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

Μελέτη επίδρασης ανταγωνισμού για κοινούς πόρους σε περιβάλλον cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΑΠΑΔΗΜΗΤΡΙΟΥ

Επιβλέπων : Γεώργιος Γκούμας
Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιανουάριος 2017

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

Μελέτη επίδρασης ανταγωνισμού για κοινούς πόρους σε περιβάλλον cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΑΠΑΔΗΜΗΤΡΙΟΥ

Επιβλέπων : Γεώργιος Γκούμας
Αν. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Ιανουαρίου 2017.

Γεώργιος Γκούμας Αν. Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Αν. Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.

Αθήνα, Ιανουάριος 2017

Κωνσταντίνος Παπαδημητρίου

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

Περίληψη

Σκοπός της παρούσης εργασίας είναι η ανάλυση βασικών αλγορίθμων χρονοδρομολόγησης σε περιβάλλον cloud computing, δηλαδή θα δούμε πόσο περίπου βγαίνει..

Σε λίγο θα ξενερώσω περισσότερο..

Λέξεις κλειδιά

Γλώσσες προγραμματισμού, Προγραμματισμός με αποδείξεις, Ασφαλείς γλώσσες προγραμματισμού, Πιστοποιημένος κώδικας.

Abstract

The purpose of this diploma dissertation is on one hand the design of a simple high-level language that supports programming with proofs, and on the other hand the implementation of a compiler for this language. This compiler will produce code for an intermediate-level language suitable for creating certified binaries.

The need for reliable and certifiably secure code is even more pressing today than it was in the past. In many cases, security and software compatibility issues put in danger the operation of large systems, with substantial financial consequences. The lack of a formal way of specifying and proving the correctness of programs that characterizes current programming languages is one of the main reasons why these issues exist. In order to address this problem, a number of frameworks with support for certified binaries have recently been proposed. These frameworks offer the possibility of specifying and providing a formal proof of the correctness of programs. Such a proof can easily be checked for validity before running the program.

The frameworks that have been proposed are intermediate-level in nature, thus the process of programming in these is rather cumbersome. The high-level languages that accompany some of these frameworks, while very expressive, are hard to use. A simpler high-level language, like the one proposed in this dissertation, would enable further use of this programming idiom.

In the language we propose, the programmer specifies the partial correctness of a program by annotating function definitions with pre- and post-conditions that must hold for their parameters and results. The programmer also provides a set of theorems, based on which proofs of the proper implementation and use of the functions are constructed. An implementation in OCaml of a compiler from this language to the NFLINT certified binaries framework was also completed as part of this dissertation.

We managed to keep the language close to the feel of the current widespread functional languages, and also to fully separate the programming stage from the correctness-proving stage. Thus an average programmer can program in a familiar way in our language, and later an expert on formal logic can prove the semi-correctness of a program. As evidence of the practicality of our design, we provide a number of examples in our language with full semi-correctness proofs.

Key words

Programming languages, Programming with proofs, Secure programming languages, Certified code.

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή αυτής της διατριβής, κ. Γιάννη Παπαδάκη, για τη συνεχή καθοδήγηση και εμπιστοσύνη του. Ευχαριστώ επίσης τα μέλη της συμβουλευτικής επιτροπής, κ.κ. Νίκο Παπαδόπουλο και Γιώργο Νικολάου για την πρόθυμη και πάντα αποτελεσματική βοήθειά τους, τις πολύτιμες συμβουλές και τις χρήσιμες συζητήσεις που είχαμε. Θέλω να ευχαριστήσω ακόμα τον συμφοιτητή και φίλο Πέτρο Πετρόπουλο, ο οποίος με βοήθησε σε διάφορα στάδια αυτής της εργασίας. Θα ήθελα τέλος να ευχαριστήσω την οικογένειά μου και κυρίως τους γονείς μου, οι οποίοι με υποστήριξαν και έκαναν δυνατή την απερίσπαστη ενασχόλησή μου τόσο με την εκπόνηση της διπλωματικής μου, όσο και συνολικά με τις σπουδές μου.

Κωνσταντίνος Παπαδημητρίου,

Αθήνα, 17η Ιανουαρίου 2017

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-42-14, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιανουάριος 2017.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Αρχές πολυπύρηνων αρχιτεκτονικών

Τα τελευταία χρόνια η σχεδίαση και οι αρχές των υπολογιστικών συστημάτων έχουν αλλάξει ραγδαία. Αρχικά ένας επεξεργαστής αποτελούνταν από ένα μόλις πυρήνα ο οποίος αναλάμβανε να εκτελέσει όλη την εργασία. Αυτό φυσικά καυτστερεί ιδιαίτερα τις διάφορες εφαρμογές και δεν εκμεταλεύεται τις δυνατότητες παραλληλισμού είτε εντός της εφαρμογής είτε μεαταξύ δύο η περισσότερων εφαρμογών. Έτσι η πρόοδος της τεχνολογίας και οι απαιτήσεις για μεγάλη υπολογιστική δύναμη έφεραν τις πολυπύρηνες αρχιτεκτονικών, στις οποίες ο φόρτος εργασίας διαμοιράζεται σε περισσότερους πυρήνες, βελτιώνοντας θεαματικά την απόδοση ενός συστήματος.

Η τεχνολογία των πολυπύρηνων αρχιτεκτονικών ήταν γνωστή και χρησιμοποιούταν από τις προηγούμενες δεκαετίες σε συστήματα μεγάλης κλίμακας όπως υπερυπολογιστές και κέντρα δεδομένων (data centers). Είναι τα τελευταία χρόνια όμως που έχει γνωρίσει μεγάλη άνθηση. Η μεγάλη ζήτηση σε αποδοτικότερα συστήματα έφερε την τεχνολογία σε υπολογιστές και συστήματα γενικού σκοπού όπως laptop και κινητά τηλέφωνα. Ωστόσο, τα συστήματα αυτά συναντάνε νέα προβλήματα (ΠΡΕΠΕΙ ΝΑ ΒΑΛΩ ΤΑ ΠΡΟΒΛΗΜΑΤΑ ΤΩΝ ΜΟΝΟΠΥΡΗΝΩΝ ΑΡΧΙΤΕΚΤΟΝΙΚΩΝ) τα οποία περιορίζουν την βελτίωση της απόδοσης/υπολογιστικής ισχύος. Οι πυρήνες μοιράζονται κοινούς πόρους για τη χρήση/χρησιμοποίηση των οποίων ανταγωνίζονται. Αυτό έχει ως αποτέλεσμα τον περιορισμό της συνολικής επίδοσης (throughput). Είναι δουλειά λοιπόν του λειτουργικού συστήματος να μοιράσει με τέτοιο τρόπο τους πόρους στις διάφορες εφαρμογές ώστε να ελαχιστοποιείσει αυτόν τον ανταγωνισμό.

1.2 Λειτουργικά Συστήματα

Με τον όρο λειτουργικό σύστημα (Operating System - OS) εννοούμε το πρόγραμμα το οποίο αναλαμβάνει να διαμοιράσει τους πόρους του συστήματος στις διάφορες διεργασίες. Βασικός ρόλος λοιπόν του ΛΣ είναι να δίνει χρόνο για εκτέλεση στις εφαρμογές και να διαμοιράζει την κύρια μνήμη όπως και τους λοιπούς πόρους. Επίσης αναλαμβάνει να διατειρεί τα διάφορα συστήματα αρχείων και να παρέχει στις διεργασίες ό,τι μπορεί να χρειαστούν.

Για την επίλυση του προαναφερθήσαντος προβλήματος το ΛΣ πρέπει να λαβμάνει υπ' όψη τις απαιτήσεις των διεργασιών και να παίρνει διάφορες αποφάσεις αναφορικά με τον διαμοιρασμό/την παροχή των πόρων. Βασικό συστατικό ενός ΛΣ αποτελεί ο χρονοδρομολογητής (scheduler). Ο χρονοδρομολογητής παίρνει αποφάσεις βασισμένες σε μια συγκεκριμένη πολιτική και αφορούν τη χρήση του επεξεργαστή από τις διεργασίες με στόχο τη βέλτιστη απόδοση του συστήματος. Η απόδοση από τη μεριά της μπορεί να μεταφράζεται είτε σε καλή αποκρισιμότητα - χαρακτηριστικό απαραίτητο σε συστήματα γενικού σκοπού, είτε σε υψηλό throughput, δηλαδή επεξεργασία μεγάλου όγκου δεδομένων ανα μονάδα χρόνου - απαραίτητο σε συστήματα μεγάλης κλίμακας, όπως servers και υπερυπολογιστές.

Έχουν προταθεί και υλοποιηθεί λοιπόν πολλές πολιτικές/αλγόριθμοι που αποφασίζουν την σειρά και με την οποία οι διάφορες διεργασίες θα καταλάβουν/χρησιμοποιήσουν την κεντρική μονάδα επεξεργασίας (ΠΡΕΠΕΙ ΝΑ ΓΕΜΙΣΟΥΜΕ ΚΑΙ ΣΕΛΙΔΕΣ!!!). Για συστήματα με μόνο μία μονάδα επε-

ξεργασίας, το πρόβλημα ανάγεται στον διαμοιρασμό του χρόνου στις διεργασίες που χρειάζονται τον επεξεργαστή και μετά από έρευνες ετών οι πολιτικές που είχαν προταθεί βελτιστοποιήθηκαν. Με την πρόοδο ωστόσο της τεχνολογίας και την είσοδο στις πολυπύρηνες αρχιτεκτονικές η πολυπλοκότητα του προβλήματος αυξήθηκε καθώς πλέον δεν πρέπει να διαμοιραστεί μόνο ο χρόνος αλλά να αποφασιστεί και ποιος πυρήνας θα διατεθεί στην εκάστοτε διεργασία. Οι υπάρχουσες υλοποιήσεις που χρησιμοποιούνται στα σημερινά λειτουργικά συστήματα δεν λαμβάνουν υπ' όψιν την πολυπλοκότητα του συστήματος και χειρίζονται τους πυρήνες ως ξεχωριστές οντότητες. Αυτή η λογική ενώ είναι αρκετά απλή και εύκολα υλοποιήσιμη, έχει συχνά ως συνέπεια την ταυτόχρονη εκτέλεση διεργασιών που χρησιμοποιούν κοινούς πόρους, με αποτέλεσμα την αύξηση του ανταγωνισμού και τον περιορισμό της απόδοσης του συστήματος.

..

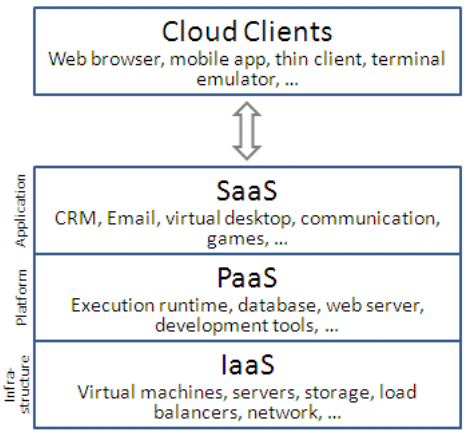
1.3 Υπολογιστικό Νέφος (Cloud Computing)

Τα τελευταία χρόνια προωθείται με νέα τάση, όπου ο χρήστης δεν χρησιμοποιεί μηχανήματα της κατοχής του για υπολογισμούς ή αποθήκευση δεδομένων, αλλά προτιμά οργανισμούς ή εταιρείες που είναι σε θέση να παρέχουν τις εν λόγω υπηρεσίες. Υπολογιστικό νέφος (Cloud Computing) είναι ένας τύπος υπηρεσίας, βασισμένης στο διαδίκτυο, όπου ένας χρήστης μπορεί να χρησιμοποιήσει απομακρυσμένα συστήματα για προσωπική χρήση. Έτσι μία οντότητα που μπορεί να είναι είτε απλός χρήστης είτε οργανισμός ή εταιρεία έχει τη δυνατότητα να μισθώσει μηχανήματα που δεν έχει στην κατοχή της και να γλιτώσει τουλάχιστον αρχικά το κόστος αγοράς και εγκατάστης δικού της κέντρου δεδομένων.

1.3.1 Μοντέλα Νεφών

Τα βασικά μοντέλα υπηρεσιών που παρέχει ένα νέφος χωρίζονται σε αφαιρετικά επίπεδα και είναι τα εξής:

Λογισμικό ως Υπηρεσία - Software as a Service (SaaS). Σε αυτόν τον τύπο, ο τελικός χρήστης / πελάτης έχει την δυνατότητα να χρησιμοποιήσει τις εφαρμογές που του δίνει ο πάροχος και τρέχουν στο νέφος. Οι εφαρμογές γίνονται προσβάσιμες από τον χρήστη μέσω κάποιας διεπαφής (πχ web browser, command-line interface κλπ). Ο πελάτης δεν έχει την δυνατότητα να τροποποιήσει τις υποδομές του νέφους (όπως το δίκτυο, τους αποθηκευτικούς χώρους ή το λειτουργικό σύστημα) παρα μόνο την εφαρμογή που του παρέχεται, τις ρυθμίσεις της και το περιβάλλον της. Τυπικά παραδείγματα SaaS είναι εφαρμογές email, εικονικές επιφάνειες εργασίας και online παιχνίδια.



Πλατφορμα ως Υπηρεσία - Platform as a Service (PaaS). To be filed

Υποδομή ως Υπηρεσία - Infrastructure as a Service (IaaS). Εδώ ο χρήστης έχει τη δυνατότητα να δημιουργήσει και να ελέγξει το σύστημα που θα χρησιμοποιήσει, δηλαδή να επιλέξει χαρακτηριστικά όπως το λειτουργικό σύστημα η τοπολογία του δικτύου κ.ο.κ αλλά και πάλι δεν μπορεί να τροποποιήσει το υποκείμενο σύστημα πάνω στο οποίο τρέχουν οι υπηρεσίες εκτός ίσως από περιορισμένο αριθμό ρυθμίσεων, όπως το τοίχος προστασίας του host (firewall). Τυπικό παράδειγμα αυτού του τύπου των υπηρεσιών είναι η εικονικές μηχανές και ο εικονικός/απομακρυσμένος (?) αποθηκευτικός χώρος. Στην περίπτωση των εικονικών μηχανών, ο χρήστης δημιουργεί εικονικά μηχανήματα, τα οποία υλοποιούνται με κάποιον hypervisor στο απομακρυσμένο σύστημα και ο πελάτης έχει πλήρη πρόσβαση σε αυτά. Ο τρόπος με τον οποίο τα εικονικά μηχανήματα τοποθετούνται στους επεξεργαστές του host είναι το αντικείμενο μελέτης της παρούσας εργασίας.

1.3.2 Πλεονεκτήματα

Ίσως το σημαντικότερο πλεονέκτημα του νέφους είναι η σημαντική εξοικονόμηση κόστους και ενέργειας. Ο χρήστης έχει τις υπηρεσίες που χρειάζεται, όταν τις χρειάζεται χωρίς να απασχολείται για το που και πως θα βρει την υπολογιστική ισχύ που απαιτείται και χωρίς να πρέπει να αγοράσει υπερκοστολογιμένα μηχανήματα. Αντίστοιχα μία εταιρεία μπορεί να είναι πλήρως λειτουργική με σχεδόν μηδενικό κόστος.

Επίσης πολύ σημαντική είναι η ευκολία και η διαθεσιμότητα των υπηρεσιών από οποιοδήποτε σημείο βρίσκεται ο τελικός χρήστης. Με απλές μεθόδους (login) ο χρήστης έχει πλήρη πρόσβαση στις εικονικές μηχανές του ή στα απομακρυσμένα δεδομένα που έχει αποθηκεύσει. Άλλωστε ο πάροχος αναλαμβάνει την προστασία των δεδομένων και την ασφαλή επαναφορά τους μετά από απώλεια, κρατώντας πολλά αντίγραφα στο νέφος. Σε περίπτωση κάποιας αναπόφευκτης αστοχίας, ο τελικός χρήστης δεν θα χρειαστεί να διαθέσει ούτε χρόνο ούτε ανθρώπινο δυναμικό για την επισκευή βλαβών.

Η χρήση του νέφους παρέχει μεγάλη ευελιξία. Το κόστος (είτε χρηματικό, είτε ενεργειακό, είτε εξοπλιστικό) είναι άμεσα συνδεδεμένο με τις !current! ανάγκες του χρήστη. Έτσι αν μία εταιρεία χρειαστεί παραπάνω πόρους για συγκεκριμένο / προκαθορισμένο χρονικό διάστημα το μόνο που έχει να κάνει είναι να τους νοικιάσει από κάποιο πάροχο αντί να υποστεί το πλήρες αντίτιμο για αυτούς (ή κάτι τέτοιο τεσπα)

1.3.3 Μειονεκτήματα

Όπως όλες οι νέες τεχνολογίες, εκτός από θετικά υπάρχουν και κάποια αρνητικά. Με τη χρήση του νέφους, υπάρχει ο -έστω και αμυδρός- κίνδυνος τεχνικής βλάβης για την οποία δεν ευθύνεται ο χρήστης. Τυπικά οι πόροι είναι μόνιμα διαθέσιμοι από διαφορετικές τοποθεσίες, ωστόσο η πρόσβαση σε αυτούς γίνεται μέσω διαδικτύου που ακόμα και σήμερα δεν θεωρείται πάντα δεδομένη.

Επίσης σημαντική είναι η ασφάλεια των δεδομένων. Μία εταιρεία εμπιστεύεται μέρος των δεδομένων της σε τρίτους, χωρίς να μπορεί να διασφαλίσει πως αυτά δεν θα υποκλαπούν. Άλλωστε η κρυπτογράφησή τους μπορεί να μειώσει πολύ την απόδοση του εγχειρήματος. Παρόμοια, η ιδιοτικότητα ευαίσθητων δεδομένων δεν μπορεί να θεωρείται δεδομένη.

1.4 Εικονικές Μηχανές (VM)

Η έννοια των εικονικών μηχανών υπάρχει από την δεκαετία του 1960 αλλά είναι τα τελευταία 10 χρόνια που έχει επανέλθει στο προσκήνιο, κυρίως επειδή η υψηλή υπολογιστική ισχύς των σύγχρονων συστημάτων προσφέρει τη δυνατότητα υλοποίησής τους. Με τον όρο εικονοποίηση, εννοούμε τη δημιουργία εικονικών αντικειμένων (αντί πραγματικών) όπως εικονικές πλατφόρμες, δίκτυα, συσκευές αποθήκευσης κλπ. Τα πλεονεκτήματα από τη χρήση εικονικών μηχανών είναι πολλά. Κάποια είναι περισσότερο προφανή, όπως η εκτέλεση εφαρμογών που δεν έχουν σχεδιαστεί για συγκεκριμένο υπολογιστικό/λειτουργικό σύστημα και η χρήση συσκευών που δεν είναι μέρος ενός συστήματος, αλλά υλοποιημένες σε λογισμικό, ενώ άλλα είναι λιγότερο προφανή και έχουν να κάνουν με την ασφάλεια και την απομόνωση που παρέχει μία εικονική μηχανή. Υπάρχουν τρεις βασικοί/ές τύποι/τεχνικές εικονοποίησης: **Full virtualization**, **Partial Virtualization** και **Paravirtualization**.

Ο/Η πρώτος/η τύπος/τεχνική σηματοδοτεί την πλήρη προσομοίωση του υλικού σε λογισμικό. Αυτό σημαίνει πως το λειτουργικό σύστημα του guest (VM) δεν γνωρίζει πως τρέχει σε προσομοίωση, δηλαδή δεν έχει υποστεί καμία αλλαγή. Αυτού του τύπου η εικονοποίηση παρέχει το πλεονέκτημα πως το λειτουργικό δεν γνωρίζει πως τρέχει σε εικονική μηχανή και συμπεριφέρεται ακριβώς σαν να έτρεχε σε πραγματικό σύστημα. Ωστόσο, αφού όλο το υλικό είναι υλοποιημένο σε λογισμικό, η όλη διαδικασία υστερεί σε απόδοση.

Με την δεύτερη τεχνική η εικονική μηχανή προσομοιώνει επαρκές τμήμα του πραγματικού υποκείμενου υλικού ώστε να επιτρέπει την εκτέλεση ενός μη τροποποιημένου ΛΣ σχεδιασμένου για την ίδια αρχιτεκτονική επεξεργαστή με αυτήν του πραγματικού. Σε αυτήν την περίπτωση δεν χρειάζεται η πλήρης εξομοίωση του συνόλου εντολών του πραγματικού επεξεργαστή και μάλιστα υπό συνθήκες

επιτρέπεται απευθείας εκτέλεση των εντολών του φιλοξενούμενου μηχανήματος στο πραγματικό με την προϋπόθεση να μην επιρρεάζεται κάποιο υποσύστημα έξω από τον άμεσο έλεγχό του. Σε κρίσιμα τμήματα ωστόσο (πχ σε προσπάθεια πρόσβασης σε συσκευή μέσω κλήσης συστήματος) η εποπτεία του hypervisor είναι αναπόφευκτη.

Με την τρίτη τεχνική, το λειτουργικό σύστημα του guest γνωρίζει την ύπαρξη του hypervisor (host) και συνεργάζεται με αυτόν για την καλύτερη απόδοση του συστήματος. Ο hypervisor σε αυτήν την περίπτωση προσφέρει μία διεπαφή στο guest os που του επιτρέπει την (απ)ευθεία(ς) χρήση του υλικού. Για την χρήση ωστόσο αυτής της διεπαφής το guest os πρέπει να υποστεί βασικές αλλαγές. Για να γίνει χρήση αυτής της τεχνικής με ικανοποιητική βελτίωση της απόδοσης, απαιτείται πρόσθετη υποστήριξη από το υλικό (Intel VT-x, AMD-V).

Τα πλεονεκτήματα της χρήσης εικονικών μηχανών είναι πολλά και όπως προαναφέρθηκε κάποια είναι περισσότερο προφανή από άλλα:

- Δημιουργία υλικών συσκευών από λογισμικό. Με χρήση εικονικών μηχανών, ένας προγραμματιστής είναι σε θέση να δημιουργήσει συσκευές χωρίς την αναγκαστική αγορά ακριβού υλικού.
- Εκτέλεση εφαρμογών σχεδιασμένων για συστήματα διαφορετικά από αυτό του host. Είναι απλή η προσομοίωση διάφορων αρχιτεκτονικών με συνέπεια ο χρήστης να έχει τη δυνατότητα να χρησιμοποιήσει λογισμικό που δεν είναι σχεδιασμένο για τον host. Με την ίδια λογική μπορεί ο χρήστης να εκτελέσει εφαρμογές γραμμένες για ένα λειτουργικό σύστημα σε κάποιο άλλο (πχ παιχνίδια γραμμένα για Windows μπορούν να τρέξουν σε host με Linux).
- Ασφάλεια του host μηχανήματος. Ο χρήστης έχει πλήρη πρόσβαση (administrator/root) στο δικό του εικονικό μηχάνημα, ωστόσο στον host διατηρεί δικαιώματα απλού χρήστη.
- Απομόνωση των εικονικών μηχανών. Ένα εικονικό μηχάνημα μπορεί να τρέχει στο ίδιο φυσικό μηχάνημα μαζί με πολλά άλλα. Ωστόσο ο χώρος του κάθε ενός είναι ιδιοτικός και δεν επιρρεάζεται από κάποιο άλλο μηχάνημα.
- Αποθήκευση μίας κατάστασης και επαναφορά σε αυτήν (πχ ύστερα από κάποια αστοχία (failover)) με χρήση στιγμιοτύπων (snapshot). Η χρήση στιγμιοτύπων επεξηγήται αναλυτικότερα στη συνέχεια.
- Χρήση νέων μηχανημάτων δίχως την αγορά νέου υλικού.
- Πειραματισμός και χρήση στην εκπαίδευση Λειτουργικών Συστημάτων χωρίς τον φόβο της καταστροφής ενός συστήματος από άπειρους/απείραστους χρήστες. Όπως αναφέρθηκε προηγουμένως, η χρήση εικονικών μηχανών προσφέρει ασφάλεια στον host. Όταν λοιπόν θέλουμε να αλλάξουμε το λειτουργικό σύστημα και να το προσαρμόσουμε στις ανάγκες μας, στο χειρότερο σενάριο θα επηρεαστεί μόνο το εικονικό μηχάνημα και όχι ο host.

Όταν ο χρήστης θέλει να αποθηκεύσει την κατάσταση της εικονικής μηχανής του, αποθηκεύει ένα **στιγμιότυπο (snapshot)**. Με το στιγμιότυπο το σύνολο των δεδομένων (τόσο από τους σκληρούς δίσκους όσο και από την μνήμη RAM) του εικονικού μηχανήματος αποθηκεύονται και μπορούν σε μελλοντικό χρόνο να επαναφερθούν και η χρήση του μηχανήματος να συνεχίσει από εκεί. Αυτό είναι και από τα χρησιμότερα χαρακτηριστικά μίας εικονικής μηχανής καθώς καθιστά την επαναφορά της σε παρελθοντική κατάσταση ιδιαίτερα εύκολη (πχ μετά από κάποια αστοχία).

1.5 Προβλημα

Το πρόβλημα ορίζεται ως εξής... Μπλα μπλα

Κεφάλαιο 2

Προηγούμενο

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

2.1 Η γλώσσα προγραμματισμού C

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα

μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

2.2 Σημασιολογία γλωσσών προγραμματισμού

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

2.3 Θεωρία πεδίων

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

Μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα, μπλα μπλα, μπλα μπλα μπλα, μπλα, μπλα μπλα μπλα.

Βιβλιογραφία

[Appe00] Andrew W. Appel and Amy P. Felty, "A Semantic Model of Types and Machine Instructions for Proof-Carrying Code", in *Proceedings of the 27th Annual Symposium on Principles of Programming Languages (POPL 2000)*, pp. 243–253, ACM Press, 2000.

[Appe01] A. W. Appel, "Foundational Proof-Carrying Code", in *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, pp. 247–258, June 2001.

[Bohm85] C. Böhm and A. Bernarducci, "Automatic Synthesis of Typed λ-Programs on Term Algebras", *Theoretical Computer Science*, vol. 39, no. 2–3, pp. 135–154, August 1985.

[Chur32] A. Church, "A Set of Postulates for the Foundations of Logic", *Annals of Mathematics*, vol. 33, no. 1, pp. 346–366, 1932.

[Chur33] A. Church, "A Set of Postulates for the Foundations of Logic", *Annals of Mathematics*, vol. 34, no. 2, pp. 839–864, 1933.

[Gira72] J.-Y. Girard, *Interprétation Fonctionelle et Élimination des Coupures Dans l'Arithmétique d'Ordre Supérieur*, Ph.D. thesis, Université Paris 7, 1972.

[Gira89] J.-Y. Girard, Y. Lafont and P. Taylor, "Proofs and Types", *Tracks in Theoretical Computer Science*, 1989.

[Harp95] Robert Harper and Greg Morrisett, "Compiling Polymorphism Using Intensional Type Analysis", in *Proceedings of the 22nd Annual Symposium on Principles of Programming Languages (POPL 1995)*, pp. 130–141, ACM Press, 1995.

[Morr98] Greg Morrisett, David Walker, Karl Crary and Neal Glew, "From System F to Typed Assembly Language", in *Proceedings of the 25th Annual Symposium on Principles of Programming Languages (POPL 1998)*, pp. 85–97, ACM Press, January 1998.

[Necu96] G. Necula and P. Lee, "Safe Kernel Extensions without Run-Time Checking", in *Proceedings of the 2nd USENIX Symposium on Operating System Design and Implementation*, pp. 229–243, USENIX Association, 1996.

[Necu97] G. Necula, "Proof-Carrying Code", in *Proceedings of the 24th Annual Symposium on Principles of Programming Languages (POPL 1997)*, pp. 106–119, New York, January 1997, ACM Press.

[Necu98] G. Necula, *Compiling with Proofs*, Ph.D. thesis, Carnegie Mellon University, September 1998.

[Paul89] C. Paulin-Mohring, *Extraction de Programmes Dans le Calcul des Constructions*, Ph.D. thesis, Université Paris 7, January 1989.

[Paul93] Christine Paulin-Mohring, "Inductive Definitions in the System Coq: Rules and Properties", in M. Bezem and J. F. Groote, editors, *Proceedings of the 1st Int. Conf. on Typed Lambda Calculi and Applications, TLCA'93, Utrecht, The Netherlands, 16–18 March 1993*, vol. 664, pp. 328–345, Springer-Verlag, Berlin, 1993.

[Pfen90] F. Pfenning and C. Paulin-Mohring, "Inductively defined types in the Calculus of Constructions", in *Proceedings of Mathematical Foundations of Programming Semantics*, vol. 442 of *Lecture Notes in Computer Science*, Berlin, 1990, Springer-Verlag. technical report CMU-CS-89-209.

[Reyn74] John. C. Reynolds, "Towards a Theory of Type Systems", in Ehring et al., editor, *Lecture Notes in Computer Science*, vol. 19, pp. 408–425, Springer-Verlag, 1974.

[Sell94] M. P. A. Sellink, "Verifying process algebra proofs in type theory", in D. J. Andrews, J. F. Groote and C. A. Middelburg, editors, *Proceedings of the International Workshop on Semantics of Specipication Languages (SOSL 1993)*, Springer, 1994.

[Shao02] Z. Shao, B. Saha, V. Trifonov and N. Papaspyrou, "A Type System for Certified Binaries", in *Proceedings of the 29th Annual Symposium on Principles of Programming Languages (POPL 2002)*, pp. 217–232, Portland, OR, USA, January 2002.

Παράρτημα Α

Ευρετήριο συμβολισμών

$A \to B$: συνάρτηση από το πεδίο A στο πεδίο B.

Παράρτημα Β

Ευρετήριο γλωσσών

Haskell : η γλώσσα της ζωής μου.

Παράρτημα C

Ευρετήριο αριθμών

42 : life, the universe and everything.