

# Σχεδιασμός και υλοποίηση μηχανισμών pipe και fork σε unikernels

Μάινας Χαράλαμπος

Επιβλέπων: Γεώργιος Γκούμας

Εργαστήριο Υπολογιστικών Συστημάτων  
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών  
Εθνικό Μετσόβιο Πολυτεχνείο

WWW: <http://cslab.ece.ntua.gr/research/>



Μάρτιος 2019

# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



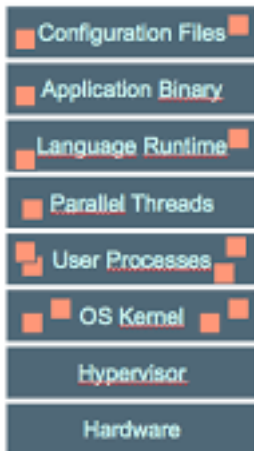
## Unikernels

Εξειδικευμένες εικόνες μηχανές, με ένα μοναδικό χώρο διευθύνσεων, τα οποία κατασκευάζονται χρησιμοποιώντας library operating systems

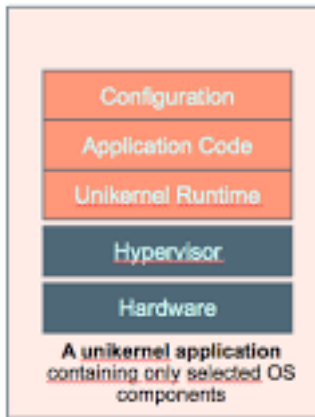
### Αναλύοντας τον ορισμό

- Εξειδικευμένες: κάθε unikernel περιέχει μία και μόνο εφαρμογή και έχει χτιστεί με σκοπό την υποστήριξη μόνο αυτής.
- Μοναδικός χώρος διευθύνσεων: δεν υπάρχει διαχωρισμός kernel space και user space.
- library operating systems: λειτουργικά συστήματα που παρέχουν τις υπηρεσίες που υποστηρίζουν (π.χ. networking) σε μορφή βιβλιοθηκών, οι οποίες μπορούν να χρησιμοποιηθούν (link) από τις εφαρμογές.

# Typical OS vs unikernel



Typical application  
running above an OS



# Χαρακτηριστικά των unikernels

## Πλεονεκτήματα

- Γρήγοροι χρόνοι εκκίνησης
- Μικρό memory footprint
- Περισσότερη ασφάλεια

## Μειονεκτήματα

- Οι εφαρμογές χρειάζονται porting
- Πολλές βιβλιοθήκες μπορεί να μην υποστηρίζονται
- Πολλά components των OSes μπορεί να μην υποστηρίζονται (single process)



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



# Unikernel frameworks

## Rumprun

- Είναι βασισμένο στο NetBSD και συγκεκριμένα στα rump kernels
- POSIX friendly
- Υποστήριξη για threads, filesystem
- Υποστηρίζει πολλές γλώσσες
- Πολλές εφαρμογές έτοιμες να εκτελεστούν σε αυτό



# Unikernel frameworks

## OSv

- Είναι φτιαγμένο από την αρχή, με στόχο να εκτελείται στο cloud
- POSIX compatible
- Υποστήριξη για threads, filesystem.
- Υποστηρίζει πολλές γλώσσες
- Πολλές εφαρμογές έτοιμες να εκτελεστούν σε αυτό
- Από τα πιο ενεργά projects





# Unikernel frameworks

## IncludeOS

- Είναι φτιαγμένο από την αρχή
- Μερικώς POSIX compatible
- single threaded
- Ταχύτατα αναπτυσσόμενο project
- Μόνο εφαρμογές σε C++ μπορούν να εκτελεστούν σε αυτό



# Unikernel frameworks

## MirageOS

- Είναι φτιαγμένο από την αρχή και είναι γραμμένο σε OCaml
- Μόνο εφαρμογές σε OCaml μπορούν να εκτελεστούν σε αυτό
- Από τα πιο παλιά unikernel frameworks



# Unikernel frameworks

## MirageOS

- Είναι φτιαγμένο από την αρχή και είναι γραμμένο σε OCaml
- Μόνο εφαρμογές σε OCaml μπορούν να εκτελεστούν σε αυτό
- Από τα πιο παλιά unikernel frameworks

## Πολλά ακόμα

- ClickOS
- LKL
- Mini-OS, Unikraft
- HalVM, LING



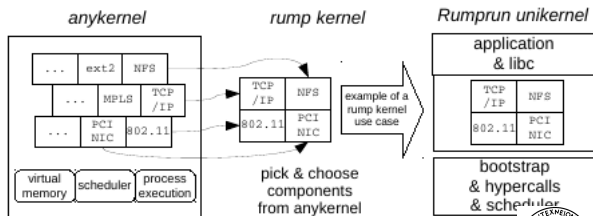
# Rumprun

## Anykernels

Codebase (πυρήνα) όπου οι οδηγοί μπορούν να εξαχθούν και να ενσωματωθούν σε οποιοδήποτε μοντέλο ΛΣ

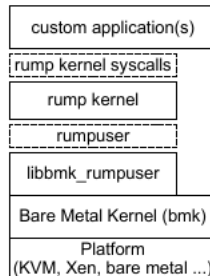
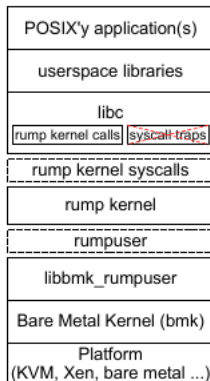
## Rump kernels

παρέχουν drivers του NetBSD ως φορητά εξαρτήματα με τα οποία μπορούμε να εκτελέσουμε εφαρμογές χωρίς να ναι απαραίτητη η ύπαρξη ΛΣ



# Rumprun stack

same threads  
throughout entire stack



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



# Βασική ιδέα

## single process

- Βασικό κοινό χαρακτηριστικό όλων των unikernels, είναι ότι είναι single process
- Συνεπώς δεν υποστηρίζεται η κλήση fork, σε κανένα unikernel

## Ιδέα

- Προσπαθούμε να κάνουμε τα unikernels όσο πιο πολύ POSIX friendly γίνεται
- Θέλουμε να κρατήσουμε τα βασικά χαρακτηριστικά των unikernels
- Αντιλαμβανόμαστε τα unikernels ως διεργασίες και το hypervisor ως λειτουργικό σύστημα

# Αντικείμενο διπλωματικής

## Μηχανισμός fork

- Με βάση αυτή την ιδέα, υλοποιούμε ένα μηχανισμό fork, ο οποίος αντί για τη δημιουργία μίας νέας διεργασίας εντός του unikernel, έχει ως αποτέλεσμα τη δημιουργία ενός νέου unikernel
- Ουσιαστικά μεταφέρουμε το μηχανισμό fork, από το επίπεδο των διεργασιών στο επίπεδο των εικονικών μηχανών

## Μηχανισμός pipe

- Συχνό φαινόμενο όταν δημιουργείται μία νέα διεργασία είναι να επικοινωνεί αυτή με τη διεργασία γονέας
- Μία από τις πιο συνηθισμένες πρακτικές για αυτό το σκοπό είναι η χρήση του μηχανισμού pipe
- Κατασκευή ενός μηχανισμού pipe σε επίπεδο εικονικών μηχανών.





# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



# Γενική εικόνα

## Pipe system call

```
int pipe(int fildes[2]);
```

Pipe semantics:

- Αν το pipe είναι άδειο, η κλήση ανάγνωσης μπλοκάρει μέχρι να προκύψουν δεδομένα
- Αν το pipe είναι γεμάτο, η κλήση εγγραφής μπλοκάρει μέχρι να ρλρυθερωθεί χώρος
- Αν δεν υπάρχουν ανοιχτά άκρα εγγραφής, τότε εανάγνωση στο pipe επιστρέφει 0, end of file
- Αν δεν υπάρχουν ανοιχτά άκρα ανάγνωσης, τότε εγγραφή στο pipe επιστρέφει το error EPIPE



# Γενική εικόνα

## Pipe σε unikernels

Η κλήση συστήματος pipe χρησιμοποιείται από τα unikernels, όπως θα χρησιμοποιούταν από τις διεργασίες σε ένα συμβατικό ΛΣ>

## Τρία στάδια υλοποίησης

- ❶ function call (με χρήση TCP/IP sockets)
- ❷ system call (με χρήση UDP sockets)
- ❸ system call (με χρήση κοινής μνήμης μεταξύ εικονικών μηχανών)



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



# Πρώτο στάδιο υλοποίησης

## function call

- Προσθέτουμε στον κώδικα της εφαρμογής, το function call pipe
- Καλώντας τη συνάρτηση δημιουργούνται δύο sockets, ένα για την αποστολή και ένα για τη λήψη δεδομένων.
- Τα δύο αυτά sockets επιστρέφονται και μπορούν να χρησιμοποιηθούν αντίστοιχα
- Η διεύθυνση του παραλήπτη, ορίζεται στον κώδικα της εφαρμογής
- Δεν υλοποιήθηκαν όλα τα semantics του pipe



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



# Δεύτερο στάδιο υλοποίησης

## system call

- Χρήση UDP sockets, αντί για TCP
- Δημιουργία system call για τον πυρήνα του NetBSD
- Δημιουργία δύο sockets, ένα για αποστολή και ένα για λήψη δεδομένων.
- Επιστρέφονται δύο file descriptors, ένα για εγγραφή και ένα για ανάγνωση.
- Η διεύθυνση του παραλήπτη, ορίζεται μέσω μίας κλήσης ioctl στο άκρο εγγραφής του pipe.
- Δεν υλοποιήθηκαν όλα τα semantics του pipe



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση





# Τρίτο στάδιο υλοποίησης

## system call

- Χρήση κοινής μνήμης μεταξύ των εικονικών μηχανών (invshmem - nahanni)
- Επιστρέφονται δύο file descriptors, ένα για εγγραφή και ένα για ανάγνωση.
- Μόνο για unikernels που μοιράζονται τον ίδιο host
- Υλοποιήθηκαν όλα τα semantics του pipe



## Τρίτο στάδιο υλοποίησης

### ivshmem

- Για την κοινή μνήμη χρησιμοποιήθηκε ο μηχανισμός ivshmem - nahanni
- Επιτρέπει την κοινή μνήμη μεταξύ δύο εικωνικών μηχανών σε QEMU
- Πρέπει να υλοποιηθεί ένας PCI driver για τη χρήση του

### Υλοποίηση

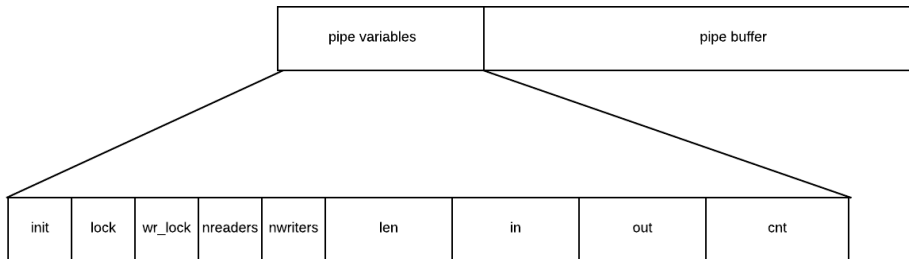
- PCI driver για το NetBSD και το rumpun
- Pipe system call που χρησιμοποιεί την κοινή μνήμη



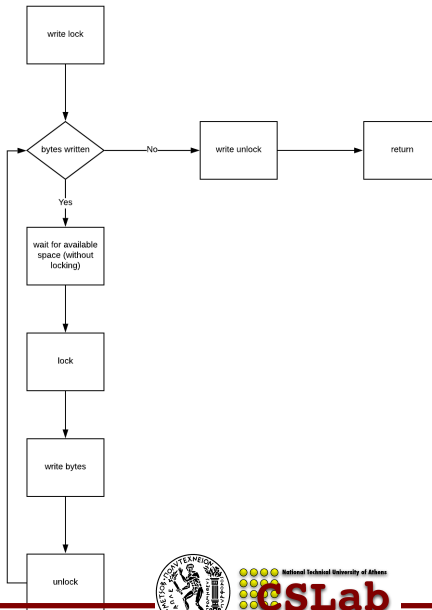
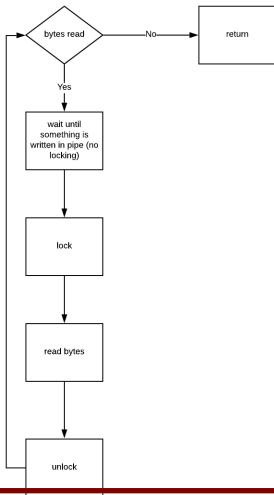
# Τρίτο στάδιο υλοποίησης

## Υλοποίηση

- PCI driver για το NetBSD και το rumprun
- Pipe system call που χρησιμοποιεί την κοινή μνήμη



# Τρίτο στάδιο υλοποίησης



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



# Γενική εικόνα

## fork system call

- Δημιουργία νέας διεργασίας
- Η νέα διεργασία είναι ίδια με τη διεργασία γονέα
- Διαχωρισμός των δύο διεργασιών με τιμή επιστροφής από την κλήση συστήματος
- Τα ανοιχτά file descriptors της διεργασίας γονέα, διατηρούνται και στη διεργασία παιδί



# Γενική εικόνα

## fork system call

- Δημιουργία νέας διεργασίας
- Η νέα διεργασία είναι ίδια με τη διεργασία γονέα
- Διαχωρισμός των δύο διεργασιών με τιμή επιστροφής από την κλήση συστήματος
- Τα ανοιχτά file descriptors της διεργασίας γονέα, διατηρούνται και στη διεργασία παιδί

## Στόχος

Η μεταφορά της ίδιας λειτουργίας σε επίπεδο εικονικών μηχανών

# Γενική εικόνα

## Υλοποίηση

- Χρήση του μηχανισμού migration, που παρέχει το QEMU, με μικρές αλλαγές
- hypercalls από το fork system call του rumpun

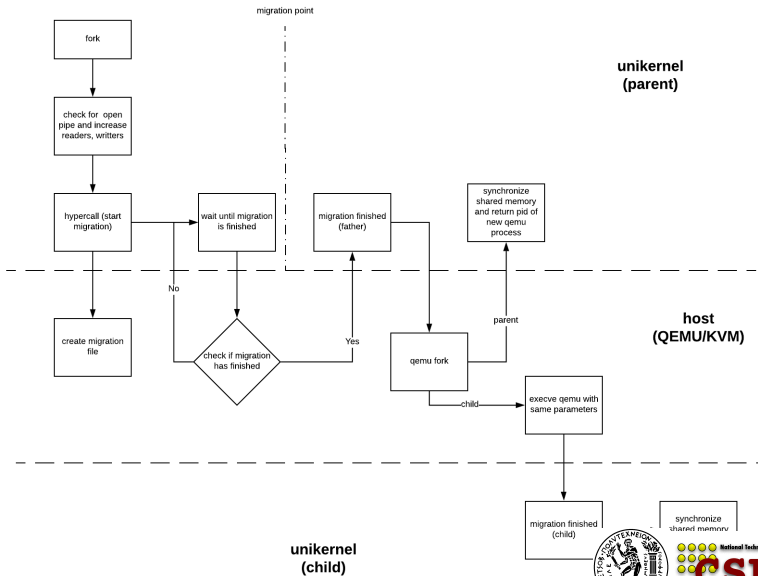
## Βήματα

- Ενημέρωση των κοινών μεταβλητών του pipe (αν υπάρχει pipe)
- Εκκίνηση διαδικασίας migration
- Αναμονή για migration
- Εκκίνηση νέας εικονικής μηχανής με βάση το migration file

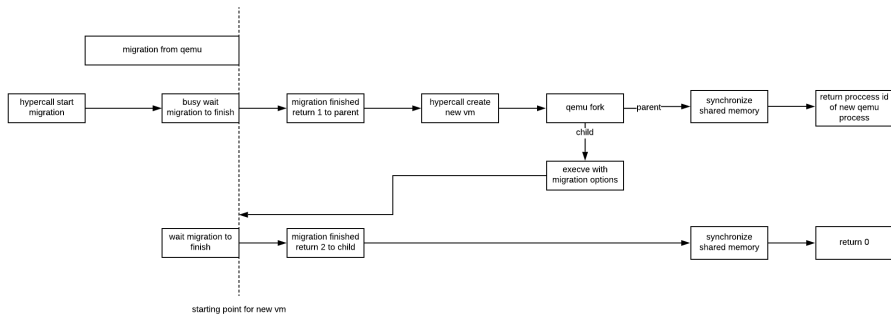




# Γενική εικόνα



# Γενική εικόνα



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



# Ενημέρωση κοινών μεταβλητών στο pipe

- Αν υπάρχει pipe θα πρέπει να ενημερωθούν οι κοινές μεταβλητές
- Έλεγχος ύπαρξης pipe, μέσω των ανοιχτών file descriptors της εφαρμογής
- Αύξηση των μετρητών ανοιχτών άκρων
- Χάρης το migration, η εικονική μηχανή παιδί θα έχει πρόσβαση στην κοινή μνήμη



# Εκκίνηση δημιουργίας migration file

## Από τη μεριά του rumpun

Hypercall για την εκκίνηση δημιουργίας migration file

## Από τη μεριά του QEMU

- Χρήση του μηχανισμού migration του QEMU
- Δημιουργία ενός thread, που αναλαμβάνει αυτή την εργασία
- Μικρές αλλαγές στον κώδικα του migration, ώστε να μη σταματήσει η λειτουργία της εικονικής μηχανής γονέας
- Καθόλη τη διάρκεια του migration η εικονική μηχανή θα πρέπει να εκτελείται



# Αναμονή για την περάτωση του migration

## Από τη μεριά του rumpun

- Busy wait με συνεχή hypercalls
- Είναι το σημείο που θα εκκινήσει τη λειτουργία του το unikernel παιδί.

## Από τη μεριά του QEMU

- Για κάθε hypercall ελέγχεται αν το migration thread εκτελείται ακόμα
- Διατήρηση counter, για τα πόσα hypercalls γίνονται



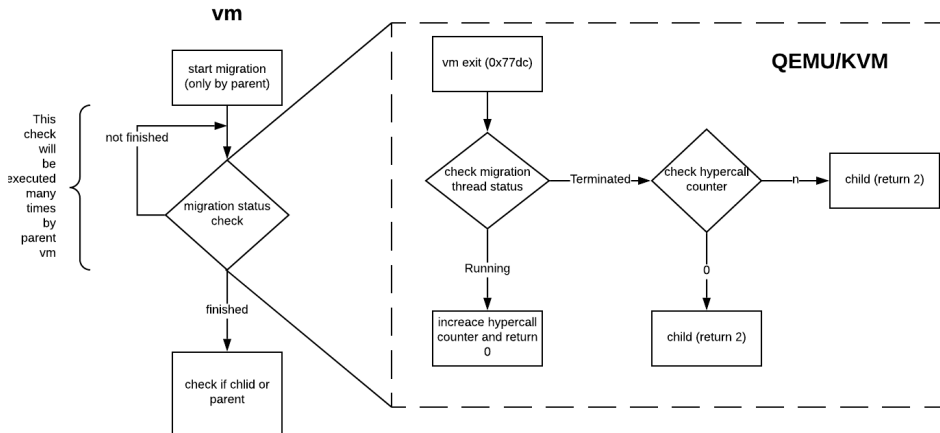
# Αναμονή για την περάτωση του migration

## Hypercalls counter

- Με αυτόν διακρίνουμε αν πρόκειται για το παιδί ή το γονέα
- Κάθε εικονική μηχανή στο QEMU είναι μία ξεχωριστή διεργασία, άρα διαφορετικοί counters για κάθε vm
- Migration είναι χρονοβόρα διαδικασία, ο γονέας θα κάνει αρκετά hypercalls
- Όταν θα εκκινήσει το παιδί το migration θα έχει ολοκληρωθεί, άρα ακριβώς ένα hypercall



# Αναμονή για την περάτωση του migration





# Δημιουργία νέας εικονικής μηχανής

## Από τη μεριά του rumpun

Hypercall για την εκκίνηση νέας εικονικής μηχανής

## Από τη μεριά του QEMU

- fork
- Επιστροφή στον γονέα το process id της νέας διεργασίας
- Εκκίνηση νέας εικονικής μηχανής με βάση το migration file



# Συγχρονισμός εικονικής μνήμης

- Μόνο στην περίπτωση που χρησιμοποιείται pipe
- Παρατηρήθηκε ότι αν το παιδί δε γράψει κάτι στην κοινή μνήμη, τότε το παιδί δεν μπορεί να διαβάσει τα δεδομένα που γράφει ο γονιός
- Busy wait από το γονιό, να αλλάξει μία τιμή το παιδί
- Το παιδί, απλά γράφει μία προκαθορισμένη τιμή σε μία συγκεκριμένη θέση μνήμης



# Επισκόπηση

## 1 Εισαγωγή

- Unikernels
- Unikernel frameworks
- Βασική ιδέα

## 2 Μηχανισμός pipe

- Γενική εικόνα
- Πρώτο στάδιο υλοποίησης
- Δεύτερο στάδιο υλοποίησης
- Τρίτο στάδιο υλοποίησης

## 3 Μηχανισμός fork

- Γενική εικόνα
- Βήματα
- Αξιολόγηση



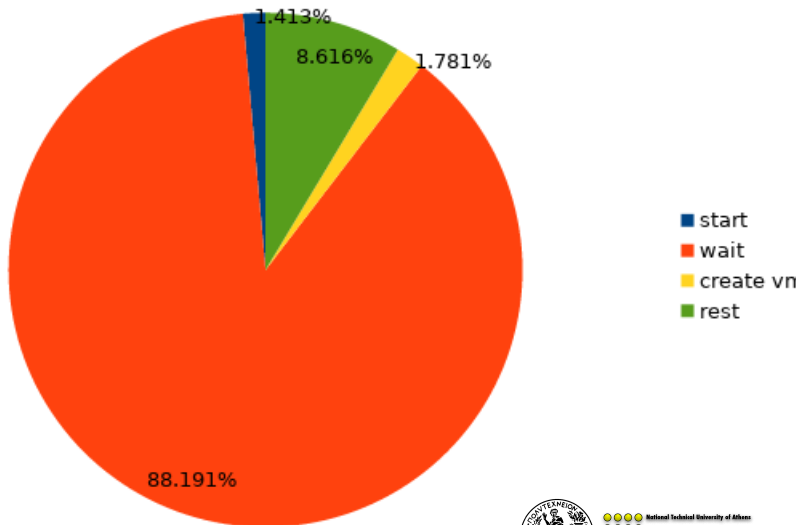
# Μετρήσεις

## Σύγκριση fork

- Σύγκριση fork σε linux πυρήνα και του fork που δημιουργήσαμε
- Σε ίδια έκδοση QEMU (2.11.2)
- Linux 4.9.0-7-amd64 #1 SMP Debian 4.9.110-3+deb9u2
- Linux: 0.541ms, Unikernel: 137,423ms
- Λογική η διαφορά, αλλά γιατί τόσο μεγάλη?



# Time for each step in fork



# Σύνοψη

Τα unikernels:

- αποτελούν μία αρκετά ενδιαφέρουσα τεχνολογία
- έχουν διαφορετικούς στόχους ανάλογα με το project
- (μερικά frameworks) προσπαθούν να είναι POSIX compatible
- είναι από μόνα τους single process

## Αντιμετώπιση των unikernels ως διεργασίες

- Μηχανισμός pipe για unikernels σε ίδιο και διαφορετικό host
- Μηχανισμός fork για unikernels σε ίδιο host



# Μελλοντικές Επεκτάσεις

- Χρήση σημάτων, ώστε να μη βασίζεται ο μηχανισμός pipe τόσο στην κοινή μνήμη
- Επέκταση των inter-unikernel μηχανισμών επικοινωνίας (signaling, message queues)
- Βελτίωση μηχανισμού fork, δημιουργώντας καλύτερο migration μηχανισμό
- Επέκταση του μηχανισμού και σε άλλες πλατφόρμες.



# Ευχαριστώ!

## Ερωτήσεις;







# Backup

