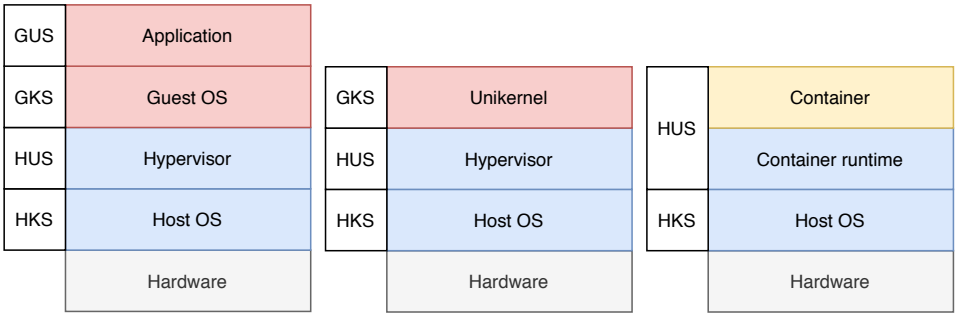
Σχήμα 2.2: Τύποι hypervisors. Πηγή Wikimedia Commons.

Τύπου 2 οι οποίοι εκτελούνται ως μέρος ενός τυπικού λειτουργικού συστήματος, αναλαμβάνοντας μόνο την εκκίνηση και παρακολούθηση των εικονικών μηχανών, ενώ η διαχείριση του host γίνεται από το λειτουργικό, ως συνήθως. Σε αυτή την κατηγορία ο κάθε guest συχνά είναι απλά μία διεργασία από την σκοπιά του host. Το πιο γνωστό παράδειγμα ελεύθερου λογισμικού που ανήκει σε αυτή την κατηγορία είναι το QEMU [17] (τυπικά σε συνδυασμό με το KVM [27], το οποίο παρέχει επιτάχυνση υλικού).

2.3 Unikernels

Στην πλέον συνήθη περίπτωση χρήσης της εικονικοποίησης, μπορούμε να θεωρήσουμε το λογισμικό που υποστηρίζει την εκτέλεση μίας εφαρμογής ως εξής (σχηματικά στο 2.3α΄):

- Στον host (kernel space) εκτελείται ένα γενικού σκοπού λειτουργικό σύστημα που έχει απευθείας πρόσβαση στους πόρους του φυσικού μηχανήματος και είναι υπεύθυνο γι΄ αυτούς.
- Στον host (τυπικά user space) εκτελείται επίσης ο hypervisor, αναλαμβάνοντας τη διαχείριση των εικονικών μηχανών που αντιστοιχούν στο σύστημα.
- Στον guest (kernel space) εκτελείται και πάλι ένα στιγμιότυπο λειτουργικού συστήματος γενικού σκοπού. Αυτό έχει την ευθύνη των πόρων του εικονικού μηχανήματος, όπως έχουν ανατεθεί από τον hypervisor.
- Στον guest (user space) εκτελείται η εφαρμογή, συχνά μαζί με συστατικά όπως βιβλιοθήκες τρίτων και συστήματα χρόνου εκτέλεσης (runtime systems).



(α′) Τυπικό VM (β′) Unikernel (γ′) Container

Σχήμα 2.3: Σύγκριση μεταξύ αρχιτεκτονικής 'κλασικής' εικονικής μηχανής, unikernel και container, με hypervisor τύπου 2. Όπου HKS=Host Kernel Space, HUS=Host User Space, GKS=Guest Kernel Space και GUS=Guest User Space.

Η στοίβα αυτή λογισμικού περιέχει διπλότυπα και υψηλή πολυπλοκότητα, δύο στοιχεία τυπικά υπαίτια για προβλήματα όπως η υποβέλτιστη χρησιμοποίηση των πόρων και οι μειωμένες επιδόσεις λόγω overheads (πχ mode switches) στη συνολική λειτουργία του συστήματος, αλλά και κίνδυνοι ασφαλείας λόγω του μεγέθους του λογισμικού, από το οποίο μεγάλο μέρος είναι περιττό (πχ οδηγοί του guest).

Μία σύγχρονη προσπάθεια αντιμετώπισης του προηγούμενου προβλήματος είναι τα unikernels [33]. Αυτά είναι εκτελέσιμες εικόνες μηχανών (machine images) που αποτελούνται από μία εφαρμογή μαζί με όλες τις απαραίτητες εξαρτήσεις (dependencies) για την εκτέλεση της, από βιβλιοθήκες και runtime systems έως υλοποιήσεις στοίβας δικτύωσης και οδηγούς συσκευών. Όλα τα παραπάνω βρίσκονται σε έναν κοινό, επίπεδο χώρο διευθύνσεων, χωρίς διάκριση σε kernel και user space, όπως φαίνεται και στο σχήμα 2.3β′. Πρόκειται συνεπώς για ειδικού σκοπού πυρήνες, κατασκευασμένους ώστε να τρέχουν μία μόνο εφαρμογή, εντός μίας εικονικής μηχανής, κατόπιν της παρατήρησης ότι συχνά τα virtual machines χρησιμοποιούνται για να εξυπηρετήσουν αποκλειστικά μία εφαρμογή.

Τα unikernels χτίζουν πάνω στην παλαιότερη ιδέα των library operating systems (OS), τα οποία περιγράφονται μαζί με τα πολύ προοδευτικά για την εποχή τους και επίκαιρα σήμερα exokernels [21]. Σε ένα library OS, η πλειονότητα των λειτουργιών που παραδοσιακά προσφέρονται ως μέρος ενός μονολιθικού πυρήνα, όπως το σύστημα αρχείων και η στοίβα δικτύωσης, εξάγονται από αυτόν και γίνονται ανεξάρτητα στοιχεία με την μορφή βιβλιοθηκών, οι οποίες συνοδεύουν την εφαρμογή. Αυτή η αναδιαμόρφωση επιτρέπει στην εφαρμογή ελευθερία επιλογής ως προς το ποια από αυτά τα στοιχεία χρειάζεται να συμπεριλαμβάνει αλλά και ως προς τη συγκεκριμένη υλοποίηση που θα χρησιμοποιήσει για κάθε ένα από αυτά. Συνεπώς, η εφαρμογή δύναται να βελτιστοποιήσει τη λειτουργία της μέσω των παραπάνω επιλογών, αλλά ταυτόχρονα επιβαρύνεται από αυτή την επιπλέον ευθύνη.

Ένα μεγάλο εμπόδιο στην υλοποίηση των unikernels είναι η ποικιλία συσκευών και

κατά συνέπεια οδηγών που χρειάζονται για την υποστήριξη τους. Αυτό ξεπερνιέται χάρη στο μοντέλο συσκευών των hypervisors, το οποίο περιλαμβάνει λίγες, κοινές και καλώς καθορισμένες συσκευές. Ένα άλλο εμπόδιο είναι η πολυπλοκότητα της διαδικασίας ταιριάσματος των διαφορετικών στοιχείων (components) μεταξύ τους και της κατασκευής της τελικής εκτελέσιμης εικόνας από αυτά. Για την αντιμετώπιση του διατίθεται πληθώρα λύσεων, με κάθε μία να σχηματίζει ένα λεγόμενο unikernel framework. Πρακτικά όλα αυτά είναι έργα ελεύθερου λογισμικού, με καθένα να προσφέρει συνήθως ένα σύνολο βιβλιοθηκών για χρήση από τις εφαρμογές των χρηστών, όπως επίσης και τα απαραίτητα εργαλεία για το χτίσιμο των εικόνων. Κάθε ένα από αυτά τα frameworks ακολουθεί τυπικά μία από δύο βασικές προσεγγίσεις:

- Αγνή (clean-slate), που χαρακτηρίζεται από την παροχή μη-πρότυπων (custom) διεπαφών εφαρμογής (APIs). Επίσης, σε αυτή την περίπτωση ο κώδικας των βιβλιοθηκών αλλά και της εφαρμογής είναι γραμμένος στην ίδια γλώσσα. Σαφές μειονέκτημα αποτελεί το γεγονός ότι οι εφαρμογές πρέπει να γραφτούν από την αρχή στη γλώσσα και με χρήση του API του εκάστοτε framework.

- Συμβατή (compatible), που χαρακτηρίζεται από την παροχή παραδοσιακών διεπαφών (πχ POSIX), επιτρέποντας σε ήδη υπάρχουσες εφαρμογές να εκτελεστούν με ελάχιστες έως καθόλου τροποποιήσεις, ανεξάρτητα από τη γλώσσα στην οποία έχουν αρχικά γραφτεί (binary compatibility). Εν προκειμένω, το μεγαλύτερο μειονέκτημα είναι η δέσμευση με συχνά παλαιωμένες διεπαφές, που δεν επιτρέπουν την πλήρη εκμετάλλευση του δυναμικού των unikernels.

Ενδεικτικά, στην πρώτη κατηγορία ανήκουν το MirageOS [33] και το IncludeOS [6], ενώ στη δεύτερη βρίσκει κανείς τα RumpRun [48], OSv [28] και HermiTux [37]. Συνολικά υπάρχει ένας αρκετά μεγάλος αριθμός unikernel frameworks. Η δημοτικότητα τους κυμαίνεται, χωρίς κάποιο να επικρατεί, ενώ ποικίλουν τα στάδια συντήρησης τους (από ενεργά έως πρακτικά εγκαταλελειμμένα) όπως και οι καταβολές τους (ακαδημαϊκά, εταιρικά ή και προσωπικά έργα).

2.3.1 OSv

Το OSv είναι ένα unikernel framework που υποστηρίζει εφαρμογές γραμμένες για το Linux, ενώ παρέχει και δικό του API, το οποίο μπορούν προαιρετικά να χρησιμοποιήσουν νέες εφαρμογές [28]. Εξ΄ αρχής σχεδιάστηκε με το cloud κατά νου, προκειμένου να χρησιμοποιηθεί ιδίως από σύγχρονες εφαρμογές που συχνά συναντώνται σε αυτό. Ως έργο ξεκίνησε από την Cloudius Systems (μετέπειτα ScyllaDB), αλλά τα τελευταία χρόνια συντηρείται εξ΄ ολοκλήρου από μία μικρή ομάδα εθελοντών, ενώ είναι διαθέσιμο με τους όρους της άδειας BSD. Η κοινότητα του χρησιμοποιεί μία mailing list[1] για την ανάπτυξη του: υποβολή patches, code review και γενική συζήτηση.

Αν και τα κομμάτια που παρέχει το ίδιο είναι γραμμένα σε C++, ενώ χρησιμοποιεί κομμάτια από άλλα έργα που ως επί το πλείστον έχουν γραφτεί σε C, δεν θέτει περιορισμούς στις εφαρμογές που υποστηρίζει. Το τελευταίο βέβαια ισχύει αρκεί αυτές να

[1] `https://groups.google.com/forum/#!forum/osv-dev`

μην κάνουν χρήση λειτουργιών που από τη φύση τους δεν υλοποιούνται στα unikernels: κλήσεις συστήματος στην οικογένεια των `fork()` και `exec()`. Επίσης, διαθέτει υποστήριξη για πολλούς hypervisors, μεταξύ των οποίων οι QEMU/KVM, Xen και firecracker [14]. Ως γενική παρατήρηση, είναι από τα πλέον εξελιγμένα unikernels από πλευράς λειτουργιών, και πιο 'βαρύ' (heavy-weight) κατά συνέπεια.

Συγκεκριμένα όσον αφορά τα συστήματα αρχείων, το OSv προσφέρει πολλές επιλογές. Κατ' αρχάς, διαθέτει ένα ολοκληρωμένο εικονικό σύστημα αρχείων (VFS, Virtual File System), το οποίο έχει βασίσει σε αυτό του Prex [8]. Κάτω από αυτό βρίσκονται οι υλοποιήσεις των διάφορων συστημάτων αρχείων που διαθέτει, τα οποία διακρινόμενα σε δύο κατηγορίες είναι:

- Τα pseudo-file systems, αντίστοιχα με αυτά στο Linux: devfs, procfs, sysfs και ramfs.
- Τα συμβατικά συστήματα αρχείων: ZFS (βασισμένο σε υλοποίηση του FreeBSD), rofs (custom, read-only file system με απλή λογική caching) και NFS (χρησιμοποιώντας τη libnfs [30]).

2.3.2 Εναλλακτικές

Όπως είναι αναμενόμενο, έχουν προταθεί και εναλλακτικές προσεγγίσεις στο πρόβλημα του bloated virtualized software stack. Με διαφορά η πιο διαδεδομένη αυτήν την περίοδο είναι εκείνη των containers. Αυτά ανήκουν στην ευρεία κατηγορία της εικονικοποίησης σε επίπεδο λειτουργικού συστήματος (OS-level virtualization [61], σχηματικά στο 2.3γ').

Υπάρχουν πολλές τεχνικές υλοποίησης των containers, οι περισσότερες εκ των οποίων βασίζονται σε μηχανισμούς του Linux kernel όπως τα cgroups και namespaces προκειμένου να παρέχουν απομόνωση μεταξύ των διαφορετικών containers. Το κοινό όλων τους είναι ότι οι εφαρμογές που τρέχουν σε όλα τα containers του ίδιου μηχανήματος μοιράζονται το ίδιο στιγμιότυπο πυρήνα (kernel). Θα μπορούσαμε να πούμε ότι σε αυτή την περίπτωση έχουμε ένα kernel, αλλά πολλαπλά user spaces.

Στα σημαντικά πλεονεκτήματα των containers είναι ότι είναι πολύ πιο 'ελαφριά' από τις εικονικές μηχανές: πολύ χαμηλότερος χρόνος εκκίνησης (από ένα virtual machine με guest γενικού σκοπού), σε συνδυασμό με μεγαλύτερη ευελιξία και πιο αποδοτική χρήση των πόρων, με αποτέλεσμα να μπορούν να επιτύχουν υψηλότερη πυκνότητα (στιγμιότυπα που εκτελούνται ταυτόχρονα στο ίδιο φυσικό μηχάνημα). Το σημαντικότερο μειονέκτημα τους είναι ότι προσφέρουν κατ' αρχήν πολύ πιο ασθενή απομόνωση από ότι οι εικονικές μηχανές, λόγω του κοινού πυρήνα που τα εξυπηρετεί απευθείας [34].

2.4 Κοινόχρηστα συστήματα αρχείων

Τα 'κλασικά' συστήματα αρχείων είναι υλοποιημένα πάνω από συσκευές block ('δίσκους'), συνδεδεμένες μέσω ενός διαύλου (bus) τοπικού συστήματος. Το λογισμι-

κό που τα υλοποιεί είναι συνήθως μέρος του πυρήνα ενός λειτουργικού συστήματος, το οποίο θεωρεί ότι έχει την αποκλειστικότητα επί του συστήματος αρχείων από τη στιγμή της προσάρτησης (mount) μέχρι και την στιγμή της αποπροσάρτησης (unmount) αυτού. Το τελευταίο αποτελεί σε πολλές περιπτώσεις έναν περιορισμό, ο οποίος ήδη από την έλευση της δικτύωσης των υπολογιστών επιδιώχθηκε να αρθεί, επιτρέποντας την ταυτόχρονη χρήση ενός συστήματος αρχείων από περισσότερους υπολογιστές, καθιστώντας το κοινό. Ο πιο γνωστός και μεταξύ των παλαιότερων εκπροσώπων κοινόχρηστων συστημάτων αρχείων είναι το NFS (Network File System) [49], ενώ η κατηγορία έχει εμπλουτιστεί σημαντικά με την επέκταση των κατανεμημένων υπολογιστικών συστημάτων τις τελευταίες δεκαετίες.

Με την εικονικοποίηση έχουμε μία ιδιαίτερη περίπτωση πολλαπλών, συνδεδεμένων μηχανημάτων: του host και των virtual machines που εκτελούνται σε αυτόν. Το ζητούμενο του κοινόχρηστου συστήματος αρχείων μεταξύ τους είναι συνεπώς κι εδώ παρόν και μάλιστα, λόγω της συνύπαρξης στο ίδιο φυσικό μηχάνημα είναι μία πολύ συχνή απαίτηση. Εξάλλου, το σύστημα αρχείων στον host αποτελεί έναν ακόμα πόρο του, οπότε είναι αναμενόμενο να μπορεί να δοθεί πρόσβαση σε αυτόν από τους guests.

Οι περισσότερες λύσεις για κοινόχρηστα συστήματα αρχείων μεταξύ host και guest βασίζονται στα ήδη υπάρχοντα, δικτυακά συστήματα αρχείων, μη διακρίνοντας αυτή την περίπτωση από εκείνη των χωριστών hosts. Αυτή η προσέγγιση λειτουργεί βεβαίως βασιζόμενη στη σύνδεση δικτύου μεταξύ των μερών, πλήρως επαναχρησιμοποιώντας ήδη υπάρχουσες λύσεις. Ένα δημοφιλές παράδειγμα κοινόχρηστου συστήματος αρχείων στο πλαίσιο της εικονικοποίησης είναι το VirtFS [26], που εκμεταλλεύεται το πρωτόκολλο 9P [1]. Αν και το τελευταίο είναι ένα δικτυακό πρωτόκολλο, στην πράξη το VirtFS δεν χρησιμοποιεί το δίκτυο στο επίπεδο μεταφοράς του, κάτι που του επιτρέπει βελτιωμένες επιδόσεις.

Αξίζει να αναφέρουμε ότι και στην περίπτωση των containers είναι ιδιαίτερα συχνή η απαίτηση της πρόσβασης σε κοινό σύστημα αρχείων με τον host. Ένας τρόπος με τον οποίο αυτό επιτυγχάνεται στο Docker [2] (με διαφορά το πιο δημοφιλές container runtime) είναι η χρήση των λεγόμενων bind mounts [11], που εκμεταλλεύονται τον κοινό πυρήνα για να προσαρτήσουν έναν κατάλογο του host εντός του συστήματος αρχείων του container.

2.5 virtio-fs

Η κοινή υποκείμενη μνήμη ανάμεσα στον host και τους guests είναι ένα στοιχείο το οποίο δεν εκμεταλλεύονται πλήρως τα υπάρχοντα κοινόχρηστα συστήματα αρχείων. Αυτό έρχεται να αλλάξει το virtio-fs [55], το οποίο έχει σχεδιαστεί εξ΄ αρχής με ετούτο το σκοπό. Το στοιχείο που το διαφοροποιεί από τα υπόλοιπα είναι ότι δεν εξαρτάται από το δίκτυο στο επίπεδο μεταφοράς, όπου χρησιμοποιεί το virtio, ούτε όμως και στο επίπεδο του πρωτοκόλλου, όπου βασίζεται στο FUSE. Αυτές οι δύο επιλογές του επιτρέπουν καλύτερες επιδόσεις, αλλά και σημασιολογία (semantics) τοπικού συστήματος αρχείων, μια διαφορά που εκδηλώνεται για παράδειγμα σε θέματα συνοχής (coherence).

Η υπάρχουσα υλοποίηση του virtio-fs στο QEMU έχει μία αρχιτεκτονική ιδιαιτερότητα. Ενώ τυπικά η υλοποίηση των συσκευών περιέχεται εξ΄ ολοκλήρου στο QEMU, εδώ ένα σημαντικό μέρος της υλοποίησης είναι διαχωρισμένο, υλοποιημένο στο λεγόμενο virtiofsd (virtio-fs daemon), μία ανεξάρτητη διεργασία που αναλαμβάνει όλες τις λειτουργίες του συστήματος αρχείων στον host, ενώ το QEMU έχει μόνο την ευθύνη της εικονικής συσκευής. Αυτή η αρχιτεκτονική έχει αφενός το όφελος υψηλότερης ασφάλειας, καθώς ο virtiofsd μπορεί περιοριστεί σε ένα sandbox χρησιμοποιώντας μηχανισμούς του host (namespaces, seccomp). Αφετέρου, είναι αρθρωτή (modular), επιτρέποντας εύκολη εναλλαγή ανάμεσα σε διαφορετικές υλοποιήσεις του virtiofsd (πχ με μία που αντί πρόσβασης στο σύστημα αρχείων του host να χρησιμοποιεί κάποιο κατανεμημένο σύστημα αρχείων). Ο διαχωρισμός της υλοποίησης σε περισσότερες διεργασίες καθίσταται δυνατός χάρη στο vhost-user πρωτόκολλο [12], το οποίο με τη σειρά του εξελίσσει την ιδέα του vhost στο QEMU [23]. Η ώθηση των συσκευών εκτός του VMM έχει αρκετά πλεονεκτήματα και είναι μία τάση που αναμένεται να ακολουθηθεί και στο μέλλον [24].

2.5.1 Virtio

Το virtio είναι μία πρότυπη (standard) προδιαγραφή συσκευών, αποκλειστικά για virtualized περιβάλλοντα [52]. Χαίρει ευρείας υποστήριξης και είναι η πλέον αποδεκτή λύση στο πρόβλημα του κατακερματισμού (fragmentation) των μοντέλων συσκευών σε αυτά. Προσφέρει έναν αποδοτικό και επεκτάσιμο μηχανισμό, ενώ βασίζεται σε ορολογία και μηχανισμούς φυσικών συσκευών, διευκολύνοντας την υποστήριξη και επιτρέποντας την επαναχρησιμοποίηση από υπάρχουσες υλοποιήσεις οδηγών στους guests.

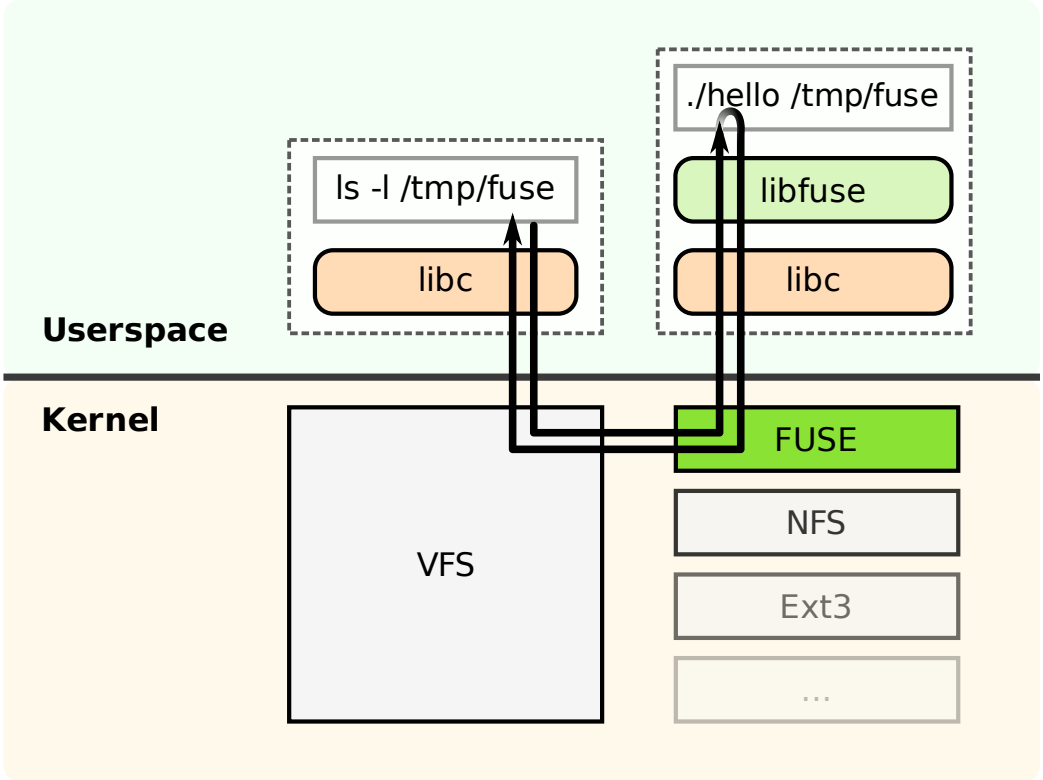
Το πρότυπο έχει δύο βασικούς άξονες:

Το επίπεδο μεταφοράς το οποίο ορίζει έναν γενικό τρόπο για την ανακάλυψη (discovery), διαμόρφωση (configuration) και κανονική λειτουργία (αμφίδρομη μεταφορά δεδομένων) των συσκευών. Η τελευταία γίνεται μέσω των λεγόμενων virtqueues, βασιζόμενη στην κοινή μνήμη μεταξύ guest και hypervisor. Το επίπεδο μεταφοράς, το οποίο κατ΄ αρχήν ορίζεται αφηρημένα, εξειδικεύεται με τρεις υλοποιήσεις: δίαυλο (bus) PCI, memory-mapped I/O (MMIO) και channel I/O (για περιπτώσεις βασισμένες στην αρχιτεκτονική S/390 της IBM).

Το σύνολο των συσκευών που χτίζεται πάνω από το επίπεδο μεταφοράς, καθορίζοντας για κάθε συσκευή τα διαθέσιμα πεδία διαμόρφωσης (configuration fields) της, το πλήθος των virtqueues που χρησιμοποιεί και τα μηνύματα που μεταφέρονται μέσω αυτών για τη λειτουργία της.

2.5.2 FUSE

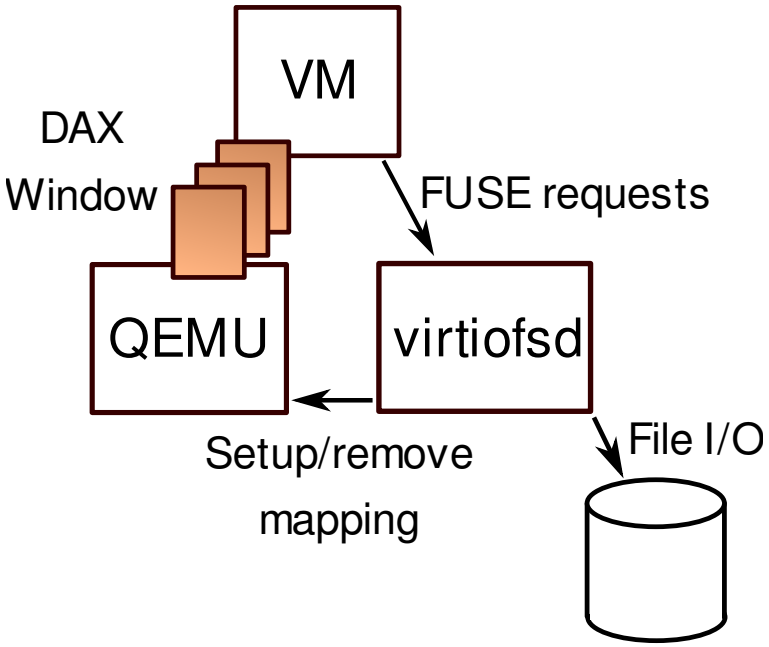
Το FUSE (Filesystem in Userspace) είναι ένα πρωτόκολλο που συστήθηκε στο Linux προκειμένου να επιτρέψει την υλοποίηση συστημάτων αρχείων στο user space



Σχήμα 2.4: Δομή του FUSE στο Linux. Από τον Sven (`https://commons.wikimedia.org/wiki/User:Sven`) υπό την άδεια CC-BY-SA 3.0 (`https://creativecommons.org/licenses/by-sa/3.0/legalcode`). Πηγή `https://commons.wikimedia.org/wiki/File:FUSE_structure.svg`.

αντί εντός του πυρήνα [4]. Όπως φαίνεται και στο σχήμα 2.4, στην αρχιτεκτονική του η υλοποίηση των λειτουργιών του συστήματος αρχείων γίνεται από ένα user space daemon που επικοινωνεί σύμφωνα με το πρωτόκολλο με τον πυρήνα μέσω μίας ειδικής συσκευής χαρακτήρων (`/dev/fuse`). Όταν το kernel δέχεται VFS αιτήματα που αντιστοιχούν σε κάποιο FUSE σύστημα αρχείων, τα προωθεί στο κατάλληλο daemon, που έχει το ρόλο του backend.

Στο πλαίσιο του virtio-fs, το FUSE αξιοποιείται ως πρωτόκολλο επικοινωνίας των αιτημάτων συστήματος αρχείων με τη συσκευή. Έτσι, κατά κάποιον τρόπο η αρχιτεκτονική του μοιάζει με την κλασική αρχιτεκτονική του FUSE με τη διαφορά ότι πελάτης (client) είναι ο guest αντί του πυρήνα, ενώ η επικοινωνία γίνεται μέσω virtio αντί της FUSE συσκευής χαρακτήρων. Στην πράξη όμως, το virtio-fs και το τυπικό FUSE είναι ασύμβατα μεταξύ τους, με το virtio-fs να περιλαμβάνει τροποποιήσεις και επεκτάσεις στο πρωτόκολλο του FUSE. Επίσης, το μοντέλο ασφαλείας (security model) είναι διαφορετικό (ουσιαστικά ανεστραμμένο), μιας και στο virtio-fs ο client (guest) είναι αναξιόπιστος, ενώ στο FUSE συμβαίνει το αντίθετο (υπάρχει εξ΄ ορισμού



Σχήμα 2.5: Αρχιτεκτονική του DAX window στο virtio-fs. Από τον Stefan Hajnoczi (`https://vmsplice.net/`) υπό την άδεια CC-BY-SA 4.0 (`https://creativecommons.org/licenses/by-sa/4.0/legalcode`). Πηγή `https://gitlab.com/virtio-fs/virtio-fs.gitlab.io/-/blob/master/architecture.svg`.

εμπιστοσύνη στον πυρήνα πάνω στον οποίο τρέχει ο daemon). Το τελευταίο μεταφράζεται σε πιο ασφαλή χειρισμό των αιτημάτων σε ένα virtio-fs device backend από ότι σε ένα FUSE backend.

2.5.3 DAX window

Ο τρόπος που διαθέτει το FUSE για την ανάγνωση και εγγραφή αρχείων είναι μέσω των FUSE_READ και FUSE_WRITE requests αντίστοιχα. Αυτά προβλέπουν την αντιγραφή των δεδομένων και, στην περίπτωση του virtio-fs συνεπάγονται και εξόδους της εικονικής μηχανής (VM exits). Καθώς όμως αυτές οι δύο λειτουργίες είναι κεντρικές στο data path, η βελτιστοποίηση τους είναι απαραίτητη. Προς αυτή την κατεύθυνση το virtio-fs συστήνει το λεγόμενο DAX window: ένα 'παράθυρο' κοινής μνήμης ανάμεσα στον host και το guest, στο οποίο απεικονίζονται τα περιεχόμενα των αρχείων, δίνοντας στον guest απευθείας πρόσβαση σε αυτά. Με αυτόν τον τρόπο επιτυγχάνεται αποφυγή των αντιγραφών (και παράκαμψη της page cache του guest στην περίπτωση του Linux), ενώ πλέον δεν συνεπάγεται κάθε λειτουργία ανάγνωσης ή εγγραφής έξοδο από την εκτέλεση της εικονικής μηχανής.

Για την υλοποίηση της λειτουργικότητας του DAX window κατ' αρχάς επεκτείνεται

το FUSE πρωτόκολλο με μηνύματα για την εγκαθίδρυση και την κατάργηση απεικονίσεων (mappings). Αυτά τα μηνύματα χρησιμοποιεί ο guest προκειμένου ο virtiofsd (κατόπιν των απαραίτητων ελέγχων) να καθοδηγήσει το QEMU μέσω της μεταξύ τους σύνδεσης να πραγματοποιήσει καθεαυτή τη λειτουργία (δημιουργία ή κατάργηση απεικόνισης). Σχηματικά τα παραπάνω φαίνονται στο 2.5.

Κεφάλαιο 3

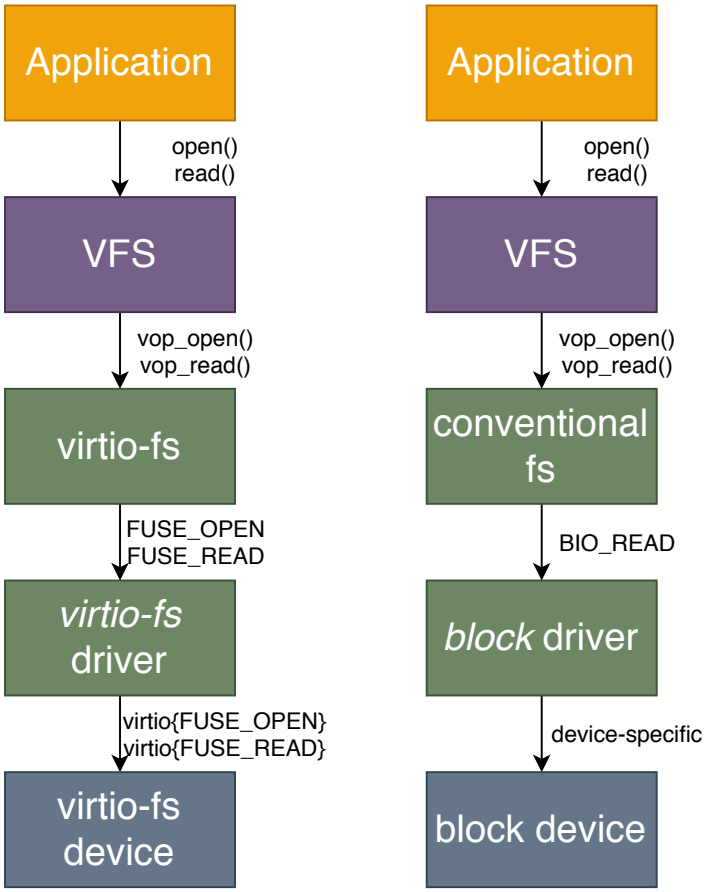
Υλοποίηση

Μία βασική υλοποίηση-σκελετός του virtio-fs προϋπήρχε στο OSv. Αυτή υποστήριζε απλή (μέσω της κλήσης FUSE_READ) ανάγνωση αρχείων (και όχι καταλόγων). Σε συνεννόηση με τα υπόλοιπα μέλη της κοινότητας του OSv, θέσαμε στόχο να αναπτύξουμε περαιτέρω τη λειτουργικότητα αυτής της υλοποίησης. Το κομβικό σημείο αυτής της προσπάθειας ήταν η υποστήριξη ανάγνωσης μέσω του virtio-fs DAX window, κάτι που είχε σχεδιαστεί και υλοποιηθεί (σε 'πειραματικό' ακόμα στάδιο) από τους ανθρώπους του virtio-fs με σκοπό την δραματική βελτίωση των επιδόσεων του συστήματος αρχείων. Εκτός αυτού, είχαμε την ευκαιρία να προσθέσουμε υποστήριξη για ανάγνωση καταλόγων, καθώς και εκκίνηση (boot) του OSv με το virtio-fs ως ριζικό (root) σύστημα αρχείων. Στην πορεία απαιτήθηκαν δομικές αλλαγές, διορθώσεις και βελτιώσεις στην προϋπάρχουσα υλοποίηση.

Σε αυτό το κεφάλαιο αναλύουμε την υλοποίηση των πιο ουσιαστικών από τις παραπάνω συμβολές: της ανάγνωσης αρχείων μέσω DAX window και του boot από virtio-fs.

3.1 DAX window στο virtio-fs

Το OSv ακολουθεί μία εσωτερική δομή συμβατή με αυτή των περισσότερων πυρήνων γενικού σκοπού, ορίζοντας χωριστά τις έννοιες του οδηγού (driver) και του συστήματος αρχείων. Οι οδηγοί (στο drivers/) είναι υπεύθυνοι για την παροχή μίας διεπαφής (interface) κοινής για κάθε οικογένεια συσκευών (αποθήκευσης block, δικτύου κλπ), η οποία επιτρέπει τη χρήση της εκάστοτε συσκευής από τα υπόλοιπα υποσυστήματα του πυρήνα. Τα συστήματα αρχείων (στο fs/) αποτελούν επίσης υλοποιήσεις μίας κοινής διεπαφής η οποία ορίζεται από το εικονικό σύστημα αρχείων (virtual file system) του OSv. Αυτή η διεπαφή αποτελείται από λειτουργίες όπως το άνοιγμα, το κλείσιμο, η ανάγνωση από αρχείο, και άλλες. Τα συστήματα αρχείων συνήθως είναι υλοποιημένα πάνω από μία συσκευή block και χρησιμοποιούν τη διεπαφή που αυτή παρέχει για να 'μεταφράσουν' ανάμεσα στις υψηλού επιπέδου λειτουργίες του συστήματος αρχείων και τις χαμηλού επιπέδου λειτουργίες της συσκευής απο-



Σχήμα 3.1: Συστατικά και εξαρτήσεις στον guest: virtio-fs σε σύγκριση με συμβατικά συστήματα τοπικά αρχείων.

θήκευσης. Υπάρχουν βεβαίως εξαιρέσεις όπως το NFS, το οποίο υποστηρίζεται από το δικτυακό υποσύστημα.

Το virtio-fs είναι ιδιαίτερο στο γεγονός ότι, ως σύστημα αρχείων εξαρτάται από την ομώνυμη συσκευή, η οποία δεν μπορεί να αντιμετωπιστεί ως μέλος κάποιας από τις συνήθεις οικογένειες (πχ block ή δικτύου). Αυτό το χαρακτηριστικό το κληρονομεί βεβαίως από το FUSE. Έτσι, αναπόφευκτα υπάρχει ισχυρή, ρητή εξάρτηση του virtio-fs συστήματος αρχείων από τον οδηγό της virtio-fs συσκευής (σχήμα 3.1), κάτι που βλέπουμε και στην περίπτωση του Linux, όπου αυτά τα δύο είναι υλοποιημένα μαζί (στο fs/fuse/virtio_fs.c). Από τα παραπάνω γίνεται σαφές ότι η διάκριση ανάμεσα σε virtio-fs οδηγό και σύστημα αρχείων (δηλαδή ο αντίστοιχος καταμερισμός των λειτουργιών) είναι εύκαμπτη. Παρ' όλα αυτά, η παρουσίαση την υλοποίησης που ακολουθεί γίνεται υπό το πρίσμα αυτής της διάκρισης.

3.1.1 Οδηγός

Ο οδηγός της virtio-fs συσκευής είναι επιφορτισμένος με την απευθείας επικοινωνία με αυτή και τη χαμηλού επιπέδου διαχείριση της. Πέρα από την αρχικοποίηση (ανακάλυψη virtqueues, διαπραγμάτευση παραμέτρων λειτουργίας) [52], το κύριο έργο του είναι η μεταφορά αιτημάτων (requests) και απαντήσεων (responses) σε αυτά από και προς τη συσκευή μέσω των virtqueues. Γι΄ αυτό το σκοπό παρέχει μία ασύγχρονη διεπαφή υποβολής αιτημάτων, στα οποία ενθυλακώνονται τα FUSE αιτήματα. Ο οδηγός παραμένει αγνωστικός ως προς το FUSE, χειριζόμενος τα αιτήματα του ως αδιαφανή.

Η αρχικά διαθέσιμη υλοποίηση του virtio-fs DAX window στο QEMU ήταν ως συσκευή PCI. Σύμφωνα με την προδιαγραφή του virtio [52] για τη συσκευή, το DAX window είναι η περιοχή κοινής μνήμης (shared memory region) με αναγνωριστικό ('shmid') 0. Στην περίπτωση του virtio PCI transport, οι περιοχές κοινής μνήμης υλοποιούνται ως PCI BARs (base address registers), κάθε ένα εκ των οποίων εκτίθεται ως ένα VIRTIO_PCI_CAP_SHARED_MEMORY_CFG PCI capability, έχοντας ένα μοναδικό αναγνωριστικό [52, 57, 38].

Το πρώτο βήμα για την προσθήκη του DAX window ήταν λοιπόν η ανακάλυψη (discovery) αυτού, εφόσον παρέχονταν από τη συσκευή. Αυτή ήταν μία ανώδυνη εργασία, χάρη στο πλήρες και καλά μοντελοποιημένο υποσύστημα PCI του OSv. Συγκεκριμένα, οι επιμέρους αλλαγές που χρειάστηκαν ήταν:

1. Προσθήκη στο PCI υποσύστημα ώστε να είναι δυνατή η ανακάλυψη πολλαπλών capabilities με τον ίδιο τύπο (στο drivers/pci-function.cc). Αυτή ήταν απαραίτητη διότι μέχρι πρότινος οι λειτουργίες ανακάλυψης στο OSv αναζητούσαν μέχρι και το πρώτο capability ενός τύπου, σταματώντας εκεί. Όμως, όπως αναφέραμε προηγουμένως, κάθε περιοχή κοινής μνήμης αντιστοιχεί σε ένα PCI capability και αυτές μπορεί να είναι περισσότερες από μία (αν και στην περίπτωση του virtio-fs υπάρχει μόνο μία).

2. Επέκταση στο μοντέλο των virtio PCI συσκευών (στο drivers/virtio-pci-device.cc) ώστε να ανακαλύπτει όλες τις περιοχές κοινής μνήμης κάθε συσκευής, αναζητώντας τα capabilities αντίστοιχου τύπου και αποθηκεύοντας τα στοιχεία (διεύθυνση και μέγεθος) των BARs που αυτά υποδεικνύουν.

3. Επέκταση στον οδηγό της virtio-fs συσκευής (στο drivers/virtio-fs.cc) ώστε να ανακαλύπτει την περιοχή κοινής μνήμης με αναγνωριστικό 0, δηλαδή το DAX window και να το εκθέτει απευθείας στη διεπαφή του, ως περιοχή MMIO (memory-mapped I/O) της συσκευής. Σημειώνεται ότι η απεικόνιση (mapping) της περιοχής στον εικονικό χώρο διευθύνσεων έχει ήδη πραγματοποιηθεί από το μοντέλο της συσκευής.

3.1.2 Σύστημα αρχείων

Το virtio-fs σύστημα αρχείων θεωρεί δεδομένο έναν μηχανισμό αποστολής FUSE αιτημάτων (ο οποίος παρέχεται από τον οδηγό) και πάνω σε αυτόν χτίζει τη λειτουργικότητα του. Είναι συνεπώς επιφορτισμένο με τη γνώση και τον χειρισμό του περιεχομένου των αιτημάτων που ο οδηγός απλώς μεταφέρει.

3.1.2.1 Απεικόνιση αρχείων

Η προσάρτηση (mounting) ενός virtio-fs συστήματος αρχείων συνίσταται στην έναρξη μίας FUSE συνεδρίας (session) με την αποστολή ενός FUSE_INIT αιτήματος [52]. Με αυτό πραγματοποιείται η διαπραγμάτευση των παραμέτρων της συνεδρίας, μεταξύ των οποίων και μίας που αφορά τη λειτουργία του DAX window: η ευθυγράμμιση των απεικονίσεων (map alignment). Η πρώτη τροποποίηση που απαιτούνταν στο σύστημα αρχείων λοιπόν ήταν η επέκταση της διαδικασίας προσάρτησης ώστε να γνωστοποιείται στη συσκευή ότι υποστηρίζεται από το λειτουργικό το DAX window και να λαμβάνεται η τιμή της παραμέτρου από την απάντηση.

Η παραπάνω παράμετρος περιορίζει τα mappings αρχείων στο DAX window ως εξής: τόσο η μετατόπιση (offset) της αρχής ενός mapping μέσα στο αρχείο όσο και στο DAX window πρέπει να είναι ευθυγραμμισμένες σύμφωνα με το map alignment. Αυτό στην πράξη επιβάλλεται στην παρούσα υλοποίηση της virtio-fs συσκευής από τη χρήση της `mmap()` [31] στον host (επίσης ο λόγος που το map alignment συνήθως ταυτίζεται με το μέγεθος της σελίδας μνήμης στον host).

Όπως ορίζεται από την προδιαγραφή, η παροχή του DAX window από τη συσκευή όπως και η χρησιμοποίηση του από τον guest είναι αμφότερα προαιρετικά. Επίσης, η χρήση του DAX window είναι ανεξάρτητη από τη χρήση της συμβατικής FUSE_READ (που είναι πάντοτε διαθέσιμη) για την ανάγνωση αρχείων. Έτσι, η υλοποίηση μας, εφόσον είναι διαθέσιμο το DAX window, επιχειρεί να ικανοποιήσει όλες τις αναγνώσεις αρχείων μέσω εκείνου. Σε περίπτωση όμως που αυτό δεν είναι διαθέσιμο ή αποτύχει η διαδικασία της ανάγνωσης από εκείνο, στρέφεται δυναμικά στην εφεδρική FUSE_READ, όπως φαίνεται και στο διάγραμμα 3.3.

Για την εγκαθίδρυση ενός νέου mapping, το virtio-fs επεκτείνει το πρωτόκολλο του FUSE συστήνοντας τη λειτουργία FUSE_SETUPMAPPING. Ένα αίτημα τέτοιου τύπου περιγράφεται από μία δομή (struct) fuse_setupmapping_in (ο ορισμός φαίνεται στον κώδικα 3.1). Για την ακύρωση υπαρχόντων απεικονίσεων έχει προστεθεί η FUSE_REMOVEMAPPING, με το σώμα της να αποτελείται από ένα struct fuse_removemapping_in ακολουθούμενο από το υποδεικνυόμενο πλήθος από struct fuse_removemapping_one.

Τέλος, σημειώνουμε ότι σύμφωνα με την προδιαγραφή, ένα νέο mapping μπορεί να ζητηθεί ενώ επικαλύπτει κάποιο υπάρχον. Εάν η κάλυψη είναι πλήρης, το προϋπάρχον αντικαθίσταται από το νέο, ενώ αν επικαλύπτεται μερικώς, διασπάται. Ένα αίτημα για νέο mapping μπορεί να αποτύχει σε περίπτωση εξάντλησης πόρων της συσκευής, οπότε προτείνεται να δοκιμαστεί ξανά κατόπιν απελευθέρωσης πόρων μέσω της ακύρωσης υπαρχουσών απεικονίσεων. Έτσι και στην υλοποίηση μας η λειτουργία

```
#define FUSE_SETUPMAPPING_FLAG_WRITE (1ull << 0)
struct fuse_setupmapping_in {
    /* An already open handle */
    uint64_t        fh;
    /* Offset into the file to start the mapping */
    uint64_t        foffset;
    /* Length of mapping required */
    uint64_t        len;
    /* Flags, FUSE_SETUPMAPPING_FLAG_* */
    uint64_t        flags;
    /* memory offset in to dax window */
    uint64_t        moffset;
};

struct fuse_removemapping_in {
    /* number of fuse_removemapping_one follows */
    uint32_t        count;
};

struct fuse_removemapping_one {
    /* Offset into the dax to start the unmapping */
    uint64_t        moffset;
    /* Length of mapping required */
    uint64_t        len;
};
```

Κώδικας 3.1: Ορισμοί του FUSE για τις λειτουργίες απεικονίσεων. Βλέπε παράρτημα A για σημείωμα πνευματικών δικαιωμάτων.

FUSE_REMOVEMAPPING δεν χρησιμοποιείται υπό φυσιολογικές συνθήκες.

3.1.2.2 Διαχειριστής (DAX window manager)

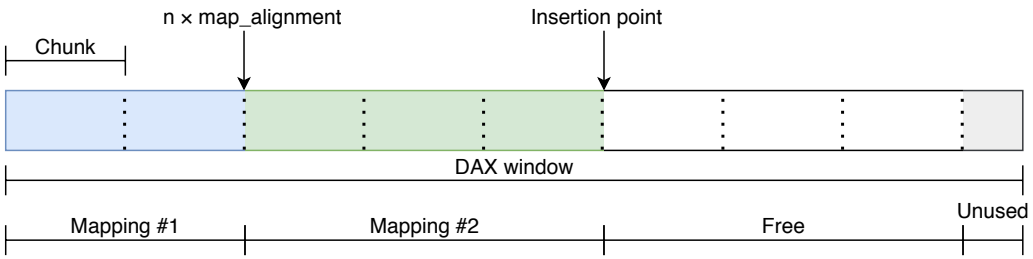
Από τη δυνατότητα λεπτού ελέγχου των DAX window mappings που προσφέρεται, προκύπτει το ζήτημα της βέλτιστης διαχείρισης του. Το τελευταίο έχει πολλά κοινά με το πρόβλημα της διαχείρισης μνήμης στο πλαίσιο ενός λειτουργικού συστήματος με σελιδοποίηση (paging), μιας και λόγω του map alignment οι απεικονίσεις γίνονται πάντα ευθυγραμμισμένες με αυτό. Για να το αντιμετωπίσουμε λοιπόν χρειάστηκε να υλοποιήσουμε έναν διαχειριστή για το DAX window στο επίπεδο του συστήματος αρχείων, μέσω του οποίου συμβαίνουν όλες οι αναγνώσεις, επιτρέποντας του να εφαρμόσει την πολιτική που περιγράφουμε στη συνέχεια.

Για την διαμόρφωση της πολιτικής διαχείρισης λήφθηκαν υπόψη τα εξής:

- Στην περίπτωση μας, το σύστημα αρχείων είναι μόνο για ανάγνωση (read-only).
- Το κόστος σε χρόνο ενός αιτήματος FUSE_SETUPMAPPING είναι ανεξάρτητο του μεγέθους της απεικόνισης η οποία ζητείται.
- Μία σχετικά απλή πολιτική μεταφράζεται σε απλούστερη υλοποίηση, η οποία είναι ευκολότερο να είναι ορθή και αποδοτική.

Σύμφωνα με αυτά οδηγηθήκαμε σε ένα σχήμα διαχείρισης που συνοψίζεται από τα παρακάτω χαρακτηριστικά (απεικονίζονται μερικώς και στο σχήμα 3.2):

- Το DAX window χωρίζεται σε κομμάτια (*chunks*) σταθερού μεγέθους (2 MiB από προεπιλογή). Αυτά αποτελούν τους δομικούς λίθους με την έννοια ότι όλες οι λειτουργίες (απεικονίσεις) γίνονται σε όρους αυτών των chunks. Κάθε νέο mapping λοιπόν ξεκινά από μία μετατόπιση εντός του αρχείου και εντός του DAX window, που αμφότερες είναι πολλαπλάσια του μεγέθους του chunk, ενώ αφορά ακέραιο πλήθος chunks. Γίνεται έτσι σαφές ότι απαιτώντας το μέγεθος του chunk να είναι συμβατό με το map alignment, αυτόματα ικανοποιούνται όλες οι απαιτήσεις ευθυγράμμισης που τίθενται από τη virtio-fs συσκευή. Αυτό επελέγη αφενός για απλότητα και αφετέρου ώστε να λειτουργεί (με την επιλογή του κατάλληλου μεγέθους chunk) ως μονάδα προφόρτωσης (prefetching).
- Ένα νέο mapping ξεκινά πάντοτε από τη χαμηλότερη διαθέσιμη διεύθυνση εντός του DAX window. Εάν δεν υπάρχει αρκετός χώρος (το παράθυρο είναι γεμάτο), τότε γίνεται επικάλυψη με τις απεικονίσεις στις υψηλότερες διευθύνσεις, μόνο όσο χρειάζεται ώστε να χωρέσει η νέα. Έτσι, ακολουθείται ένα LIFO (Last In - First Out) μοντέλο, με το DAX window να προσιδιάζει μία στοίβα. Αυτό έγινε αφενός για απλότητα και αφετέρου ώστε στην περίπτωση μίας εφαρμογής που διαβάζει ένα αρχείο σειριακά με διαδοχικές κλήσεις, τα επιμέρους διαδοχικά chunks να βρίσκονται διατεταγμένα στο παράθυρο.



Σχήμα 3.2: Ενδεικτική εικόνα του DAX window υπό τον manager.

- Δεν γίνεται προφόρτωση (prefetching) των αρχείων πέρα του ενός chunk, προκειμένου να αποφευχθεί δυνητική σπατάλη του χώρου στο DAX window. Παρ' αυτά, σημειώνουμε ότι στην υλοποίηση περιλαμβάνεται η επιλογή (απενεργοποιημένη από προεπιλογή) για 'επιθετικό' prefetching, όπου απεικονίζεται ολόκληρο το αρχείο με την πρώτη πρόσβαση σε αυτό.

Ολόκληρη η διαδικασία ανάγνωσης σε ότι αφορά την υλοποίηση του virtio-fs απεικονίζεται ενδεικτικά στο διάγραμμα ροής 3.3.

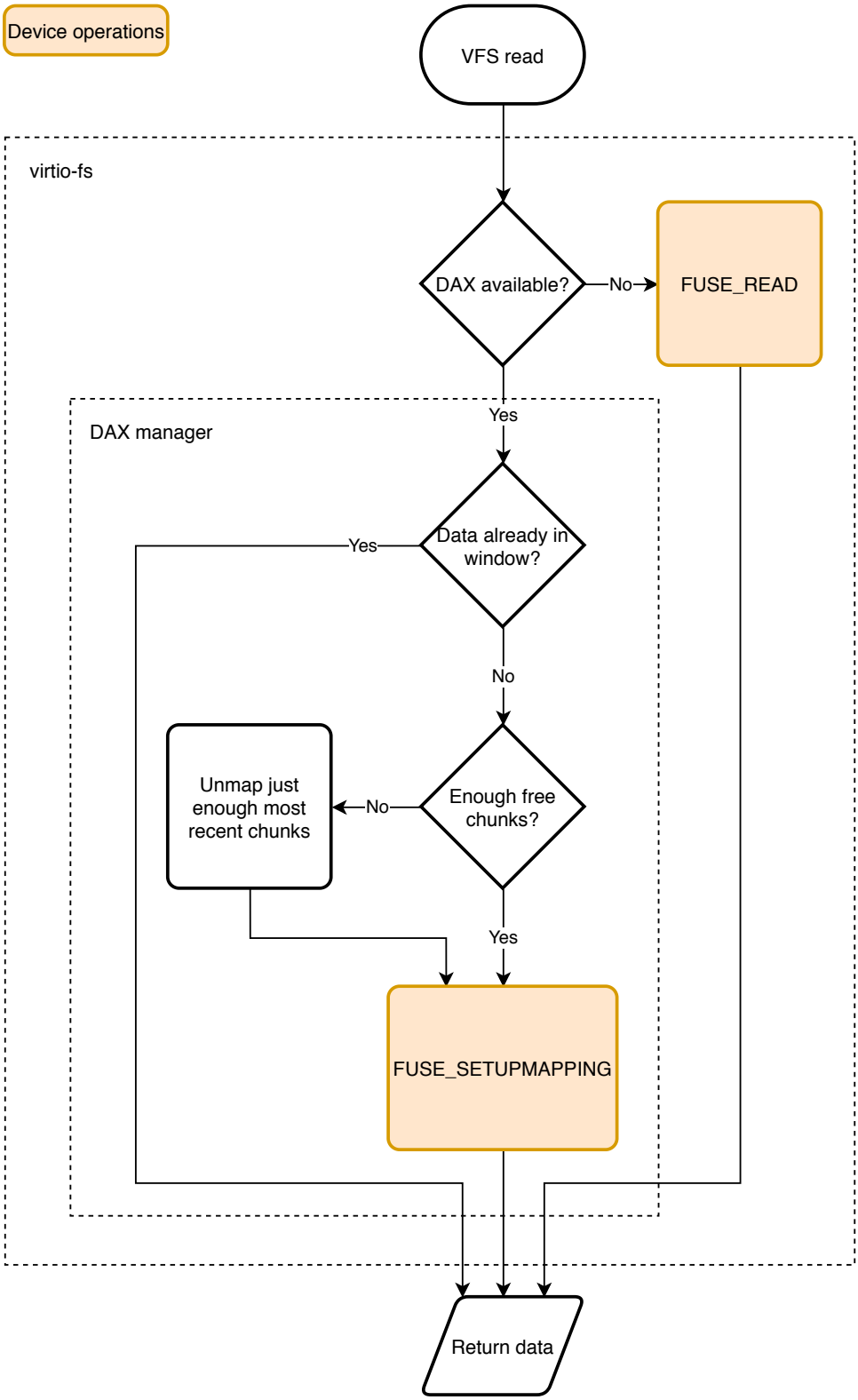
3.2 Boot από virtio-fs

Η προσθήκη υποστήριξης για εκκίνηση (boot) από το virtio-fs στο OSv είχε δύο συνιστώσες: την κυριότερη που αφορούσε τον πυρήνα του λειτουργικού συστήματος αλλά και μία βοηθητική που αφορούσε τα εργαλεία αυτοματισμού για τη δημιουργία και εκτέλεση των εικόνων. Αμφότερες βασίστηκαν σε προηγούμενες αντίστοιχες επεκτάσεις, μιας και το OSv ήδη μπορούσε να χρησιμοποιήσει είτε το ZFS είτε το rofs (και το ramfs βέβαια, που είναι ιδιαίτερη περίπτωση) για το κύριο (root) σύστημα αρχείων του.

Συνοπτικά, τα σημεία που αφορούν το root file system κατά τον κύκλο ζωής ενός OSv unikernel, ως είχαν πριν την επέκταση μας είναι:

1. Χτίσιμο της εικόνας (κεντρικό σημείο το scripts/build): σε αυτό το στάδιο επιλέγεται από τον χρήστη ο τύπος του συστήματος αρχείων και το root file system δημιουργείται με το περιεχόμενο που προκύπτει από την εφαρμογή. Επίσης, εδώ ξεκινά ο καθορισμός της εντολής εκκίνησης (command line) του (uni)kernel, η οποία περιέχει επιλογές για τον ίδιο τον πυρήνα, αλλά και το command line της εφαρμογής που πρόκειται να εκτελεστεί. Αυτή αποθηκεύεται σε ένα προσωρινό αρχείο, από όπου την παραλαμβάνει το επόμενο βήμα [39].

2. Εκτέλεση της εικόνας (κεντρικό σημείο το scripts/run.py): εδώ προαιρετικά εμπλουτίζεται το command line με εφήμερες επιλογές, για παράδειγμα το επίπεδο λεπτομέρειας (verbosity) στα μηνύματα του πυρήνα. Κατόπιν, το ολοκληρωμένο command line εγγράφεται στο image, προτού αυτό εκτελεστεί.

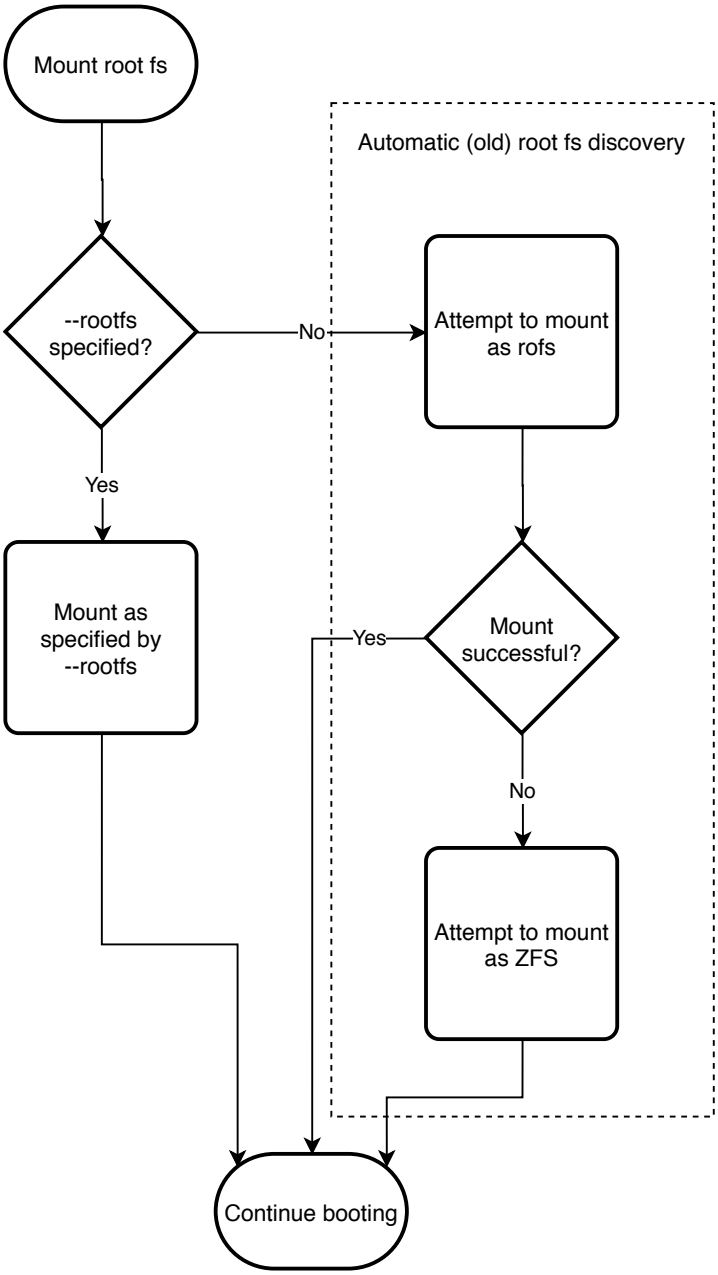


Σχήμα 3.3: Διαδικασία ανάγνωσης από το virtio-fs υπό τον DAX window manager.

3. Φόρτωση του πυρήνα (κεντρικό σημείο το loader.cc): μετά από, μεταξύ άλλων, την αρχικοποίηση του πυρήνα, των συσκευών και των οδηγών, αλλά και το διάβασμα του command line, το OSv επιχειρεί να περάσει από το αρχικό, ενσωματωμένο σύστημα αρχείων του (ramfs) στο κύριο σύστημα αρχείων (root file system) [43]. Η διαδικασία ετούτη συνίσταται στην προσάρτηση (mounting) του συστήματος αρχείων σε κάποιο σημείο του υπάρχοντος, αρχικού ramfs, ακολουθούμενη από το πέρασμα της ρίζας (pivoting) του εικονικού συστήματος αρχείων σε αυτό. Τα τελευταία γίνονται απλά, μέσω του εικονικού συστήματος αρχείων, αφού όλα τα προαπαιτούμενα για τη χρήση του έχουν ολοκληρωθεί.

Η επιλογή του συστήματος αρχείων που θα χρησιμοποιηθεί ως root κατά τη φόρτωση μέχρι πρότινος ήταν δυναμική: γινόταν απόπειρα προσάρτησης από την πρώτη συσκευή block αρχικά του rofs και εάν αυτό αποτύγχανε, του ZFS. Εάν αμφότερα αποτύγχαναν, η εκτέλεση συνεχιζόταν με το ramfs αρχικό σύστημα αρχείων. Αν και η προηγούμενη διαδικασία θα μπορούσε εύκολα να επεκταθεί ώστε να συμπεριλάβει το virtio-fs, γινόταν εμφανές ότι βασιζόταν σε πολλές σιωπηλές υποθέσεις, χωρίς να αφήνει μεγάλο περιθώριο ελέγχου στον χρήστη. Έτσι αποφασίσαμε να πάμε ένα βήμα παραπέρα, ώστε να δοθεί περισσότερος έλεγχος επί της διαδικασίας: προσθέσαμε μία επιλογή `--rootfs` στη γραμμή εντολών του πυρήνα (command line option), η οποία επιτρέπει να καθοριστεί ρητά ο τύπος του root file system (ανάμεσα σε ZFS, rofs, virtio-fs και ramfs). Όπως φαίνεται και στο διάγραμμα 3.4, αυτή η επιλογή είναι προαιρετική και εάν δεν έχει καθοριστεί χρησιμοποιείται η παλαιότερη, δυναμική διαδικασία, για συμβατότητα. Τέλος, κάναμε τις απαραίτητες αλλαγές στη διαδικασία χτισίματος (scripts/build) ώστε να θέτει αυτή την επιλογή ανάλογα με το σύστημα αρχείων που έχει καθορίσει ο χρήστης κατά το χτίσιμο. Εξάλλου, με αυτόν τον τρόπο έγινε πιο προβλέψιμη και τεκμηριώθηκε καλύτερα η διαδικασία καθώς και οι διαθέσιμες επιλογές για το root file system.

Τέλος, όσον αφορά καθεαυτό το virtio-fs, η χρήση του ως root file system δεν ενείχε κάποια αξιοσημείωτη πρόκληση: εάν επιλέγονταν μέσω του `--rootfs`, γινόταν προσάρτηση του από την πρώτη virtio-fs συσκευή (στο OSv, σε αντίθεση με το Linux, αυτές εκτίθενται στο devfs, αριθμημένες σειριακά αντί να αναγνωρίζονται από το virtio-fs tag τους). Προκειμένου να συμπληρωθεί αυτόματα με τα κατάλληλα περιεχόμενα για την εκάστοτε εφαρμογή ένας κατάλογος στον host (ο οποίος στη συνέχεια θα χρησιμοποιηθεί ως το virtio-fs shared directory), είναι διαθέσιμες οι παράμετροι `export` και `export_dir` του scripts/build.



Σχήμα 3.4: Διαδικασία προσάρτησης του root file system.

Κεφάλαιο 4

Αξιολόγηση

Για την αξιολόγηση του νέου συστήματος αρχείων στο OSv, διεξάγαμε δοκιμές συγκρίνοντας με τα υπόλοιπα διαθέσιμα συστήματα αρχείων αλλά και το virtio-fs στο Linux, όπου αυτή η σύγκριση είχε νόημα. Οι δοκιμές περιλαμβάνουν τρία σενάρια, όπως αναλύονται στη συνέχεια: ένα συνθετικό μετροπρόγραμμα (synthetic benchmark ή microbenchmark), μια δοκιμή χρόνου εκκίνησης και τέλος μία πραγματική εφαρμογή (application benchmark).

4.1 Μεθοδολογία

Όλες οι δοκιμές πραγματοποιήθηκαν σε προσωπικό υπολογιστή με τα στοιχεία που αναφέρονται στον πίνακα 4.1, ενώ ο πίνακας 4.2 περιλαμβάνει τις προδιαγραφές των OSv και Linux guests. Ως προς τη διεξαγωγή τους πρέπει να αναφερθούν τα εξής:

- Για την εκτέλεση όλων των δοκιμών χρησιμοποιήθηκε ένα προσωρινό σύστημα αρχείων (tmpfs) στον host, τόσο για τις εικόνες του OSv όσο και για τους κοινόχρηστους καταλόγους στην περίπτωση των virtio-fs και NFS. Αυτό έγινε ώστε να μην επηρεάζονται οι επιδόσεις από τη συσκευή αποθήκευσης του host.

- Η δυναμική προσαρμογή της συχνότητας της CPU ήταν απενεργοποιημένη, χρησιμοποιώντας τον "performance" CPU scaling governor στον host.

- Η διεργασία του QEMU ήταν απομονωμένη από τις υπόλοιπες (virtiofsd, perf, vegeta) ως προς τις CPUs στις οποίες εκτελούνταν (CPU pinning). Συγκεκριμένα, στο QEMU αφιερώνονταν επεξεργαστικοί πυρήνες ισάριθμοι με το πλήθος των CPUs του guest, ενώ οι υπόλοιπες προαναφερόμενες διεργασίες δεσμεύονταν στους υπόλοιπους διαθέσιμους. Αυτό έγινε με σκοπό την ελαχιστοποίηση των παρεμβολών που θα μπορούσαν να επηρεάσουν τα αποτελέσματα.

- Για τη μέτρηση της χρησιμοποιούμενης επεξεργαστικής ισχύος (CPU usage) επελέγη το perf [45], στην έκδοση 5.7.g3d77e6a8804a. Συγκεκριμένα, χρησι-

Επεξεργαστής	Intel Core i7-6700 @3.4GHz
Μνήμη	2×8 GiB @2666MHz
Swap	Όχι
Linux kernel	5.8.13-arch1-1
QEMU	5.1.50 @ c37a890d12e57a3d28c3c7ff50ba6b877f6fc2cc [54]

Πίνακας 4.1: Προδιαγραφές του host όπου διεξήχθησαν οι δοκιμές.

μοποιήθηκε η εντολή `perf stat`, με το "task-clock" perf event. Η μέτρηση έγινε για τη διεργασία του QEMU, το virtiofsd, το vhost kernel thread [23] και τα NFS kernel threads (για το καθένα όπου ήταν εφαρμόσιμο).

- Όλες οι δοκιμές επαναλήφθηκαν 10 φορές, από τις οποίες στη συνέχεια παρουσιάζονται η μέση τιμή (mean) και η τυπική απόκλιση (standard deviation). Στην περίπτωση του OSv κάθε επανάληψη ήταν μία εκ νέου εκκίνηση της εικονικής μηχανής, ενώ στο Linux όλες οι επαναλήψεις έγιναν στην ίδια εκτέλεση της εικονικής μηχανής.

- Το virtiofsd εκτελούνταν με το cache mode απενεργοποιημένο (`cache=none`) και ένα thread (`--thread-pool-size=1`). Το δεύτερο επελέγη διότι οδηγούσε σε ελαφρώς πιο συνεπείς μετρήσεις, χωρίς όμως να παρατηρείται καλύτερη επίδοση όπως είχε αναφερθεί σε συζήτηση στη mailing list του virtio-fs[1].

- Για την εκτέλεση του OSv έγινε χρήση του βοηθητικού script (scripts/run.py) που παρέχει, το οποίο τροποποιήθηκε για την διεξαγωγή των δοκιμών, ώστε να ενορχηστρώνει την εκτέλεση όλων των υπόλοιπων εργαλείων (virtiofsd, perf, vegeta). Ως προς τη δικτύωση της εικονικής μηχανής, χρησιμοποιήθηκε το tap backend του QEMU με το vhost ενεργοποιημένο, ενώ στο VM δινόταν στατική διεύθυνση IP.

- Ως NFS server χρησιμοποιήθηκε η αντίστοιχη υλοποίηση του Linux στον host. Όλες οι δοκιμές έγιναν με την έκδοση 3 του NFS, καθώς η έκδοση 4 δεν ήταν λειτουργική από την πλευρά του OSv. Το readahead στον NFS client (libnfs), αντίστοιχο του μεγέθους των chunks στην υλοποίηση μας του DAX window manager, ήταν ίσο με 2 MiB.

4.2 Microbenchmark

4.2.1 Περιγραφή

Για τη μέτρηση των επιδόσεων του συστήματος αρχείων χρησιμοποιήσαμε το flexible I/O tester (fio) [15] στην έκδοση 3.23. Προκειμένου να εκτελεστεί στο OSv

[1] `https://www.redhat.com/archives/virtio-fs/2020-September/msg00068.html`

CPUs	4
Μνήμη	4 GiB
DAX window	4 GiB
OSv	5372a230ce0abf0dc72e92ec1116208145e595c5 [44]
Linux kernel	5.8.0-rc4-33261-gfaa931f16f27 [53]

Πίνακας 4.2: Προδιαγραφές των guests.

χρειάστηκαν μόνο δύο τροποποιήσεις[2] και τα κατάλληλα ορίσματα στο configure script του fio ώστε να απενεργοποιηθούν χαρακτηριστικά που δεν παρέχονται από το OSv. Σημειώνουμε ότι το ίδιο ακριβώς εκτελέσιμο (με τα παραπάνω απενεργοποιημένα) χρησιμοποιήθηκε και στο Linux, εκτός από το OSv.

Στη σύγκριση περιλαμβάνονται τα ZFS, rofs, ramfs, virtio-fs (με και χωρίς DAX window και με ramfs root file system) και NFS στο OSv, το virtio-fs (με και χωρίς DAX window και με ext4 root file system) στο Linux, καθώς και το tmpfs στον host, το οποίο συμπεριλάβαμε για πληρότητα, ως σημείο αναφοράς (baseline). Συγκεκριμένα για τη διαδικασία έχουμε:

- Σε όλες τις περιπτώσεις εκτός του ramfs, τα αρχεία ελέγχου (test files) του fio έχουν παραχθεί εκ των προτέρων από το ίδιο και έχουν τοποθετηθεί στην εκάστοτε τοποθεσία (εικονικό δίσκο ή κοινόχρηστο κατάλογο). Αυτό γίνεται παρότι το fio μπορεί να τα παράξει δυναμικά κατά το χρόνο εκτέλεσης, διότι κάποια από τα συστήματα αρχείων είναι read-only. Επειδή το ramfs είναι περιορισμένο ως προς το μέγεθος μεμονωμένων αρχείων στο image του, στην περίπτωση του τα αρχεία παράγονται κατά το χρόνο εκτέλεσης.

- Εξετάζονται δύο περιπτώσεις ως προς τα αρχεία ελέγχου:

  - Ένα μεγάλο αρχείο (1 GiB).
  - Πολλαπλά (10) μικρότερα αρχεία (80-100 MiB). Σημειώνουμε ότι τα αρχεία σε όλες τις δοκιμές είναι πανομοιότυπα (το αντίστοιχο αρχείο έχει το ίδιο μέγεθος πάντα).

  Και στις δύο περιπτώσεις το συνολικό μέγεθος των αρχείων δεν ξεπερνά το 1 GiB. Αυτό εν μέρει γίνεται διότι το συγκεκριμένο μέγεθος είναι περιορισμένο από τη μνήμη του guest για τα rofs και ramfs και το μέγεθος του DAX window για το virtio-fs με DAX.

- Εξετάζονται δύο περιπτώσεις ως προς το πρότυπο (pattern) ανάγνωσης των αρχείων: σειριακό (`rw=read`) και τυχαίο (`rw=randread`).

- Σε όλες τις περιπτώσεις υπάρχει μόνο ένα thread ανάγνωσης (`numjobs=1`).

[2]Όλες οι αλλαγές βρίσκονται στο 'fio-3.23-osv' tag του git repository `https://github.com/foxeng/fio`.

- Στο Linux, για την αυτοματοποίηση των δοκιμών βασιστήκαμε στο σχετικό βοήθημα του Vivek Goyal[3] [22]. Το συγκεκριμένο ακυρώνει τις page, inode και dentry caches χρησιμοποιώντας το sysctl (/proc/sys/vm/drop_caches) πριν από κάθε εκτέλεση του fio.

- Στην περίπτωση του Linux (guest και host) δεν έγινε μέτρηση του CPU usage γιατί μία τέτοια σύγκριση κρίθηκε ότι δεν έχει αξία, δεδομένου του διαφορετικού χαρακτήρα του από το OSv (λειτουργικό γενικού σκοπού από τη μία και unikernel από την άλλη).

4.2.2 Αποτελέσματα

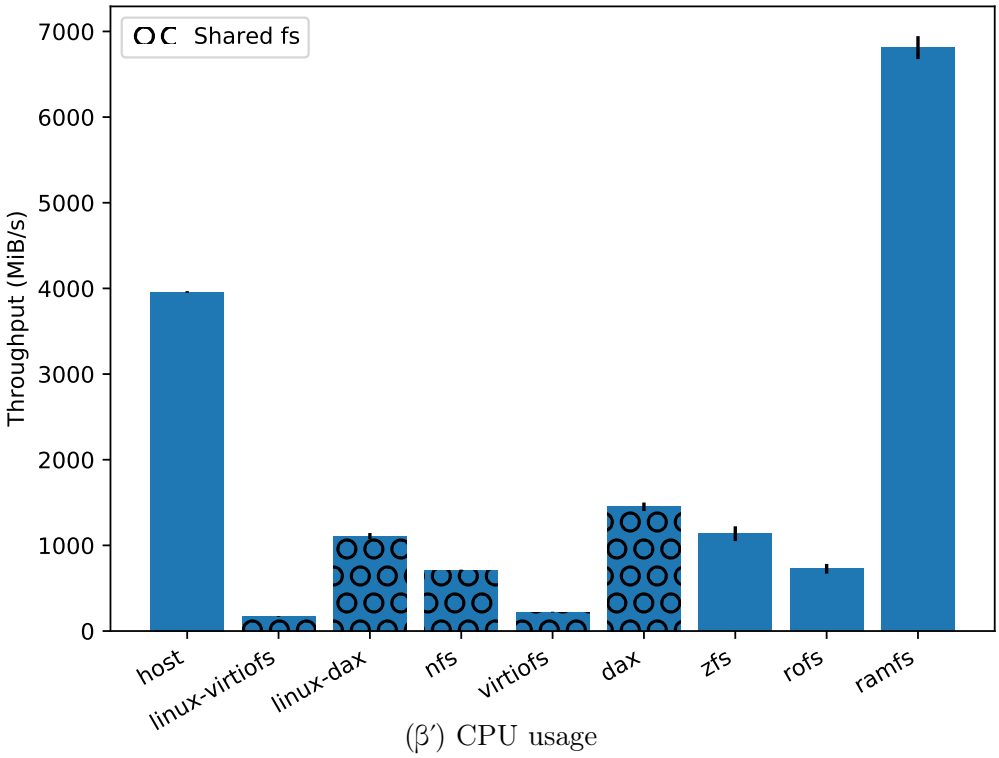
Όπως βλέπουμε στα σχήματα 4.1 έως 4.4, το υψηλότερο throughput επιτυγχάνεται από το ramfs στο OSv, υψηλότερο και από αυτό του tmpfs στον host. Αυτό συμβαίνει διότι, με τα δεδομένα στη μνήμη το virtualization overhead ελαχιστοποιείται, ενώ ταυτόχρονα η απλοποιημένη υλοποίηση του συστήματος αρχείων και η έλλειψη mode switches λόγω κλήσεων συστήματος στο OSv φαίνεται ότι κάνουν τη διαφορά. Το virtio-fs με DAX προσφέρει τις αμέσως καλύτερες επιδόσεις στο OSv, σε όλες τις περιπτώσεις, και παράλληλα έχει τη μικρότερη επιβάρυνση σε όρους επεξεργαστικής ισχύος, και πάλι μετά το ramfs.

Εστιάζοντας στο virtio-fs και συγκρίνοντας ανάμεσα σε OSv και Linux, διακρίνουμε συνεπή συμπεριφορά σε όλες τις περιπτώσεις: σε αμφότερα τα λειτουργικά το DAX window υπερτερεί του virtio-fs χωρίς αυτό με μεγάλη διαφορά ($> 6\times$ throughput), ενώ στο OSv βλέπουμε $20 - 30\%$ καλύτερες επιδόσεις από ότι στο Linux. Το δεύτερο είναι αναμενόμενο, αφενός λόγω της απλούστερης οπότε και αποδοτικότερης υλοποίησης στο OSv, που υποστηρίζει πολύ λιγότερες λειτουργίες και αφετέρου λόγω των συγκριτικών πλεονεκτημάτων ενός unikernel.

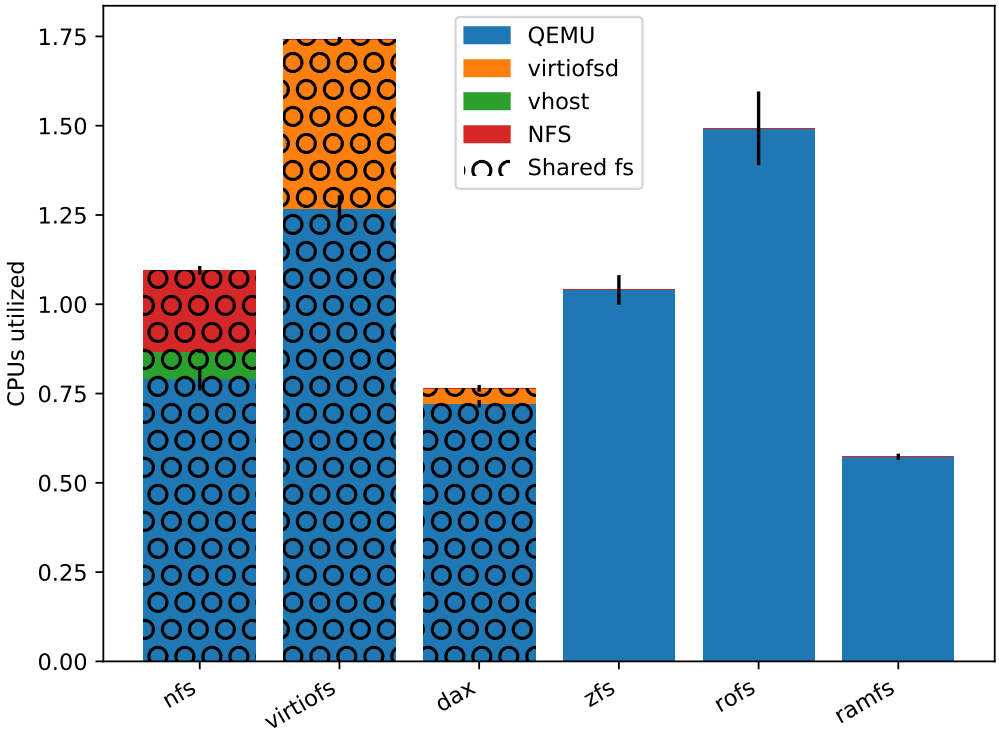
Ιδιαίτερη μνεία οφείλουμε στη σύγκριση ανάμεσα σε virtio-fs και NFS, τα μόνα κοινόχρηστα συστήματα αρχείων στο OSv. Εδώ το virtio-fs με DAX window έχει το προβάδισμα στις επιδόσεις, με το NFS να ακολουθεί και το virtio-fs χωρίς DAX να βρίσκεται τελευταίο, στις δοκιμές με σειριακή ανάγνωση. Όπως φαίνεται και στον πίνακα 4.3, στις δοκιμές τυχαίας ανάγνωσης τα αποτελέσματα διαφοροποιούνται, με το NFS να έχει το χειρότερο throughput και μεγαλύτερο CPU usage. Φαίνεται λοιπόν ότι έχει επηρεαστεί σαφώς περισσότερο σε σχέση με το virtio-fs από την αλλαγή στο random access pattern, που θέτει πίεση στους μηχανισμούς πρόβλεψης και caching όλων των συστημάτων αρχείων.

[3]commit hash 8c99f50c878cb39db76abec7e0882fd83c99f4b3

(α′) Throughput

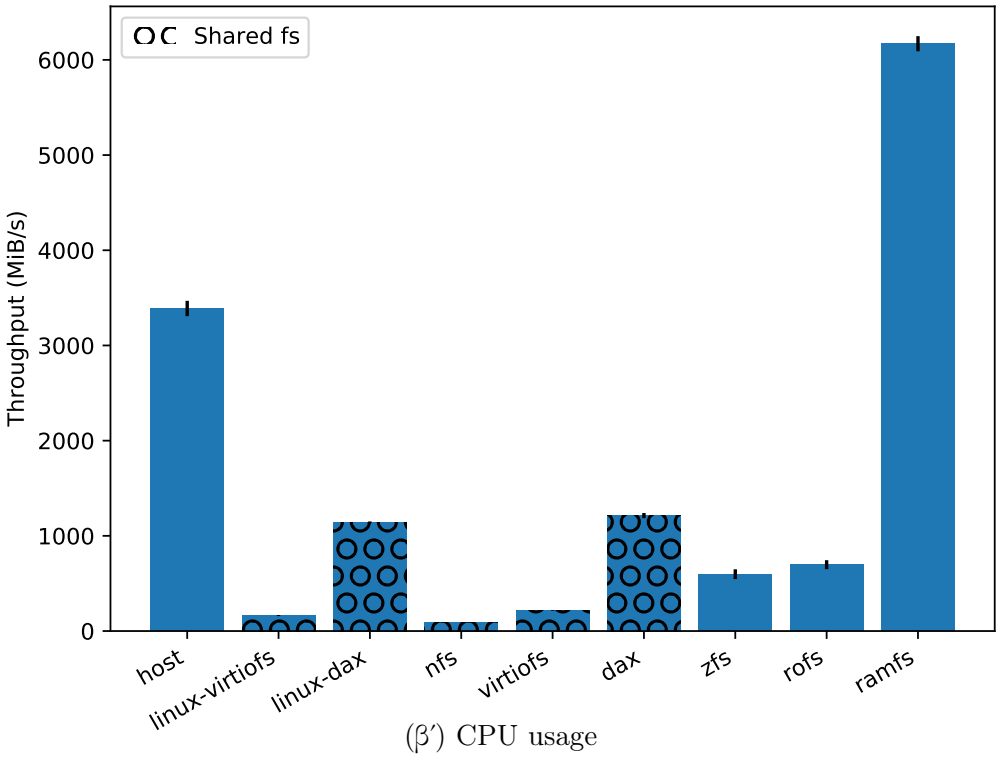


(β′) CPU usage

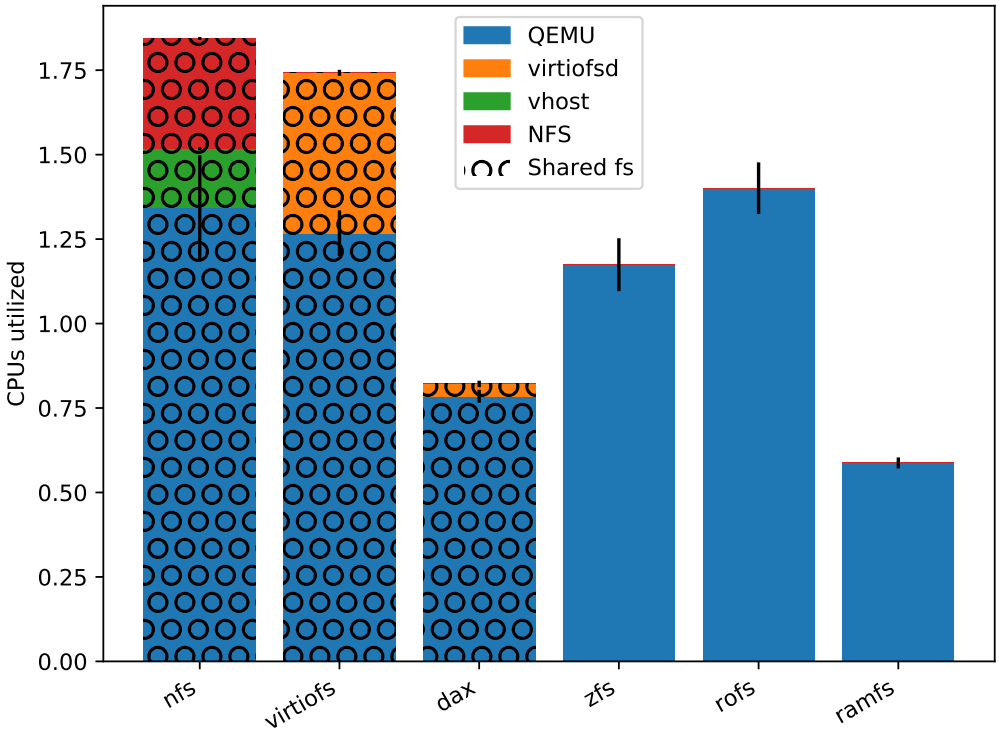


Σχήμα 4.1: fio, ένα αρχείο, σειριακή ανάγνωση

(α′) Throughput

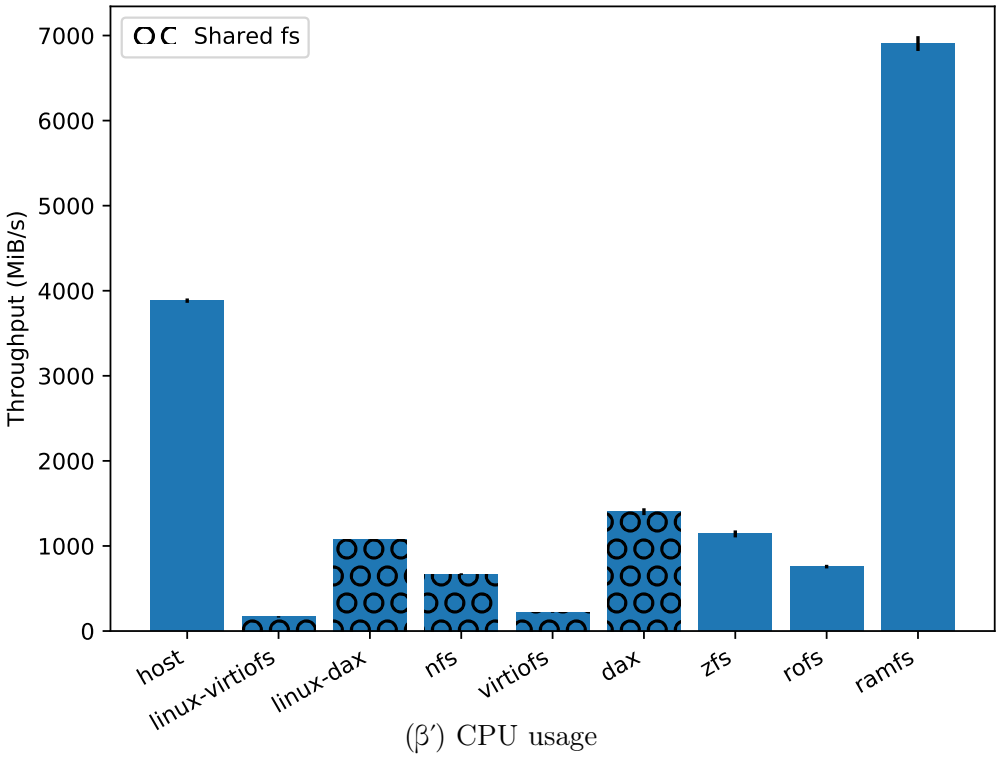


(β′) CPU usage

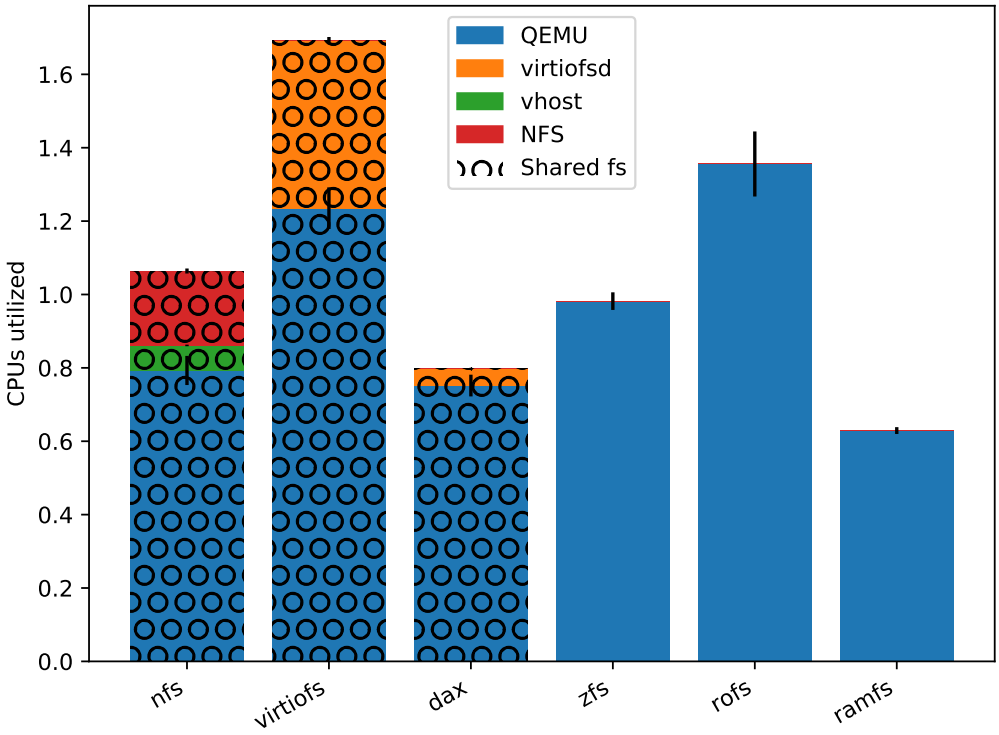


Σχήμα 4.2: fio, ένα αρχείο, τυχαία ανάγνωση

(α′) Throughput

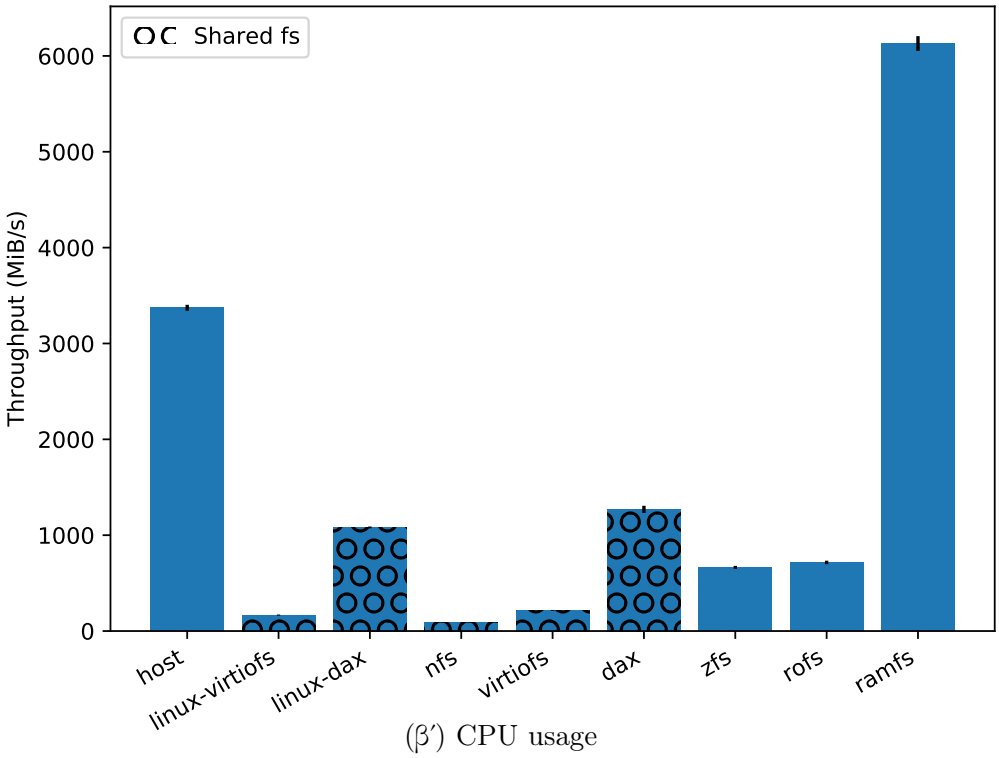


(β′) CPU usage

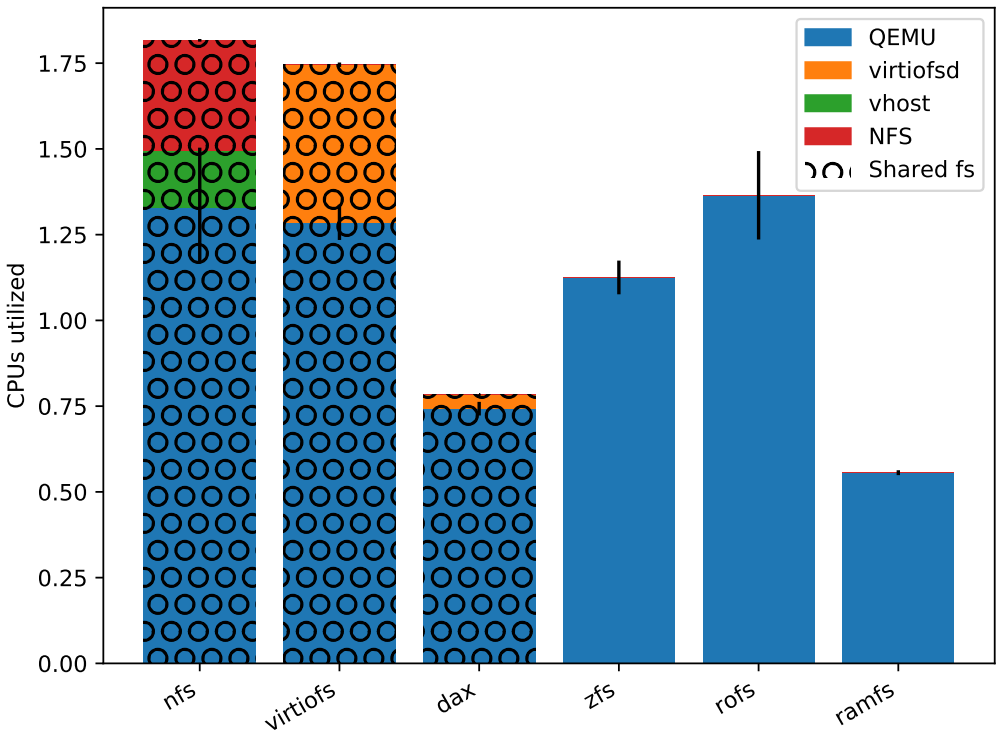


Σχήμα 4.3: fio, πολλαπλά αρχεία, σειριακή ανάγνωση

(α′) Throughput



(β′) CPU usage



Σχήμα 4.4: fio, πολλαπλά αρχεία, τυχαία ανάγνωση

pattern	virtio-fs	NFS	virtio-fs DAX
σειριακό	1,0	3,1	6,5
τυχαίο	1,0	0,4	5,7

Πίνακας 4.3: Κανονικοποιημένο fio throughput virtio-fs και NFS στο OSv.

4.3 Χρόνος εκκίνησης

4.3.1 Περιγραφή

Προκειμένου να αξιολογήσουμε τον χρόνο εκκίνησης στο virtio-fs επιλέξαμε μία εφαρμογή από αυτές που έχουν ήδη μεταφερθεί στο OSv ως παραδείγματα [41]. Συγκεκριμένα, επελέγη το spring-boot-example, μια απλή web εφαρμογή από το [25] χτισμένη με το spring boot framework, στην έκδοση του 2.3.4.[4] Ως Java runtime επιλέξαμε και πάλι από την ίδια συλλογή εφαρμογών το openjdk8-zulu-full. Η επιλογή της συγκεκριμένης εφαρμογής έγινε καθώς θεωρήθηκε αντιπροσωπευτική, ως stateless web app, εφαρμογής που θα γινόταν deploy ως unikernel, σε πλαίσιο cloud.

Στις δοκιμές συμμετείχαν όλα τα συστήματα αρχείων του OSv τα οποία μπορούν να χρησιμοποιηθούν ως root file systems: ZFS, rofs, ramfs και virtio-fs. Η διαδικασία περιελάμβανε την εκκίνηση του OSv με το εκάστοτε σύστημα αρχείων να χρησιμοποιείται ως root file system. Αυτό αφηνόταν να τρέξει για ένα εύλογο χρονικό διάστημα κάποιων δευτερολέπτων, μέσα στο οποίο ολοκληρωνόταν η πλήρης αρχικοποίηση της εφαρμογής και στη συνέχεια τερματιζόταν από το μηχανισμό (script) ενορχήστρωσης των δοκιμών, τερματίζοντας τη διεργασία του QEMU.

4.3.2 Αποτελέσματα

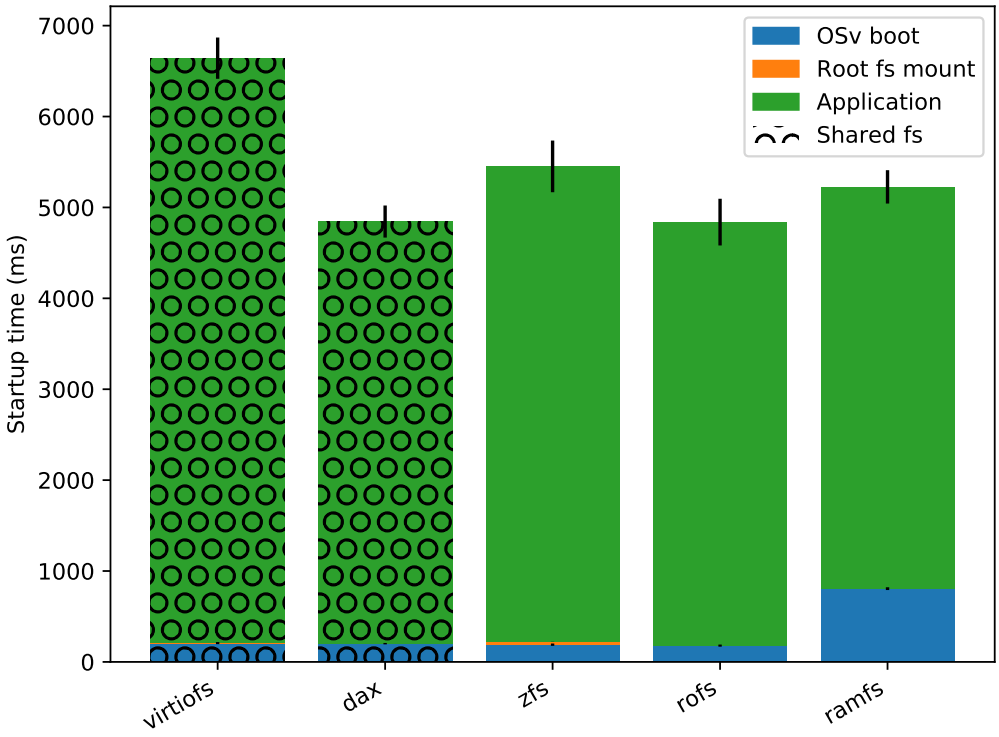
Στο σχήμα 4.5 απεικονίζεται μία ανάλυση του συνολικού χρόνου εκκίνησης ως εξής:

OSv boot είναι ο χρόνος εκκίνησης (boot) του συστήματος του OSv, δηλαδή του μέρους που είναι ανεξάρτητο της εκάστοτε εφαρμογής.

Root fs mount είναι ο χρόνος προσάρτησης (mount) του root file system και της αλλαγής της 'ρίζας' (/) του εικονικού συστήματος αρχείων σε αυτό (pivot). Επισημαίνεται ότι αυτό είναι ένα στάδιο του προηγούμενου, αλλά εν προκειμένω έχει αφαιρεθεί από εκείνο και παρουσιάζεται χωριστά.

Application είναι ο χρόνος αρχικοποίησης της ίδιας της εφαρμογής, όπως καταγράφεται από αυτήν.

[4] Όλες οι αλλαγές που έγιναν για τις δοκιμές μας βρίσκονται στο 'virtio-fs-tests' tag του git repository `https://github.com/foxeng/osv-apps`.



Σχήμα 4.5: Spring boot example, χρόνοι εκκίνησης

ZFS	rofs	ramfs	virtio-fs	virtio-fs DAX
1,13	1,00	1,08	1,37	1,00

Πίνακας 4.4: Κανονικοποιημένος συνολικός χρόνος εκκίνησης spring-boot-example στο OSv.

Όπως βλέπουμε καλύτερα στον πίνακα 4.4, με όρους συνολικού χρόνου εκκίνησης, το virtio-fs χωρίς DAX window υστερεί έναντι των υπολοίπων, ενώ το virtio-fs με DAX window έχει την καλύτερη επίδοση, μαζί με το rofs (τα ramfs και ZFS είναι ελαφρώς πιο αργά).

Όσον αφορά τον χρόνο προσάρτησης (mount time), για όλα τα συστήματα αρχείων είναι πρακτικά αμελητέος ($< 2\%$ του OSv boot time), εκτός από το ZFS. Στην περίπτωση του τελευταίου, ο χρόνος προσάρτησης αποτελεί περίπου το 14% του χρόνου εκκίνησης του συστήματος, κάτι αναμενόμενο δεδομένης της πολυπλοκότητας του συγκεκριμένου συστήματος αρχείων, η οποία μεταφράζεται σε μία σαφώς πιο μακρά διαδικασία αρχικοποίησης.

Τέλος, αξίζει να αναφερθούμε στον αισθητά αυξημένο χρόνο εκκίνησης του OSv στην περίπτωση του ramfs. Αυτή η διαφοροποίηση δικαιολογείται εάν αναλογιστούμε ότι στην περίπτωση του ramfs, το root file system ταυτίζεται με το boot file system (σε ορολογία OSv, το αντίστοιχο initramfs στο Linux). Αυτό είναι μέρος του ELF object που περιέχει το kernel, το οποίο φορτώνεται και αποσυμπιέζεται κατά τα πρώιμα στάδια του boot [42], σε μια διαδικασία με χαμηλό throughput, λόγω του περιορισμένων δυνατοτήτων του αρχικού περιβάλλοντος κατά την εκκίνηση με BIOS στην αρχιτεκτονική x86. Έτσι, όταν στην περίπτωση του ramfs, το εν λόγω ELF object είναι έως και μία τάξη μεγέθους μεγαλύτερο από τα υπόλοιπα, αυτό είναι υπαίτιο για την αύξηση του χρόνου εκκίνησης. Βέβαια, εν προκειμένω έχουμε κάνει κατάχρηση του ramfs, το οποίο δεν είναι προορισμένο για τόσο μεγάλα images.

4.4 Application benchmark

4.4.1 Περιγραφή

Για να αξιολογήσουμε το virtio-fs με όρους μιας ολοκληρωμένης, πραγματικής εφαρμογής επιλέξαμε και πάλι από το πεδίο των stateless εφαρμογών σε πλαίσιο cloud το σενάριο ενός στατικού web εξυπηρετητή (server). Συγκεκριμένα, χρησιμοποιήσαμε τον nginx [51] (στην έκδοση του 1.19.2), έναν από τους πλέον δημοφιλείς web servers ελεύθερου λογισμικού, ο οποίος είναι επίσης διαθέσιμος στη συλλογή εφαρμογών του OSv.[5]

Στις δοκιμές συμμετείχαν όλα τα συστήματα αρχείων του OSv (στην περίπτωση του virtio-fs, ως root file system χρησιμοποιήθηκε το ramfs) εκτός από το NFS, με το

[5]Όλες οι αλλαγές που έγιναν για τις δοκιμές μας βρίσκονται στο 'virtio-fs-tests' tag του git repository `https://github.com/foxeng/osv-apps`.

οποίο το σενάριο μας αποτύγχανε. Συγκεκριμένα, κατά την εξυπηρέτηση του πρώτου αιτήματος (request) από τον server, μετά την αποστολή λίγων αρχικών δεδομένων της απάντησης (response), η διαδικασία σταματούσε και ο guest φαινόταν 'παγωμένος'. Αυτό διαπιστώθηκε ότι δεν οφειλόταν στον nginx, καθώς την ίδια ακριβώς συμπεριφορά επιδείκνυε το σύστημα με τον lighttpd (επίσης περιλαμβάνεται στη συλλογή του 'osv-apps') στη θέση του. Περαιτέρω διερεύνηση απαιτείται για να εντοπιστεί και πιθανώς να διορθωθεί η αιτία, η οποία από την εμπειρία μας φαίνεται να μοιάζει με κάποιου είδους αδιέξοδο (deadlock) που πιθανώς να εντοπίζεται στην υλοποίηση του NFS ή της στοίβας δικτύωσης του OSv.

Τα αρχεία τα οποία εξέθετε ο web server ήταν συνολικά 10, με μέγεθος που κυμαίνονταν από περίπου 500 KiB μέχρι και 12 MiB. Το μέγεθος κάθε αντίστοιχου αρχείου ήταν σταθερό σε όλες τις δοκιμές.

Για την διεξαγωγή των δοκιμών, που είχαν τη μορφή δοκιμών φόρτου HTTP (HTTP load tests), από τη μεριά του πελάτη (HTTP client) χρησιμοποιήσαμε το vegeta [50], ένα δημοφιλές εργαλείο γι' αυτό το σκοπό, στην έκδοση του 12.8.3. Η διαδικασία (αυτοματοποιημένη από το script ενορχήστρωσης) είχε ως εξής:

1. Αρχικά εκκινούνταν ο OSv guest, στον οποίο δίνονταν περιθώριο ενός δευτερολέπτου προκειμένου να ολοκληρωθεί η αρχικοποίηση του nginx server.

2. Στον host εκκινούσε το vegeta, ρυθμισμένο να παράγει HTTP 1.1 GET requests για όλα τα αρχεία του server, με το μέγιστο δυνατό ρυθμό, χρησιμοποιώντας 20 'εργάτες' και έως 4 CPUs, με την επαναχρησιμοποίηση (keepalive) των συνδέσεων TCP ενεργοποιημένη.

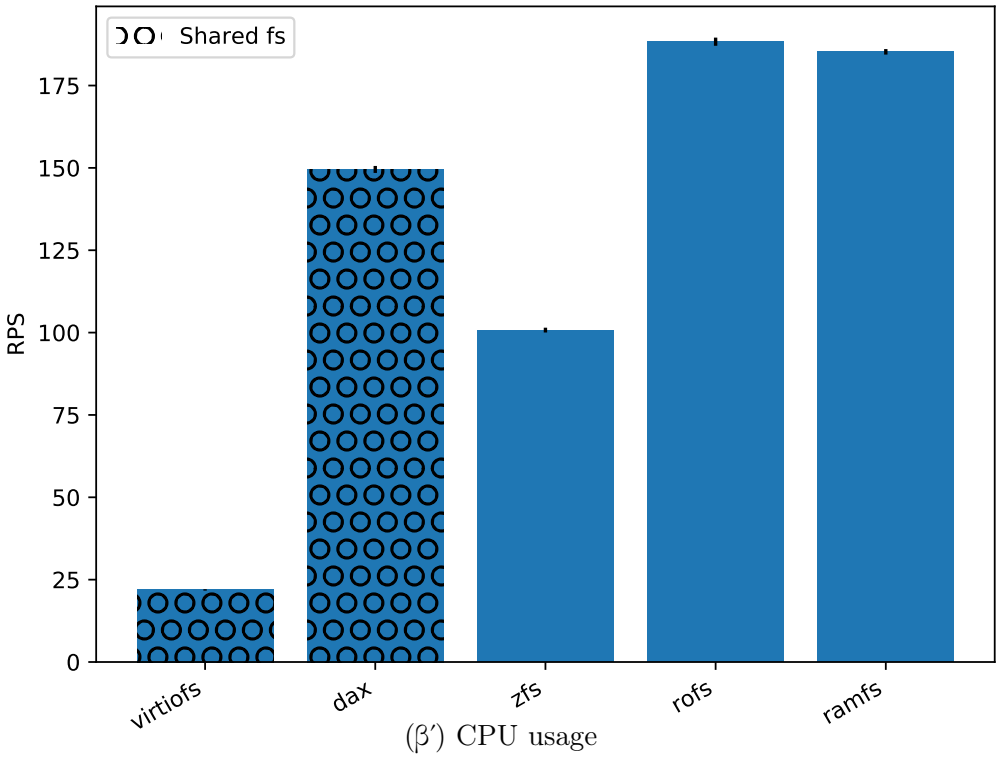
3. Μετά από δέκα δευτερόλεπτα, το vegeta ολοκλήρωνε το έργο του και τερματιζόταν, οπότε και ο αυτοματισμός τερμάτιζε τον guest μέσω της διεργασίας του QEMU.

4.4.2 Αποτελέσματα

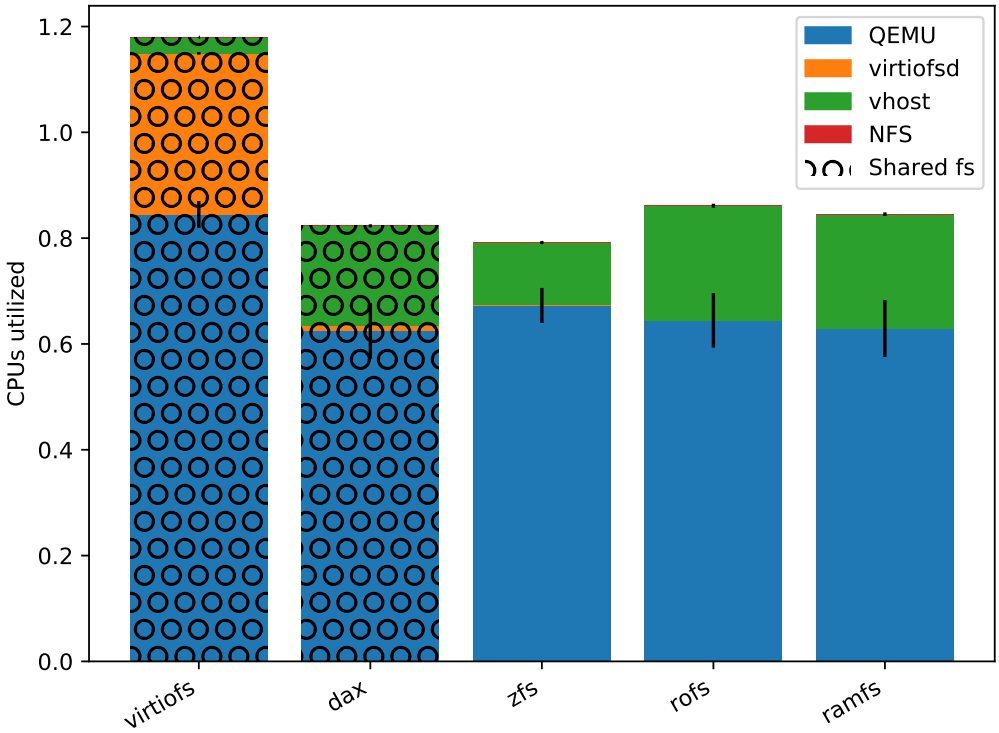
Όπως βλέπουμε στο σχήμα 4.6α', το virtio-fs με DAX window προσφέρει throughput (σε όρους εξυπηρετούμενων αιτημάτων ανά δευτερόλεπτο) που υπολείπεται αυτού των rofs και ramfs (τα οποία προηγούνται) κατά $\sim 20\%$. Δεδομένου ότι το CPU usage είναι οριακά χαμηλότερο από των άλλων δύο, αυτή η διαφορά χρήζει περαιτέρω μελλοντικής διερεύνησης (μέσω profiling), με εστίαση στην υλοποίηση του virtio-fs με DAX read datapath στο OSv, ως κύριο ύποπτο.

Όπως είναι αναμενόμενο σε αυτό το σενάριο χρήσης, το virtio-fs χωρίς DAX υστερεί με διαφορά έναντι όλων των υπόλοιπων συστημάτων αρχείων. Αυτό διότι είναι το μόνο που δεν χρησιμοποιεί οποιασδήποτε μορφής κρυφή μνήμη (cache), οπότε κάθε λειτουργία ανάγνωσης συνεπάγεται έξοδο και επεξεργασία στον host (όπως φαίνεται και από το υψηλό CPU usage), ενώ αυτές οι λειτουργίες είναι πολύ συχνές, πολλαπλασιάζοντας τις συνέπειες αυτού του overhead.

(α′) Requests ανά δευτερόλεπτο



(β′) CPU usage



Σχήμα 4.6: nginx HTTP load test

Κεφάλαιο 5

Συμπεράσματα και μελλοντικές επεκτάσεις

Στο πλαίσιο της παρούσας εργασίας επεκτείναμε επιτυχώς την υλοποίηση του virtio-fs στο OSv ώστε να μπορεί να χρησιμοποιήσει το DAX window (μόνο για ανάγνωση) και να εκκινήσει με virtio-fs root file system. Για το πρώτο χρειάστηκε να προσθέσουμε έναν διαχειριστή (manager) για το DAX window, προκειμένου να επιτύχουμε αποδοτική χρήση του υπό την παρουσία πολλαπλών απεικονίσεων. Επίσης, αξιολογήσαμε την υλοποίηση μας σε διαφορετικά σενάρια, συμπεραίνοντας ότι βελτιώνει πολύ σημαντικά τις επιδόσεις του virtio-fs, οι οποίες στις περισσότερες περιπτώσεις καθίστανται με αυτό ανταγωνιστικές των αποκλειστικών συστημάτων αρχείων στο OSv και πάντα καλύτερες από του NFS, του μόνου άλλου διαθέσιμου κοινόχρηστου συστήματος αρχείων σε αυτό. Τέλος, είδαμε στην πράξη την αξία του κοινόχρηστου συστήματος αρχείων, με το εκ νέου χτίσιμο του unikernel να γίνεται περιττό στις περισσότερες περιπτώσεις αλλαγών στην εφαρμογή.

Η εμπειρία χρήσης και ανάπτυξης στο OSv ήταν πολύ ενδιαφέρουσα, με κάποια μεγάλα πλεονεκτήματα και κάποια σημεία που μπορούν να βελτιωθούν. Το κυριότερο πλεονέκτημα απορρέει από τη χρήση της C++, σε συνδυασμό με τη φύση του library OS και τη σημαντική δουλειά που έχει γίνει ήδη στον πυρήνα του OSv: συχνά κατά την ανάπτυξη έχει κανείς την εντύπωση ότι γράφει κώδικα εφαρμογής και όχι κώδικα συστήματος, με παροχές όπως δυναμική και ημιαυτόματη (μέσω smart pointers) διαχείριση μνήμης και πρότυπες βιβλιοθήκες διαθέσιμες, όπως στις συνήθεις εφαρμογές. Ο ιδιόμορφος κύκλος ανάπτυξης επίσης συγκρίνεται περισσότερο με αυτόν μίας εφαρμογής σε γενικού σκοπού λειτουργικό, χάρη στους σύντομους χρόνους χτισίματος και εκκίνησης. Το debugging αποτελεί παραδοσιακά πρόκληση στα unikernels, όμως στην εμπειρία μας, το OSv παρέχει πολύτιμη υποστήριξη γι΄ αυτό [40] και βέβαια η σχετική ευκολία προγραμματισμού βοηθάει στο να αποφευχθούν λάθη εξ΄ αρχής. Η τεκμηρίωση είναι ένας τομέας που σαφώς θα μπορούσε να είναι καλύτερος, μιας και πολύ συχνά ο μόνος τρόπος να καταλάβει κανείς πως λειτουργεί κάτι είναι να διαβάσει τον αντίστοιχο κώδικα, κάτι που δύναται να αποθαρρύνει δυνητικούς νέους

χρήστες. Αυτό φυσικά είναι ένα συχνό πρόβλημα σε έργα λογισμικού με πολύ λίγους συνεισφέροντες (contributors) και αρκεί συνειδητή προσπάθεια για να αλλάξει.

Στην μη-τεχνική πτυχή της εργασίας, είχαμε την ευκαιρία να συνεισφέρουμε χρήσιμες προσθήκες σε ένα έργο ελεύθερου λογισμικού, ενώ ήρθαμε σε επαφή και γίναμε μέρος της κοινότητας δύο τέτοιων έργων, συμμετείχαμε στις διαδικασίες και γνωρίσαμε τις ροές εργασίας (workflows) τους. Έτσι επιτύχαμε έναν εξ΄ αρχής στόχο, αυτόν της ανταπόδοσης στην κοινότητα του έργου στο οποίο βασιστήκαμε.

Αμφότερες οι κοινότητες με τις οποίες ήρθαμε σε επαφή, αν και διαφορετικές σε μέγεθος και ενασχόληση των περισσότερων μελών τους (μικρή και καθαρά εθελοντική στο OSv, μεγαλύτερη και επαγγελματική στο virtio-fs) είχαν βασικά κοινά στοιχεία. Αυτά δεν ήταν άλλα από την ανοιχτότητα, την υποδοχή, ενθάρρυνση και υποστήριξη νέων μελών και τη διαθεσιμότητα και προθυμία για συλλογική ανάπτυξη μέσω συζήτησης, σχολιασμού, καλοπροαίρετης κριτικής και επιβράβευσης.

Όσον αφορά επεκτάσεις της δουλειάς μας στο OSv, μία που κρίνεται ιδιαίτερα ενδιαφέρουσα είναι η απεικόνιση των αρχείων στη μνήμη της εφαρμογής μέσω `mmap()`. Η πρόκληση εδώ είναι να δοθεί ουσιαστικά πρόσβαση στο DAX window διατηρώντας τη σημασιολογία της `mmap()`, εξαλείφοντας έτσι την αντιγραφή που γίνεται αναπόφευκτα από το DAX window προς το buffer της εφαρμογής όταν χρησιμοποιείται η `read()`. Η μεγαλύτερη δυνατή επέκταση βεβαίως θα ήταν η υποστήριξη εγγραφών, με ή χωρίς το DAX window, ώστε το σύστημα αρχείων αν είναι πλέον read-write. Εκτιμάται ότι αυτό θα απαιτούσε σημαντική προσπάθεια για να γίνει ικανοποιητικά, ενώ η αξία που θα προσέθετε στις περιπτώσεις χρήσης του OSv δεν πείθει για την αναγκαιότητα του προς το παρόν.

Appendix A

FUSE copyright notice

Βιβλιογραφία

[1] *9P website*. http://9p.cat-v.org/, accessed 30/10/2020.

[2] *Docker website*. https://www.docker.com/, accessed 30/10/2020.

[3] *DPDK website*. https://www.dpdk.org/, accessed 30/10/2020.

[4] *FUSE*. https://www.kernel.org/doc/html/latest/filesystems/fuse.html, accessed 30/10/2020.

[5] *How are serverless computing and Platform-as-a-Service different?* https://www.cloudflare.com/learning/serverless/glossary/serverless-vs-paas/, accessed 30/10/2020.

[6] *IncludeOS website*. https://www.includeos.org/, accessed 30/10/2020.

[7] *Nanos website*. https://nanos.org/, accessed 30/10/2020.

[8] *Prex website*. http://prex.sourceforge.net/, accessed 30/10/2020.

[9] *SPDK website*. https://spdk.io/, accessed 30/10/2020.

[10] *Toro Kernel website*. https://torokernel.io/, accessed 30/10/2020.

[11] *Use bind mounts*. https://docs.docker.com/storage/bind-mounts/, accessed 30/10/2020.

[12] *Vhost-user Protocol*. https://www.qemu.org/docs/master/interop/vhost-user.html, accessed 30/10/2020.

[13] Adams, Keith και Ole Agesen: *A comparison of software and hardware techniques for x86 virtualization*. Στο *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, σελίδα 2–13, New York, NY, USA, 2006. Association for Computing Machinery, ISBN 1595934510. https://doi.org/10.1145/1168857.1168860.

[14] Αγαςηε, Αλεξανδρυ, Μαρς Βροοκερ, Αλεξανδρα Ιορδαςηε, Αντηονψ Λιγυορι, Ρολφ Νευγεβαυερ, Πηιλ Πιωονκα, και Διανα Μαρια Ποπα: *Φιρεςραςκερ: Λιγητωειγητ ΐρτυαλιζατιον φορ Σερερλεσς Απλιςατιονς.* Στο *17τη ΥΣΕΝΙΞ Σψμποσιυμ ον Νετωορκεδ Σψστεμς Δεσιγν ανδ Ιμπλεμεντατιον (ΝΣΔΙ 20)*, σελίδες 419–434, Σαντα Ϋλαρα, ΫΑ, Φεβρουάριος 2020. ΥΣΕΝΙΞ Ασσοςιατιον, ΙΣΒΝ 978-1-939133-13-7. https://www.usenix.org/conference/nsdi20/presentation/agache.

[15] Axboe, Jens και fio contributors: *Flexible I/O tester repository.* https://github.com/axboe/fio.

[16] Barham, Paul, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, και Andrew Warfield: *Xen and the art of virtualization.* Στο *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, σελίδα 164–177, New York, NY, USA, 2003. Association for Computing Machinery, ISBN 1581137575. https://doi.org/10.1145/945445.945462.

[17] Bellard, F.: *Qemu, a fast and portable dynamic translator.* Στο *USENIX Annual Technical Conference, FREENIX Track*, 2005.

[18] Castro, Paul, Vatche Ishakian, Vinod Muthusamy, και Aleksander Slominski: *The rise of serverless computing.* Commun. ACM, 62(12):44–54, Νοέμβριος 2019, ISSN 0001-0782. https://doi.org/10.1145/3368454.

[19] Cloud Hypervisor contributors: *Cloud Hypervisor repository.* https://github.com/cloud-hypervisor/cloud-hypervisor.

[20] Eismann, Simon, Joel Scheuner, Erwin Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, και Alexandru Iosup: *A review of serverless use cases and their characteristics spec rg cloud working group.* Τεχνική αναφορά, Μάιος 2020.

[21] Engler, D. R., M. F. Kaashoek, και J. O'Toole: *Exokernel: An operating system architecture for application-level resource management.* Στο *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, σελίδα 251–266, New York, NY, USA, 1995. Association for Computing Machinery, ISBN 0897917154. https://doi.org/10.1145/224056.224076.

[22] Goyal, Vivek: *virtiofs-tests repository.* https://github.com/rhvgoyal/virtiofs-tests.

[23] Hajnoczi, Stefan: *QEMU Internals: vhost architecture.* https://blog.vmsplice.net/2011/09/qemu-internals-vhost-architecture.html, accessed 24/10/2020.

[24] Hajnoczi, Stefan: *Requirements for out-of-process device emulation.* `https://blog.vmsplice.net/2020/10/requirements-for-out-of-process-device.html`, accessed 30/10/2020.

[25] in28minutes contributors: *spring-boot-examples repository.* `https://github.com/in28minutes/spring-boot-examples`.

[26] Jujjuri, Venkateswararao, Eric Van Hensbergen, Anthony Liguori, και Badari Pulavarty: *Virtfs - a virtualization aware file system pass-through.* Στο *Proceedings of the Linux Symposium*, σελίδες 109–120, 2010.

[27] Kivity, Avi, Yaniv Kamay, Dor Laor, Uri Lublin, και Anthony Liguori: *kvm: the linux virtual machine monitor.* Στο *Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07*, σελίδες 225–230, 2007.

[28] Kivity, Avi, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, και Vlad Zolotarov: *Osv: Optimizing the operating system for virtual machines.* Στο *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, σελίδα 61–72, USA, 2014. USENIX Association, ISBN 9781931971102.

[29] Lankes, Stefan, Simon Pickartz, και Jens Breitbart: *Hermitcore: A unikernel for extreme scale computing.* Στο *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '16, New York, NY, USA, 2016. Association for Computing Machinery, ISBN 9781450343879. `https://doi.org/10.1145/2931088.2931093`.

[30] libnfs contributors: *libnfs repository.* `https://github.com/sahlberg/libnfs`.

[31] The Linux man-pages project: *mmap(2) Linux Programmer's Manual*, 5.08 έκδοση. `https://man7.org/linux/man-pages/man2/mmap.2.html`.

[32] Madhavapeddy, Anil, Thomas Leonard, Magnus Skjegstad, Thomas Gazagnaire, David Sheets, Dave Scott, Richard Mortier, Amir Chaudhry, Balraj Singh, Jon Ludlam, Jon Crowcroft, και Ian Leslie: *Jitsu: Just-in-time summoning of unikernels.* Στο *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, σελίδες 559–573, Oakland, CA, Μάιος 2015. USENIX Association, ISBN 978-1-931971-218. `https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/madhavapeddy`.

[33] Madhavapeddy, Anil, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, και Jon Crowcroft: *Unikernels: Library operating systems for the cloud.* Στο *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, σελίδα 461–472, New York, NY, USA, 2013. Association for Computing Machinery, ISBN 9781450318709. `https://doi.org/10.1145/2451116.2451167`.

[34] Manco, Filipe, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, και Felipe Huici: *My vm is lighter (and safer) than your container*. Στο *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, σελίδα 218–233, New York, NY, USA, 2017. Association for Computing Machinery, ISBN 9781450350853. https://doi.org/10.1145/3132747.3132763.

[35] Mell, Peter M. και Timothy Grance: *Sp 800-145. the nist definition of cloud computing*. Τεχνική αναφορά, Gaithersburg, MD, USA, 2011.

[36] Nabla Containers contributors: *memfsd repository*. https://github.com/nabla-containers/qemu-virtiofs.

[37] Olivier, Pierre, Daniel Chiba, Stefan Lankes, Changwoo Min, και Binoy Ravindran: *A binary-compatible unikernel*. Στο *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE 2019, σελίδα 59–73, New York, NY, USA, 2019. Association for Computing Machinery, ISBN 9781450360203. https://doi.org/10.1145/3313808.3313817.

[38] OSDev contributors: *PCI*. https://wiki.osdev.org/index.php?title=PCI&oldid=25084, accessed 24/10/2020.

[39] OSv contributors: *Components of OSv*. https://github.com/cloudius-systems/osv/wiki/Components-of-OSv, accessed 24/10/2020.

[40] OSv contributors: *Debugging OSv*. https://github.com/cloudius-systems/osv/wiki/Debugging-OSv, accessed 30/10/2020.

[41] OSv contributors: *osv-apps repository*. https://github.com/cloudius-systems/osv-apps.

[42] OSv contributors: *OSv early boot (MBR)*. https://github.com/cloudius-systems/osv/wiki/OSv-early-boot-(MBR), accessed 18/10/2020.

[43] OSv contributors: *OSv lzloader and early loader*. https://github.com/cloudius-systems/osv/wiki/OSv-lzloader-and-early-loader, accessed 24/10/2020.

[44] OSv contributors: *OSv repository*. https://github.com/cloudius-systems/osv.

[45] perf contributors: *Perf Wiki*. https://perf.wiki.kernel.org, accessed 18/10/2020.

[46] Popek, Gerald J. και Robert P. Goldberg: *Formal requirements for virtualizable third generation architectures*. Commun. ACM, 17(7):412–421, Ιούλιος 1974, ISSN 0001-0782. https://doi.org/10.1145/361011.361073.

[47] Raza, Ali, Parul Sohal, James Cadden, Jonathan Appavoo, Ulrich Drepper, Richard Jones, Orran Krieger, Renato Mancuso, και Larry Woodman: *Unikernels: The next stage of linux's dominance*. Στο *Proceedings of the Workshop on Hot Topics in Operating Systems*, σελίδες 7–13. ACM, 2019.

[48] RumpRun contributors: *RumpRun repository*. `https://github.com/rumpkernel/rumprun`.

[49] Σανδβεργ, Ρυσσελ, Δαιδ Γολδβεργ, Στεε Κλειμαν, Δαν Ωαλση, και Βοβ Λψον: *Δεσιγν ανδ Ιμπλεμεντατιον ορ τηε Συν Νετωορκ Φιλεσψστεμ*, 1985.

[50] Senart, Tomás και vegeta contributors: *vegeta repository*. `https://github.com/tsenart/vegeta`.

[51] Sysoev, Igor και nginx contributors: *nginx*. `http://nginx.org`.

[52] Tsirkin, Michael S. και Cornelia Huck: *Virtual I/O Device (VIRTIO) working draft*. `https://github.com/oasis-tcs/virtio-spec/commit/af6b93bfd9a0ea1b19147e5357d4434b5075b3c8`.

[53] virtio-fs contributors: *virtio-fs linux repository*. `https://gitlab.com/virtio-fs/linux`.

[54] virtio-fs contributors: *virtio-fs qemu repository*. `https://gitlab.com/virtio-fs/qemu`.

[55] virtio-fs contributors: *virtio-fs website*. `https://virtio-fs.gitlab.io/`, accessed 30/10/2020.

[56] virtio-win contributors: *virtio-win repository*. `https://github.com/virtio-win/kvm-guest-drivers-windows`.

[57] Wikipedia contributors: *PCI configuration space — Wikipedia, the free encyclopedia*. `https://en.wikipedia.org/w/index.php?title=PCI_configuration_space&oldid=976450463`, accessed 24/10/2020.

[58] Wikipedia contributors: *Cloud computing — Wikipedia, the free encyclopedia*, 2020. `https://en.wikipedia.org/w/index.php?title=Cloud_computing&oldid=985719602`, accessed 30/10/2020.

[59] Wikipedia contributors: *Hardware virtualization — Wikipedia, the free encyclopedia*, 2020. `https://en.wikipedia.org/w/index.php?title=Hardware_virtualization&oldid=964209293`, accessed 30/10/2020.

[60] Wikipedia contributors: *Hypervisor — Wikipedia, the free encyclopedia*, 2020. `https://en.wikipedia.org/w/index.php?title=Hypervisor&oldid=985571176`, accessed 30/10/2020.

[61] Wikipedia contributors: *Os-level virtualization — Wikipedia, the free encyclopedia*, 2020. https://en.wikipedia.org/w/index.php?title=OS-level_virtualization&oldid=983612845, accessed 30/10/2020.