Το θέμα της εργασίας είναι η “υλοποίηση της μνήμης utmem σε περιβάλλον unikernel”@

Η εργασία εκπονήθηκε στο ΕΥΣ. Ο επιβλέπων είναι ο κύριος γκούμας γεώργιος, αναπληρωτής καθηγητής @

**Εισαγωγή**

Θα αναφερθώ σύντομα σε σύγχρονες τεχνολογίες και ορολογίες ώστε να γίνει κατανοητό το γενικό πλαίσιο της εργασίας.@

**Cloud computing**

Το cloud comp είναι ένας γενικός όρος που καλύπτει ένα ευρύ φάσμα εφαρμογών@

Προσφορά υπολογιστικών πόρων ή και υπηρεσιών μέσω του internet,. Κύριο χαρακτηριστικό οτι ο χρήστης βρίσκεται σε απομακρυσμένη θέση από το σύστημα που προσφέρει την ανώτερη υπηρεσία.@

Ουσιαστικά είναι η πλέον σύγχρονη τάση να επιτελούμε υπολογιστικά tasks.@

Εχει επικρατήσει λόγων των πλεονεκτημάτων σε

* Ασφάλεια.
* Ευκολία κλιμακωσιμότητας των συστημάτων
* Χαμηλά κόστη λειτουργίας@

**Εικονικοποίηση 1**

@Το cloud στηρίζεται σε πάρα πολύ μεγάλο βαθμό στην τεχνολογία της εικονικοποίηση.@

Εικονικοποίηση: Εκτέλεση ενός εικονικού συστήματος ή η προσφορά ενός εικονικού πόρου. Χρησμιοποιούμε την λέξη «εικονικό» καθώς ο πορος ή το σύστημα πιθανότατα δεν υπάρχει σε φυσική μορφή στο υφιστάμενο hardware.@

Υπάρχουν διάφορα είδη. Εμείς ασχολούμαστε μόνο με την **εικονικοποίηση σε επίπεδο συστήματος**. Όπου εικονικοποιείται το σύνολο του hardware ενός υπολ. συστήματος.@

Ένας άλλο όρος που θα ακουστεί στην συνέχεια είναι η εικονική μηχανή η VM. Δηλαδή **ένα στιγμιότυπο ενός εικονικοποιημένου συστήματος**, Περιέχει τα πάτνα από το υφιστάμενο hardware, το λειτουργικό σύστημα έως και τα προγράμμάτα πάνω από το λειτουργικό σύστημα.@

**Εικονικοποίηση 2**

Ακόμα λίγη ορολογία γύρω από την εικονικοποίηση@

* Guest, το λειτουργικό που εικονοποιείται, ή συνήθως και ολόκληρη η εικονική μηχανή.@
* Hypervisor ή επόπτης, το κομμάτι λογισμικού που παρακολουθεί και επιτρέπει την εκτέλεση του guest@
* Αν ο hypervisor παρεμβάλετε ανάμεσα στο υλικο-guest τύπου Ι. Αν μέρος άλλου λειτουργικού τότε χαρακτηρίζεται τύπου ΙΙ, ενώ @ το άλλο λειτουργικό ονομάζεται host@

Συνήθως οι όροι host και hypervisor χρησιμοποιούνται ο ένας στην θέση του άλλου, με την έννοια της οντότητας που βρίσκεται ‘κατω’ από την εικονική μηχανή.@

**Παραεικονοποίηση**

**Para virtualization:** Υποείδος της εικονικοποίησης. @Στο οποίο υφίσταται συνεργασία guest και hypervisor ώστε να αντιμετωπίζονται αποτελεσματικότερα συγκεκριμένα σενάρια. @

Μια αξιοσημείωτη διαφροά είναι ο guest γνωρίζει πως βρίσκεται εντός τεχτητού περιβάλλοντος, και μεταβάλλει την συμπεριφορά του.@

**Μια ιδέα εικονικοποίσης**

Εδώ βλέπουμε μια εικόνα του πως μοιάζει η εικονικοποίοηση σε επίπεδο συστήματος.@ Έχουμε ένα γενικό λειτουργικό, τον host, @εντός του οποίου εκτελείται μια εικονική μηχανή , στην περίπτωση εδώ με Win10, τον guest@

**Προβλήματα**

Ωραία, όλα αυτά, όμως υπάρχουν κάποια προβληματά στην cloud εικονικοποίηση.

* @Τα συμβατικά λειτορυγικά δεν είναι σχεδιασμένα για το cloud. Πολύ **σύνθετα**, **αργά ακτι εκκίνηση**, **απαιτητιτκά σε πόρους** και **ευάλωτα** σε θέματα ασφάλεια. @

Για να αντιμετωπιστεί αυτό, έχουν εμφανιστεί τα unikernels. Πλαίσια ανάπτυξης μικρών εικονικών μηχανών που καλύπτουν τις ανώτερες παθογένειες.@

* Σε αρκετές περιπτώσεις οδηγούμαστε στην κακή χρήση και κατανομή των διαθέσιμων πόρων. O Hypervisor αποφασιζει ποση μνήμη από την συνολικά διαθέσιμη θα ανατεθεί στον Quest. Με αυτον τον τρόπο πιθανώς . **να μένει ανεκμετάλευτο** ένας μέρος της συνολικής.Ενώ σε περίπτωση ανεπάρκειας λόγω κακής κατανομής μπορεί **να επιβραδυνθεί η εκτέλεση** τού guest.

@Μια λύση σε αυτό προσφέρει η tmem. Ένας μηχανισμός συνεργασίας guest και host για καλύτερη αξιοποιήση της μνήμης. @

Μάλιστα, σε προηγούμενη εργασία του εργαστηρίου τροποποιήθηκε ώστε να μπορούν να τον χρησιμοποιούν απευθείας οι εφαρμογές του guest και ονομάστηκε utmem.@

**Σκοπός της εργασίας.**

@Όπως φαίνεται από τον τίτλο, ο σκοπός είναι η ενσωμάτωση του μηχανισμού utmem σε κάποιο unikernel πλαίσιο.@

Για να το πετύχουμε αυτό **Μελετήσαμε** τα διάφορα unikernel frameworks όπως και την utmem . Υστερα, **ενσωματώσαμε** την utmem σε ένα συγκεκριμένο framework της επιλογής μας και τέλος **εκτιμήσαμε** με πειράματα και μετρήσεις την αξία που έχει αυτή η υλοποίηση.@

**Θεωρία**

Θα ακολουθήσει τώρα μια απαραίτητη θεωρητική ανάλυση των τεχνολογιών που μας ενδιαφέρουν@

**Unikernel**

Τι χαρακτηρίζεται ως Unikernel;@

Μια εικονική μηχανή ή οποία είναι μικρή σε μέγεθος, εξειδικευμένη σε μία ενέργεια και με ενιαίο χώρο διευθύνσεων μνήμης @

Ουσιαστικά μια διαφορετική ΕΜ από αυτό που θα προέκυπτε αν εικονικοποιούσαμε ένα όλοκληρο συμβατικό ΛΣ. @

Η φιλοσοφία που τα χαρακτηρίζει είναι πως: αντί να δημιουργούμε εφαρμογές που να συμφωνούν με το λειτουργικό σύστημα, προσαρμόζουμε το λειτουργικό σύστημα ώστε να ταιριάζει στην εφαρμογή.@

**Unikernel 2**

Σε αυτή την εικόνα μπορούμε να δούμε πως συγκρίνεται ένα unikernel σε σχέση με μια εικονική μηχανή με συμβατικό ΛΣ.@

Στην συμβατική ΕΜ υπάρουν διακριτά επίπεδα software στην στοίβα, και το κυριότερο δυο διαφορετικοι χώροι, ο χώρος του πυρήνα και ο χώρος του χρήστησ. Αυτό καθιστά ολόκληρ το σύστημα ιδιαίτερα σύνθετο με ό,τι αρνητικό συνεπάγεται.@

Αντίθετα στο unikernel τα ανώτερα επίπεδα, στα οποία συμπεριλαμβάνεται ο κώδικας της εφαρμογής, και ο πυρήνα πρακτικά αποτελεούν ένα σώμα. Ένα εναρμονισμένο σώμα τέλεια σχεδιασμένο ώστε να εκτελεί την εφαρμογή για την οποία παρήχθηκε το unikernel.

Και στις δύο περιπτώσει από κάτω υπάρχει ο εκάστοτε επόπτης και το πραγματικό υλικό.@

**Frameworks**

Υπάρχουν διάφορα frameworks ή πλαίσια με τα οποία μπορούμε να παράγουμε unikernels.@ Όπως το **mirage OS**, ένα από τα πρώτα που αναπτύχθηκαν,@ το **include os**, @το **OSv**, @το **solo5**.@

Τέλος υπάρχει το Rumprun , το οποίο στοχεύ στην εκτέλση εφαρμογών που ακουλουθουν το Posix πρότυπο. Με αυτό το πλαίσιο ασχοληθήκαμε στην εργασία. Θα το αναλύσουμε εκτενώς στη συνέχεια.@

**Rump kernels-Rumprun**

@Αυτός εδώ είναι ο antii kantee, η συνεισφορά του οποίου είναι μέγιστη καθώς από αυτόν προήλθαν το Rump run και τα rump kernel.@

Επίσης εισήγαγε την ένοια του **anykernel**. Παραδοσιακώς οι σημαντικές υπηρεσίες ενός ΛΣ όπως οι drivers των συσκευών, το σύστημα αρχείων κλπ, είναι εστιασμένες στο σύστημα για το οποίο έχουν σχεδιαστεί.

Anykenrel είνια μια φιλοσοφία σχεδίασης συστημάτων, όπου οι drivers και άλλα σημαντικλα υποσυστήματα ανεξαρτητοπούνται από το λειτουργικό συστήματα.

Αυτό σημαίνει πως αναπτύσσονται αυτόνομα, ως ανεξάρτητες οντότες, οι οποίες εν τέλει μπορούν να χρησιμοποιηθούν από διαφορετικά ΛΣ χωρίς τροποποιήσεις.@

Το ΛΣ NetBSD ικανοποιεί την φιλοσοφία του anykernel, άλλωστε επάνω σε εργαζόταν και ο Kantee.@

**Rump kernels**

Προηγουμένος ανέφερα τα rump kernels.@

Η δουλεία του Kantee ήταν να αναπτύσει drivers για το NetBSD.

Για να το κάνει αυτό : **εξήγαγε** από το σύνολο των Driver του NetBSD μόνο **τα στοιχεία** με τα οποία ήθελε κάθε φορά να ασχοληθεί.

Το κατάλοιπο (rump στα αγγλικά) αυτής της διαδικασίας αφαίρεσης συστατικών είναι ένας μικρός πυρήνας που διαθέτει μόνο τις λειτουργίες που επιθυμούμε κάθε φορά. @

Για παράδειγμα, αν μας ενδιαφέρει να εστιάσουμε στην στοίβα δικτύου, δεν έχουμες κανένα λόγο να συμπεριλάβουμε και drivers για μια άσχετη συσκευή. Συνεπώς αφαιρούμε από το τελίκό πυρήνα τους συγκεκριμένους drivers. Κρατάμε μόνο ό,τι χρειαζόαμστε.@

Ονόμασε, λοιπόν, rump kernels όλα τα στοιχεία τα οποία μπορούν να εξαχθούν από τον πυρήνα NetBSD και να εκτελούνται ως σύνολο.@

**Πως φτάσαμε στο Rumprun**

Διαθέτουμε:@

* Τους **rumpkernls**, τον πυρήνα ενός λειτουργικού συστήματος
* Έναν **τρόπο** ώστε να μπορούν να εκτελούνται τα rump kernels επάνω στο Xen, ως guest
* Μια **διεπαφή** ώστε τα rump kernels να μπορουν να εξυπηρετήσουν κώδικα εφαρομγών από υψηλότερο επίπεδο@

Έχοντας και τις τρεις αυτές δυνατότητες, πρακτικά δημιουργείται το Rumprun, το unikernel περιβάλλον που στηρίζεται στους rump kernels. @

Στην παρακάτω εικόνα βλέπουμε την σχεση αυτών των τριών όρων. @Αριστερά το **anykernel**, με τα ανεξάρτητα στοιχεία και drivers---- @στην μέση τα **rump kernels**, ό,τι απομένει από την εξαγωγή στοιχείων από το NetBSD,-----@ και δεξιά το **Rumprun**, ένα Unikernel πλαίσιο που στηρίζεται στους rump kernels@

**Λόγοι επιλογής Rumprun**

Οι λόγοι που επιλέξαμε μεταξύ των άλλων frameworks το Rumprun είναι οι εξής.@

* Αυτό που επιθυμούμε, **είναι** να ενσωματώσουμε στο Rumprun την utmem, η οποία είναι ένας ιδαιίτερος driver . Το NetBSD είνια ένα well-documented λειτουργικό, οπότε εικάζουμε πως δεν θα είναι ιδιαιτέρα δύσκολη η ανάπτυξη του συγκεκριμένου driver.@
* Η υποστήριξη POSIX εφαρμογών είναι ένα μεγαλό θετικό του Rumprun. Καθώς τα περσσότερα προγράμματα υπακούν σε αυτό το πρότυπο. Συνεπώς, με το **Rumprun** προσφέρουμε την utmem σε ένα μεγαλύτερο κοινό unikernel εφαρμογών.@
* Στο Rumprun απουσιάζει η λειτοργία εικονικής μνήμης. Άρα ένας μηχανισμός σαν την utmem, προβλέπουμε, πως θα αποκτά και ουσιαστική αξία σε περιπτώσεις έλειψης μνήμης.@

**Μνήμη**

@Γενικά, ένα σημαντικότατο θέμα για τα υπολογιστικά συστήματα είναι αυτό της διαχείρησης της κύριας μνήμης, της μνήμης RAM δηλαδή.@

Συνοπτικά, αυτό που επιζητάμε είναι να υπάρχει πάντα διαθέσιμη μνήμη για τις ανάγκες των προγραμμάτων που τρέχουμε. **Ακομά** και αν αυτή εξαντληθεί υπάρχουν τεχνικές, όπως το swapping, με τις οποίες εξασφλαίζεται η αδιάκοπη εκτέλεση των προγραμμάτων, με σημαντικό κόστος, βεβαια, σε ταχύτητα.@

**Μνήμη και εικονικοποίηση**

Όταν αναφερόμαστε σε εικονικοποιημένους guest, η διαχείρηση μνήμης γίνεται ακόμα πιο σύνθετη. @Εκτός από επάρκεια μνήμης για τα προγράμαμτα, θέλουμε και να κατανέμεται η μνήμη με σωστό τρόπο ανάμεσα στους διάφορους guest, που μπορεί να συνυπάρχουν στο ίδιο σύστημα. @

Αν δωσουμε σε κάποιον guests υπερβολικά πολλή μνήμη, τότε γίνεται σπατάλη αυτής.

Αν δεν του δώσουμε αρκετή, ενώ υπάρχει διαθέσιμη αλλά ανενεργή στο σύστημα, τότε μιλάμε για υποχρησιμοποιήση της μνήμης.@

**Tmem**

Μια λύση προσφέρει ο μηχανισμός transcendent memory ή tmem. @

Ο μηχανισμός απαιτεί την συνεργασία μεταξύ του guest και του host-hypervisor. Οι δύο πλεύρες συνεργάζονται ώστε να ανταλλάσουν μεταξύ τους δεδομένα μνήμης.

Για παράδειγμα αν γεμισεί η μνήμη του guest, τότε αυτός ζητά από τον hypervisor-host να αποθηκεύσει, όπου αυτος κρίνει, δεδομένα από την μνήμη του guests. Στην συνέχεια ο guest μπορεί να απελευθερώσει τον χώρο αυτών των δεδομένων, σβήνοντας τα, καθώς γνωρίζει πως ανά πάσα στιγμή μπορεί να τα ανακτήσει επικοινωνώντας πάλί με τον host.@

Τα πιο σημαντικά μυνήματα που μπορεί να στείλει ο guest είναι Α. το αίτημα **Put**, με το οποίο μεταφέρονται δεδομένα από τον guest στον host, Β.το **Get**  με το οποίο ο guest ζητά πίσω δεδομένα.@

Η tmem δημιουργήθηκε από την Oracle το 2009 για το linux και αφορά guests επάνω στον Xen hypervisor.@

**Utmem**

Ο μηχανισμός tmem αφορά επικοινωνία ανάμεσα τον χώρο πυρήνα του guest και τον hypervisor-host. Οι εφαρμογές απολαμβάνουν μόνο έμμεσα τα πλεονεκτήματα του.@

Σε προηγούμενη εργασία του ΕΥΣ, αναπτύχθηκε μηχανισμός που εκθέτει την tmem απευθείας στον χώρο χρήστη, ώστε να μπορεί να την χρησιμοποιήσει όποια εφαρμογή το επιθυμεί. Ονομάστηκε utmem, από το user space tmem.@

Επίσης, αξιολογήθηκε θετικά σε διάφορα σενάρια εικονικοποίησης. @

Τέλος ,Η utmem αφορά linux host και linux guest, **ενώ** στηρίζεται στο KVM ως επόπτη, όχι στο Xen.@

**Τεχνική υλοποίηση**

Θα περάσουμε στα πιο ενδιαφέροντα. Θα παρουσιάσουμε πως έγινε η ενσωμάτωση της utmem στο Rumprun@

**Δομή utmem**

Σε αυτήν την εικόνα βλέπουμε την δομή του μηχανισμού utmem. @

Ξεκινώντας από επάνω έχουμε:

* @Μια εφαρμγογή του guest,που χρησιμοποιεί την utmem και που επικοινωνεί με μια ειδική εικονική συκευή.
* @Ύστερα η συσκευή αυτή, χρησιμοποιεί έναν ειδικό driver που ενσωματώνεται στον πυρήνα του linux, ο οποίος επικοινωνεί με τον hypervisor, το KVM
* @Τέλος το KVM χρησιμοποιεί το backend του μηχανισμού, δηλαδή με έναν υπομηχανισμό αποθήκευσης των δεδομένων. Εδώ πλεον βρισκόμαστε στο επίπεδο του host.@

Ο host στην δική μας περίπτωση θα παραμένει ο ίδιος, δηλαδή ένα linux μηχάνημα με το utmem backend.@

Άρα για το Rumprun, πρέπει να υλοποιήσουμε 2 πράγματα.

1. Έναν driver ο οποίος να είναι σε θέση να επικοινωνήσει με τον host και να ανταλλάξει δεδομένα.
2. Μια διεπαφή ώστε οι εφαρμογές του Rumprun να μπορούν να χρησιμοποιούν αυτόν τον driver.@

**System call**

@Αρχικά, για να μην μπερδευόμαστε θα αναφέρομαι στην linux utmem ως αυθεντική utmem, ενώ στην utmem για Rumprun που δημιουργήσαμε ως unikernel ή Rumprun utmem.@

Για να πετύχουμε την επικοινωνία με τις εφαρμογές δημιουργήσαμε μια νέα **κλήση συστήματος** του NetBSD, την οποία ονομάσαμε tmem(). Επίσης, φροντίσαμε να ενημερώσουμε τα rump kernels ώστε να την συμπεριλάβουν στα στοιχεία που προσφέρει το Rumprun.@

Η επικοινωνία με τον host γίνεται εντός του κώδικα της system call, οπότε δεν υπάρχει ξεχωριστός driver όπως υπάρχει στον αυθεντικό μηχανισμό.@

**Οπότε**, Υλοποιήσαμε,τα 2 ζητούμενα, και συνεπώς **πλέον** υπάρχει ο μηχανισμός της utmem και στο Rumprun πλαίσιο.

Για χρησιμοποιήσει κάποια εφαρμογή την utmem πρέπει απλά να καλέσει την συγκεκριμένη κλήση συστήματος, με τις κατάλληλες παραμέτρους **βεβαίως**. Από εκεί μετά θα αναλλαβουν τα rump kernels να προωθήσουν τα αιτήματα μέχρι και το backend.@

**Τεχνικές Δυσκολίες**

Προφανώς κατά την υλοποίηση των προηγούμενων εμφανίστηκαν δυσκολίες.@

Η πρώτη είναι πως στο NetBSD **δεν υπάρχει κάποιος μηχανισμός** ώστε να μπορούμε να καλούμε hypercalls προς το KVM, όπως υπάρχει στο linux. Hypercall είναι ο τρόπος με τον οποίο ένας guest μπορεί να επικοινωνήσει με τον επόπτη του. @Η λύση σε αυτό το εμποδιο ήταν να εξάγουμε από το source code του linux τις γραμμές που εκτελούν το κατάλληλο hypercall, και να τις μεταφέρουμε εντός της system call.@

H δεύτερη είναι πως host και guest **βλέπουν με διαφορετικό τρόπο** τις διευθύνσεις μνήμης όπου είναι γραμμένα τα δεδομένα. Για να επιτευχθεί σωστή επικοινωνία απαιτείται ένας τρόπος να μεταφραστούν αυτές οι διευθύνσεις από την «γλώσσα» του ενός στου άλλου. @Ευτυχώς τα rump kernels προσφέρουν έτοιμη συνάρτηση που επιτελεί την μετάφραση, η οποία ονομάζεται vtophys(), και έτσι μας έλυσαν τα χέρια.@

**Function call**

@Το ότι δημιουργήσαμε μια system call του NetBSD δεν έχει και τόσο μεγάλο νοήμα για το Rumprun, καθώς οι rump kernels θα την μετατρέψουν εν τέλει σε μια απλή κληση συνάρτησης.

Γιατί, λοιπόν, να μην δημιουργούσαμε εξ αρχής μια function call που να είχε ακριβώς την ίδια χρησιμότητα; Αρχικά ο ενιαίος χώρος μας επιτρέπει να εκτελούμε hypercalls από όποιο σημείο θέλουμε. Επίσης θα γλιτώναμε και μια αντιγραφή δεδομένων την οποία είμαστε υποχρεωμένοι να πράξουμε με την system call.@

Και αυτό ακριβώς κάναμε!@

Δημιουργήσαμε μια βιβλιοθήκη γλώσσας C, όπου οι ενέργειες utmem προσφέρονται ως απλές function calls. Η βιβλίοθήκη αυτή απλά πρέπει να απλά συμπεριληφθεί στον κώδικα της εφαρμογής που χρησιμοποιεί utmem, κατά την μεταγλώτισση της. Ό,τι κάναμε εντός της system call το κάνουμε τώρα εντός της νέας βιβλιοθήκης, του νέου αυτού driver.

Μάλιστα με αυτόν τον τρόπο δεν χρειάζεται καν να τροποποιήσουμε το Rumprun.@

Εις το εξής θα ασχοληθούμε και αναφερόμαστε μόνο με αυτή την έκδοση Rumprun utmem, δηλαδή ως απλή function call.@

**Εικόνα**

Στην εικόνα βλέπουμε πως συγκρίνονται οι δύο εκδόσεις.@

Αριστερά μια εφαρμογή χρειάζεται να καλέσεις την tmem() system call, και ύστερα η ροή περνά εντός των rump kernels. @

Αντίθετα, δεξία παρακάμπτουμε τους rump kernels, και στηριζόμαστε σε έναν driver που προσφέρεται εξ ολοκλήρου ως υψηλού επιπέδου βιβλιοθήκη της C.

Και στις δύο περιπτώσεις επικοινωνούμε με το KVM και το linux host μας.@

**Αξιολόγηση**

Για να αξιολογήσουμε την unikernel utmem διεξάγαμε στοχευμένες μετρήσεις και συγκρίσεις.@

**Breakdown**

Αρχικά θέλουμε να δούμε πως συμπεριφέρεται ο ίδιο ο μηχανισμός και τι επιδόσεις έχει.@

Αναλύουμε την καθυστέρηση στα στάδια των δύο βασικών ενεργειών, της Put και Get. Θυμίζουμε πως για αυτές τις ενέργειες, υπάρχει έχουμε δημιουργήσει αντίστοιχη συνάρτηση.@

Για να το κάνουμε αυτό, δημιουργήσαμε ένα βοηθητικό unikernel, το οποίο χρησιμοποιεί τις δύο συναρτήσεις, επικοινωνώντας δηλαδή με το back end. Αποστέλει συνεχώς, για προκαθορισμένο αριθμό φορών, το ίδιο αίτημα Put και Get και καταγράφει χρόνους εξυπηρέτησης. Ύστερα υπολογίζει μέσους όρους. @

Το μέγεθος των δεδομένων που μεταφέρονται με κάθε αίτημα είναι 1 mega byte. Οι χρόνοι καταγράφονται μέσω βοηθητικών συναρτήσεων που τοποθετήσαμε στα κατάλληλα σημεία εντός του κώδικα @

Διακρίνουμε δύο στάδια κατά την εκτέλεση κάθε ενέργειας. Το πρώτο στάδιο είναι όσο η ροή βρίσκεται εντός του driver που υλοποιήσαμε, αλλά οχί στο hypercall, και το ονομάζουμε driver χρόνο. Το δεύτερο: όσο βρισκόμαστε στο hypercall, δηλάδη στο επίπεδο του host.@

**Breakdown 2**

@Το σύστημα, ο host δηλαδή, στον οποίο έγιναν οι μετρήσεις ήταν ένα VM με ubuntu linux με τα κατάλληλα patches για το utmem backend. Το ίδιο σύστημα χρησιμοποιείται και για τις επόμενες μετρήσεις.

@Το αποτέλεσμα φαίνεται σε αυτόν τον πίνακα.

Βλέπουμε πως χρειάοζνται μερικές εκατοντάδες microseconds κατά μέσο όρο για να ικανοποιηθούν τα δύο αιτήματα.

@Επίσης Βλέπουμε πως η Put είναι αισθητά πιο αργή από την Get. Αυτό δεν είναι ανησυχιτικό, καθώς η ίδια ασυμμετρία παρατηρήθηκε και στην αυθεντική utmem, κα σχετίζεται με το γεγονός πως κατά την Put πρέπει ο Host να κατανείμει καινουρια μνήμη.@

**Αξία του μηχανισμού**

Ωραία, εμείς ενσωματώσαμε την utmem στο Rumprun. για ποιο λόγο όμως να θέλει ένας προγραμματιστής να την χρησιμοποιήσεις;@

Θα συγκρίνουμε, λοιπόν, την utmem με in-memory αποθήκευση δεδομένων. @

Για αυτόν τον σκοπό χρησιμοποιουμε το redis, ένα open source μετροπρόγραμμα που λειτουργεί ως αποθήκη αφηρημένων δεδομένων.@

Το μοντέλο του redis είναι, ένας **redis-server** που εξυπηρετεί αιτήματα από το δίκτυο και αποθηκεύει δεδομένα, και διάφοροι clients που στέλνουν αιτήματα στο server.@

Τα βασικά network requests του redis είναι set και get, υπάρχει μια σαφέσταστη αντιστοιχία με τις utmem ενέργειες.

Τέλος, Ο redis-server υπάρχει ήδη σε μορφή unikernel για το Rumprun.@

**Αξία του μηχανισμού 2**

@Για να είναι σε θέση να χρησιμοποιεί το redis την utmem προσθέσαμε νέες εντολές-network requests, ονόματι tmemPut tmemGet, οι οποίες χρησιμοποιούν την utmem για αποθήκευση δεδομένων.@

Επιλέγουμε να συγκρίνουμε την in-memory εντολή set που υπήρχεi ήδη με την tmemPut που προσθέσαμε εμείς.@

Φτιάξαμε ένα εξωτερικό πρόγραμμα, ονόματι client, το οποίο αποστέλει αιτήματα για προκαθορισμένο χρονικό διάστημα. @Καταγράφει επίσης τον μέσο ρυθμό εξυπηρέτησης των αιτημάτων.@

Διευκρυνίζουμε πως ο client σιγά-σιγά γεμίζει με δεδομένα την μνήμη είτε του redis στην περίπτωση της set, είτε του host στην περίπτωση της tmemPut.@

**Αξία του μηχανισμού 3**

@Έχουμε ένα **πρώτο σενάριο**, όπου το unikernel με το redis διαθέτει υπερ αρκετή μνήμη.@

Τότε η tmemPut εξυπηρετείται πιο αργά από την set. Αναμενόμενο, καθώς για κάθε αίτημα Put απαιτείται μετακίνηση δεδομένων από το redis έως και τον host, και 1 hypercall.

Στο σχήμα φαίνεται η ποσοστιαία σχέση των δύο εντολών ανά τιμή του μεγέθους των δεδομένων του εκάστοτε αιτήματος.@

Τι γίνεται,**όμως**, όταν η μνήμη του unikernel δεν αρκεί για να αποθηκεύσει όλα τα δεδομένα;@

Όταν στηριζόμαστε στην set, η εκτέλση του redis μετά από ένα χρονικό διάστημα καταρέει! Δεν Δεν υπάρχει κανένας τρόπος να αποθηκευτούν περεταίρω δεδομένα, το redis εμφανίζει μύνημα σφάλματος και τερματίζει την λειτουργία του.@

Αντίθετα τα tmemPut συνεχίζουν να εξυπηρετούνται κανονικά. Εφόσον δεν αποθηκεύεται τίποτα μόνιμα στην μνήμη του redis, αρκεί να μπορει να εξυπηρετηθεί ένα request, ώστε να εξυπηρετούνται όλα.@

**Σύγκριση με τον αυθεντικό μηχανισμό**

@Τέλος, θα θέλαμε να δούμε πως συγκρίνεται η unikernel utmem με την αυθεντική σε αντίστοιχο σενάριο με το προηγούμενο.@

Χρησιμοποιούμε πάλι το redis, τώρα όμως ως απλή διεργασία ένος linux guest επάνω στον ίδο host.@

Δεν γεμίζουμε με δεδομένα την μνήμη, αλλά αποστέλουμε το ίδιο αίτημα tmemPut για προκαθορισμένο χρόνο.

Πάλι **μετράμε** commands / δευτερόλεπτο, καιγια διάφορες τιμές value size. @

Κάνουμε την ίδια δοκιμή και για την set εντολή του redis. Αν το διαφορετικό σύστημα (στην μια Rumprun στην άλλη linux) δεν επηρεάζει το αποτέλσμα τότε η set πρέπει να εμφανίζει όμοια συμπεριφορά ανάμεσα στα δύο συστήματα, καθώς η διαχείρηση της set είναι κοινή και στα δύο redis.@

**Σύγκριση με τον αυθεντικό μηχανισμό2**

@Μια μια πρώτη, αλλά αφελή, ματία φαίνεται η unikernel utmem είναι ιδιαίτερα ταχύτερη από την αυθεντική.@

Εντούτοις, Όμοια συμπεριφορά παρουσιάζει και η set εντολή.

Χρειάζεται σε αυτό το σημείο να προχωρήσουμε σε επιπλέον ανάλυση του φαινομένου.@

**Σύγκριση με τον αυθεντικό μηχανισμό3**

@Αναλύουμε, τώρα τα στάδια καθυστέρηση των αιτημάτων για τα δύο σύστημα. Εστιάζουμε, βέβαια μόνο στην tmemPut, που μας ενδιέφερε εξ αρχής.@

Τοποθετούμε πάλι γραμμές εντός του κώδικα που καταγράφουν χρόνους.

Για την unikernel Utmem έχω τους εξής χρόνους-σταδια:

* **Network**, όσος χρόνος μεσολαβεί από την στιγμή που ο client αποστέλνει το δικτυακό αίτημα μέχρι να αναλλάβει η εντολή μέσα στο redis.
* **Redis:** o χρόνος εντός του redis, εως ότου να αναλλάβει ο μηχανισμός utmem.
* **Driver** : ο χρόνος εντός του driver αλλά όχι εντός της hypercall
* **Hypercall** : ο χρόνος εντός του hypercall@

Οι δύο τελευταίο χρόνοι είνια οι ίδιοι που όρισαμε στο κομμάτι του breakdown, προηγουμένως.

Για την αυθεντική **network και redis time** ορίζονται με τον ίδιο τρόπο.@

Οι δύο άλλοι, συμψηφίζονται σε έναν καθώς είναι σχετικά δύσκολο να μετρήσουμε τον χρόνο της hypercall ξεχωριστά.

Τον ονομάζουμε ioctl(), επειδή τέτοιου είδους system call πραγματοποιείται στην αυθεντική utmem για τα αιτήματα. Η αντιστοιχία με τους άλλους δύο χρόνους είναι αρμονική, καθώς στον κώδικα η ioctl() και η συνάρτηση που υλοποιήσαμε χρησιμοποιούνται στις ίδιες γραμμές από τις εφαρμογές.@

**Σύγκριση με τον αυθεντικό μηχανισμό3**

Επαναλαμβάνουμε το ίδιο πείραμα, και τώρα καταγράφουμε χρόνους.

Αναφέρουμε πως τελικά ο χρόνος redis ήταν ασήμαντος και στις δύο περιπτώσεις, οπότε τον αγνοούμε.@

Από τα αποτελέσματα το πρώτο που παρατηρούμε είνια πως **και στα δύο** συστήματα, ο χρόνος network επιβάλεται κατά κράτος στον συνολικό. Η καθυστέρηση που παρατηρεί ένας εξωτερικός χρήστης πρακτικά οφείλεται σε αυτόν, δηλάδη σε ό,τι αφορά το δικτυακό κομμάτι.@

Επίσης, η σχέση μεταξύ των network time των δύο συστημάτων είναι ακριβώς τέτοια που να δικαιολογεί την προηγούμενη συμπεριφορά που βλέπαμε ως εξωτερικοί χρήστες.@

**Σύγκριση με τον αυθεντικό μηχανισμό3**

Ας εστιάσουμε τώρα στα στάδια που εξαρτώνται άμεσα από τον εκάστοτε μηχανισμό.@

Με βάση αυτό, παρατηρούμε πως η unikernel utmem είναι ελαφρώς πιο γρήγορη από την αυθεντική,@

Αυτό οφείλεται στο οτι

* 1. ο αυθεντικός μηχανισμός είναι σχετικά πιο σύνθετος σε ο,τι αφορά την εσωτερική virtual device, σε σχέση με τον δικό μας driver.
* 2. Ο αυθεντικός μηχανισμός είναι υποχρεωμένος να εκτελεί μια αντιγραφή των δεδομένων παραπάνω κατά την ροή των αιτημάτων.@

**Συμπεράσματα**

Με βάση τα παραπάνω βγαίνουν τα εξής συμπεράσματα.@

Χρησιμοποιόντας την utmem

1. Οι απαιτήσεις σε μνήμη που πρέπει να αναταθετεί σε ένα unikernel ελαχιστοποιούνται. Έτσι, μπορούμε να υποστηρίζουμε περισσότερες εικονικές μηχανές σε ένα υπολογιστικό σύστημα ταυτόχρονα.
2. Αποφεύγουμε το καταστροφικό σενάριο εξάντλησης της μνήμης, το οποίο όπως είδαμε στην περίπτωση του redis οδηγεί στην κατάρευση της εκτέλεσης του.@

Επίσης, αν είναι δυνατόν να χρησιμοποιήσουμε unikernel utmem αντί για τον αυθεντικό μηχανισμό, ένας εξωτερικός χρήστης απολαμβάνει το πρόνομιο της ταχύτερης επικοινωνίας και εξυπηρέτησης. Το προνόμιο αυτό, **βέβαια**, πηγάζει από την σχεδίαση του Rumprun και όχι τόσο από την διαφορά των δύο μηχανισμών.

Το τελευταίο συμπέρασμα είναι πάρα πολύ σημαντικό, καθώς η πλειοψηφία των εφαρμογών για cloud αφορούν διακτυακή επικοινωνία.@

Το γενικότερο συμπερασμα όλων : ο μηχανισμός utmem συνδυάζεται άρτια με ένα unikernel πλαίσιο όπως το Rumprun.@

**Τέλος**

Και αυτή ήταν η διπλωματική εργασία μου.

Σας ευχαριστώ όλους για τον χρόνο και την προσοχη σας