ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΡΟΤΥΠΟΠΟΙΗΣΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ EXT2 ΜΕ ΧΡΗΣΗ ΤΥΠΙΚΩΝ ΜΕΘΟΔΩΝ

<<Ρουμπάτι Ματέο>>

Α.Μ. 71347122

Εισηγητής: Κωνσταντίνος Μπάρλας, Ε.ΔΙ.Π.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΡΟΤΥΠΟΠΟΙΗΣΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ ΑΡΧΕΙΩΝ EXT2 ΜΕ ΧΡΗΣΗ ΤΥΠΙΚΩΝ ΜΕΘΟΔΩΝ

Ρουμπάτι Ματέο

Α.Μ. 71347122

Εισηγητής:

Κωνσταντίνος Μπάρλας, Ε.ΔΙ.Π.

Konstantinos Barlas
Digitally signed by Konstantinos Barlas
Date: 2023.07.20 15:33:32 +03'00'

Εξεταστική Επιτροπή:

Φατούρος Σταύρος: Αναπληρωτής Καθηγητής

Stavros Fatouros
Digitally signed by Stavros Fatouros
Date: 2023.09.13 08:48:17 +03'00'

Τρούσσας Χρήστος: Επίκουρος Καθηγητής

Christos Troussas
Digitally signed by Christos Troussas
Date: 2023.07.20 15:46:59 +03'00'

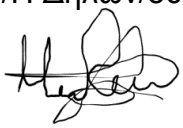
Ημερομηνία εξέτασης 18/7/2023

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Ρουμπάτι Ματέο του Μάρκο, με αριθμό μητρώου 71347122 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Βεβαιώνω ότι είμαι συγγραφέας αυτής της Διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμενες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο/Η Δηλών/ούσα



ΕΥΧΑΡΙΣΤΙΕΣ

Κατ'αρχάς θα ήθελα να πω ένα μεγάλο ευχαριστώ στους γονείς μου και τους φίλους που με στήριξαν όλα αυτά τα χρόνια. Και να ευχαριστήσω στον κύριο Μπάρλα για την βοήθεια του και την καθοδήγησή του στην παρούσα εργασία.

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία ασχολείται με την χρήση των Τυπικών Μεθόδων μέσα από την προτυποποίηση της διαδικασίας δημιουργίας αρχείων που χρησιμοποιεί το σύστημα αρχείων ext2. Μέσα στην εργασία αυτή παρουσιάζονται πληροφορίες σχετικά με τις τυπικές μεθόδους και τις πρακτικές που τις απαρτίζουν καθώς και διάφορες περιπτώσεις μελέτης που εξετάζουν την προδιαγραφή που δημιουργήθηκε για την εργασία αυτή. Σκοπός της εργασίας είναι η παρουσίαση των πρακτικών αυτών και η συνεισφορά των τυπικών μεθόδων στην μηχανική λογισμικού και το πώς μπορεί κάποιος μέσα από τις πρακτικές αυτές να σχεδιάσει και να υλοποιήσει ένα αξιόπιστο σύστημα λογισμικού.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Λογισμικού

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Τυπικές μέθοδοι, Τυπικές προδιαγραφές, Σύστημα αρχείων ext2

ABSTRACT

This thesis deals with the use of the Formal Methods through the standardization of the file creation process within the ext2 file system. In this paper, information is presented about the formal methods and practices that make them up, as well as various case studies that examine the specification created for this paper. The purpose of the work is the presentation of these practices and the contribution of standard methods to engineering software and how one can design and use a reliable software system through these practices.

SCIENTIFIC AREA: Software Engineering
KEY WORDS: Formal Methods, Fromal Specifications, ext2 file system

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

Εισαγωγή

Με την εμφάνιση νέων τεχνολογιών και την αυξανόμενη πολυπλοκότητα των συστημάτων λογισμικού, υπάρχει μια αυξανόμενη ανάγκη για αξιόπιστες και αποτελεσματικές πρακτικές ανάπτυξης λογισμικού. Οι τυπικές προδιαγραφές είναι μια τέτοια πρακτική που έχει κερδίσει αυξανόμενη προσοχή τα τελευταία χρόνια, λόγω της ικανότητάς της να διασφαλίζει την ορθότητα και την αξιοπιστία των συστημάτων λογισμικού. Αυτό οφείλεται στο γεγονός ότι οί τυπικές μέθοδοι χρησοιμοποιούν μαθηματικές προτάσεις για την περιγραφή της συμπεριφοράς και της λειτουργικότητας ενός συστήματος λογισμικού, με τρόπο που είναι ακριβής, σαφής και αναγνώσιμος από μηχανή. Οι τυπικές μέθοδοι μπορούν να χρησιμοποιηθούν για την επαλήθευση ότι ένα σύστημα πληροί τις απαιτήσεις του, για τον εντοπισμό σφαλμάτων και πιθανών προβλημάτων νωρίς στη διαδικασία ανάπτυξης. Αυτό έχει ως αποτέλεσμα η χρήση των τυπικών μεθόδων να γίνεται ολοένα και πιο σημαντική στην ανάπτυξη πολύπλοκων συστημάτων λογισμικού, όπου οι παραδοσιακές μέθοδοι δοκιμών και επαλήθευσης μπορεί να μην επαρκούν για να διασφαλίσουν την ορθότητα και την αξιοπιστία του συστήματος. Παρέχοντας μια σαφή και ξεκάθαρη περιγραφή της συμπεριφοράς και της λειτουργικότητας ενός συστήματος, οι τυπικές μέθοδοι μπορούν να βοηθήσουν σημαντικά τους προγραμματιστές να εντοπίζουν πιο εύκολα σφάλματα και πιθανά προβλήματα και να παρέχουν μια αυστηρή βάση για δοκιμές και επαλήθευση.

1.1 Σκοπός Της Διπλωματικής Εργασίας

Στη παρούσα διπλωματική εργασία, θα γίνει εφαρμογή των τυπικών μεθόδων σε ένα μέρος του συστήματος αρχείων Ext2. Αυτό περιλαμβάνει τη δημιουργία μιας Τυπικής Προδιαγραφής του συστήματος και τη χρήση τεχνικών και εργαλείων για την επικύρωση και την επαλήθευση των προδιαγραφών. Μέσω της διαδικασίας αυτής , θα εξεταστεί η πρακτική αξία των τυπικών προδιαγραφών στην ανάπτυξη αξιόπιστων και υψηλής ποιότητας συστημάτων λογισμικού . Συνολικά, η διπλωματική εργασία έχει σκοπό να συμβάλλει στον τομέα της μηχανικής λογισμικού, επιδεικνύοντας την αποτελεσματικότητα των τυπικών προδιαγραφών στην ανάπτυξη σύνθετων συστημάτων λογισμικού, χρησιμοποιώντας το σύστημα αρχείων Ext2 ως μελέτη περίπτωσης.

1.2 Δομή της Διπλωματικής Εργασίας

Στα κεφάλαια που ακολουθούν, θα παρουσιαστούν τα δομικά στοιχεία του συστήματος αρχείων ext2, τα χαρακτηριστικά και οι αρχές των τυπικών μεθόδων καθώς και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της τυπικής προδιαγραφής. Στην συνέχεια θα παρουσιαστεί η υλοποίηση της προδιαγραφής μαζί με τα αποτελέσματα και τα συμπεράσματα που προέκυψαν κατά την διάρκεια της έρευνας.

Επεξήγηση Συστημάτων Αρχείων

Αυτό το κεφάλαιο παρέχει μια εις βάθος επισκόπηση του συστήματος αρχείων ext2, της δομής και των χαρακτηριστικών του. Το κεφάλαιο ξεκινά παρουσιάζοντας τις βασικές έννοιες των συστημάτων αρχείων και τον τρόπο με τον οποίο σχετίζονται με το σύστημα αρχείων ext2. Στη συνέχεια, παρουσιάζεται η εσωτερική οργάνωση του συστήματος αρχείων ext2, συμπεριλαμβανομένων των λειτουργιών και των αντικειμένων που το απαρτίζουν. Συνολικά, αυτό το κεφάλαιο παρέχει μια περιεκτική επισκόπηση του συστήματος αρχείων ext2, προσφέροντας τις πληροφορίες σχετικά με τη σχεδίαση και την υλοποίησή του που είναι απαραίτητες για την ανάλυση και των σχεδιασμό των τυπικών προδιαγραφών που παρουσιάζονται στην εν λόγω διπλωματική εργασία.

2.1 Τι είναι ένα σύστημα αρχείων

΄Ένα σύστημα αρχείων είναι οι μέθοδοι και οι δομές δεδομένων που χρησιμοποιεί ένα λειτουργικό σύστημα για να παρακολουθεί αρχεία στον δίσκο ή σε κάποιο διαμέρισμα του δίσκου , δηλαδή τον τρόπο οργάνωσης των αρχείων στο δίσκο. Η λέξη χρησιμοποιείται επίσης για αναφορά σε κάποιο διαμέρισμα του δίσκου ή στον ίδιο των δίσκο που χρησιμοποιείται για την αποθήκευση των αρχείων ή τον τύπο του συστήματος αρχείων' [1, σ. 36].

Σε υψηλό επίπεδο, ένα σύστημα αρχείων αποτελείται από δύο κύρια στοιχεία[2]: **τις κλήσεις συστήματος αρχείων** και τη **δομή του συστήματος αρχείων**. Οι κλήσεις συστήματος αρχείων είναι υπεύθυνες για την αλληλεπίδραση με την μονάδα αποθήκευσης και τη μετάφραση των αιτημάτων του συστήματος αρχείων σε λειτουργίες δίσκου [2]. Αντίστοιχα, η δομή του συστήματος αρχείων καθορίζει τον τρόπο με τον οποίο τα αρχεία και οι κατάλογοι οργανώνονται και αποθηκεύονται στο δίσκο [3].

Σύμφωνα, με τον Tanenbaum [3], τα αρχεία συνήθως οργανώνονται σε καταλόγους, οι οποίοι μπορούν να περιέχουν τόσο αρχεία όσο και υποκαταλόγους. Κάθε αρχείο και κατάλογος αναγνωρίζεται από μια μοναδική διαδρομή(path), η οποία καθορίζει τη θέση του εντός της ιεραρχίας του συστήματος αρχείων και ο ριζικός κατάλογος(root directory) είναι ο κατάλογος ανώτατου επιπέδου του συστήματος αρχείων και περιέχει όλους τους άλλους καταλόγους και αρχεία [4].

Όταν δημιουργείται ένα αρχείο σύμφωνα με τον Bach [2], ο πυρήνας του λειτουργικού συστήματος δέχεται την κλήση από το σύστημα αρχείων και στην συνέχεια δεσμεύει των κατάλληλο χώρο στο δίσκο για να αποθηκεύσει τα περιεχόμενά του νέου αρχείου και τα μεταδεδομένα του και στην συνέχεια, τα μεταδεδομένα αποθηκεύονται στη δομή του συστήματος αρχείων που το αντιπροσωπεύει [2]. Πέρα όμως από την δημιουργία αρχείων οι κλήσεις συστήματος αρχείων πραγματοποιούν και διαδικασίες όπως είναι η ανάκτηση, η τροποποίηση και η διαγραφή αρχείων αλλά και πιό προηγμένες λειτουργίες όπως είναι η αφαίρεση διπλότυπων καταχωρήσεων και η αλλαγή δικαιωμάτων χρήστη σε ένα αρχείο [2,3] .

Συνοπτικά, ένα σύστημα αρχείων είναι ένα κρίσιμο στοιχείο οποιουδήποτε λειτουργικού συστήματος που επιτρέπει την οργάνωση και αποθήκευση δεδομένων στις συσκευές αποθήκευσης ενός υπολογιστή [1]. Τα δύο κύρια στοιχεία του, οι κλήσεις συστήματος αρχείων που είναι υπεύθυνες για την αλληλεπίδραση συστήματος αρχείων και λειτουργικού συστήματος και η δομή του συστήματος αρχείων, που συνεργάζονται για να παρέχουν μια διεπαφή για τους χρήστες και τις εφαρμογές για την αλληλεπίδραση με τα αρχεία και τους καταλόγους [2].

2.2 Περιγραφή Ext2

Το σύστημα αρχείων Ext2 είναι ένα σύστημα αρχείων βασισμένο σε Unix και αποτελεί τη δεύτερη έκδοση του συστήματος αρχείων Extended και αναπτύχθηκε από τους Remy Card, Theodore Ts'o και Stephen Tweedie το 1993 [5]. Το σύστημα αρχείων ext2 για αρκετά χρόνια ήταν το πιο διαδεδομένο σύστημα αρχείων σε λειτουργικά συστήματα Unix και σύμφωνα με τους δημιουργούς του, ο σκοπός του ext2 ήταν να αντιμετωπίσει τα προβλήματα του προκατόχου του (extended file system) προσφέροντας στους χρήστες ένα πιο ισχυρό και στιβαρό σύστημα αρχείων χωρίς ο χρήστης να χρειαστεί να κάνει κάποια σημαντική αλλαγή στο σύστημα αρχείων που χρησιμοποιεί ήδη [4,5].

Όλα τα συστήματα αρχείων που βασίζονται σε Unix έχουν κάποια συγκεκριμένα χαρακτηριστικά. Αυτά τα χαρακτηριστικά είναι θεμελιώδη για την υλοποίηση ενός τέτοιου συστήματος αρχείων [1]. Όπως αναφέρθηκε και πριν το ext2 είναι ένα σύστημα αρχείων βασισμένο σε Unix και επομένως το ext2 φέρει αυτά τα χαρακτηριστικά. Το πρώτο χαρακτηριστικό που φέρει το ext2 με τα υπόλοιπα συστήματα αρχείων Unix, είναι ότι και τα δύο συστήματα χρησιμοποιούν τις ίδιες δομές για να αναπαραστήσουν τα αντικείμενα που απαρτίζουν το ίδιο το σύστημα αρχείων [5]. Οι δομές αυτές είναι το **Block** για την αποθήκευση δεδομένων, το **inode** για την αναπαράσταση των αρχείων και τέλος το **superblock** που περιέχει όλες τις σημαντικές πληροφορίες του συστήματος [1]. Το δεύτερο χαρακτηριστικό είναι ότι το ext2 υποστηρίζει τους ίδιους, τύπους αρχείων (directories, regular files, symbolic links) που υποστηρίζουν τα συστήματα αρχείων Unix καθώς και την χρήση καταλόγων (directories) για την εσωτερική οργάνωση συστήματος ακολουθώντας μια ιεραρχική δομή δέντρου [5].

Τα χαρακτηριστικά που διαφοροποιούν το ext2 και το κάνουν πιο αξιόπιστο και στιβαρό σύστημα αρχείων σε σχέση με τον προκάτοχό του, βασίζονται στην διάταξη του συστήματος στον δίσκο και στην χρήση του VFS (Virtual file System) [5]. Όσον αφορά τη διάταξη του συστήματος στον δίσκο, οι δημιουργοί βασίστηκαν σε αυτήν που έχει το σύστημα αρχείων BSD [7]. Πιο συγκεκριμένα, ένα πλήθος από blocks ομαδοποιούνται δημιουργώντας τα λεγόμενα **block groups** και κάθε block group , περιέχει ένα αντίγραφο από το superblock και των άλλων αντικειμένων που απαρτίζουν το σύστημα αρχείων και αυτό, έχει ως αποτέλεσμα να παρέχει στον χρήστη την δυνατότητα ανάκτησής του συστήματος σε περίπτωση προβλήματος με το αρχικό superblock αλλά και συνεισφέρει στην μείωση χρόνων, της αναζήτησης κεφαλής του δίσκου όταν γίνεται κάποια διεργασία που προέρχεται από το I/O ή όταν ο χρήστης θέλει να διαβάσει ή να γράψει κάποιο αρχείο στον δίσκο [5].

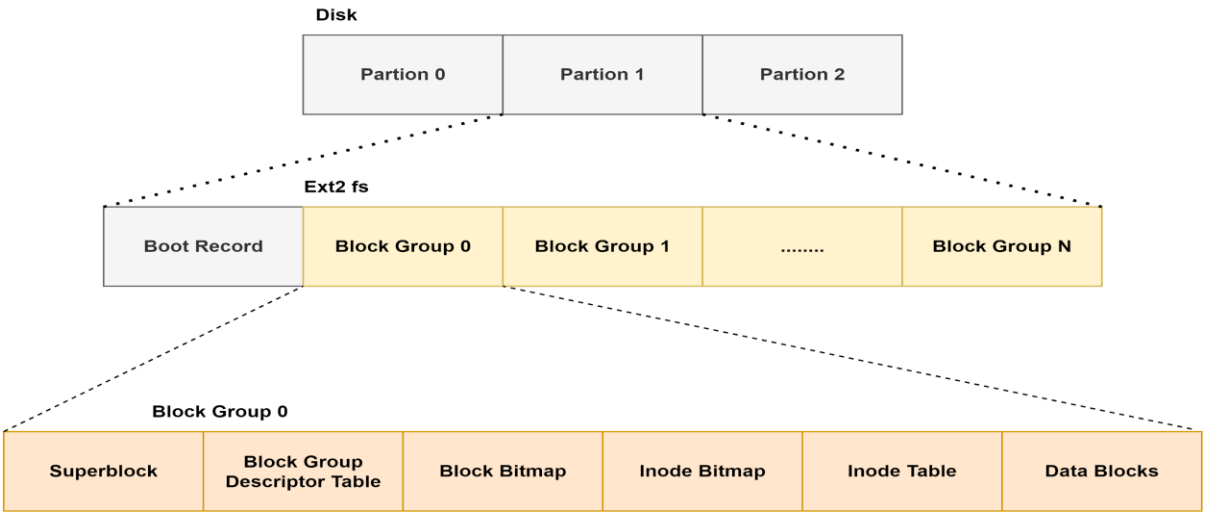
Χρησιμοποιώντας το VFS οι δημιουργοί του συστήματος , κατάφεραν να ξεπεράσουν το εμπόδιο για το μέγιστο μέγεθος του συστήματος που ήταν στα 2Gb και κατάφεραν να φτάσουν το όριο στα 4 Tb. Αυτό έδινε την δυνατότητα στους χρήστες να μπορούν να χρησιμοποιήσουν το σύστημα αρχείων σε μεγαλύτερης χωρητικότητας δίσκους χωρίς να χρειάζεται να κάνουν κάποια διχοτόμηση στον δίσκο[5]. Μέσα από αυτά τα χαρακτηριστικά το ext2 κατάφερε να γίνει ένα από τα πιό διαδεδομένα συστήματα αρχείων καθώς και οι μετέπειτα εκδόσεις του (ext3, ext4) χρησιμοποιούνται μέχρι και σήμερα.

2.3 Η δομή και τα αντικείμενα του ext2

Με βάση την ανάλυση του Poirier [6], το σύστημα αρχείων ext2 αποτελείται από τα εξής 6 στοιχεία :

1. Superblock
2. Block group descriptor table
3. Block bitmap
4. Inode bitmap
5. Inode table
6. Directory linked list

Τα στοιχεία από το 1 εώς το 5 αποτελούν στοιχεία που δομούν το σύστημα αρχείων στην μονάδα αποθήκευσης και το έκτο στοιχείο δίνει την δομή και την οργάνωση των αρχείων μέσα στο σύστημα. Στην Εικόνα 2.1 βλέπουμε την δομή συστήματος στον δίσκο όπως έχει οριστεί από τους δημιουργούς του [5].

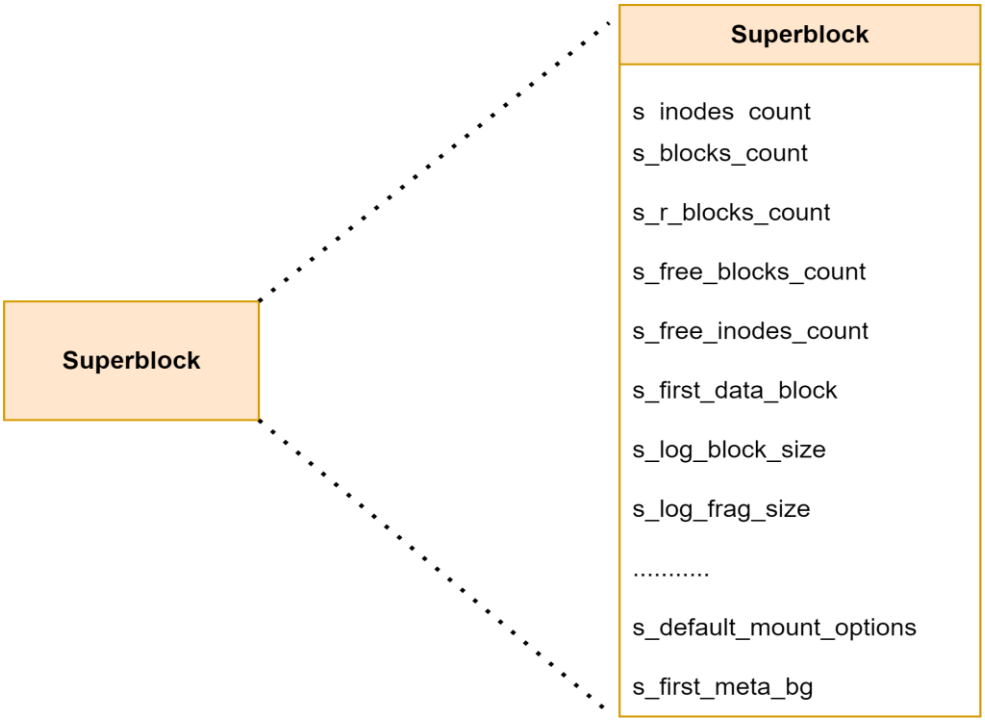


Εικόνα 2.1 : Αναπαράσταση συστήματος αρχείων ext2 στον δίσκο

2.3.1 Superblock

Σύμφωνα με την Εικόνα 2.1 το πρώτο στοιχείο στην μονάδα αποθήκευσης είναι το superblock [5]. Το superblock είναι η θεμελιώδης δομή του συστήματος αρχείων καθώς εκεί βρίσκονται όλες οι πληροφορίες που χρειάζεται το σύστημα αρχείων για να λειτουργήσει και βρίσκεται στο πρώτο block

group (block group 0) στο data block 1 και έχει μέγεθος 1KiB[1] [8]. Στις πρώτες εκδόσεις του συστήματος αρχείων ext2 κάθε block group έχει ένα αντίγραφο από το superblock ενώ στις μετέπειτα εκδόσεις το αντίγραφο του superblock βρίσκεται μόνο σε συγκεκριμένα block groups [6]. Οι πληροφορίες που περιέχει το superblock, έχουν να κάνουν με το συνολικό πλήθος των ελεύθερων data blocks και inodes στο σύστημα, το πόσα blocks groups υπάρχουν , το πόσα inode και data block έχει κάθε block group αλλά και πληροφορίες που έχουν να κάνουν με την διαμόρφωση του συστήματος αρχείων όπως π.χ. την έκδοση του συστήματος αρχείων και το πότε το σύστημα αρχείων έγινε mount [8]. Επίσης κάθε φορά που γίνεται κάποια αλλαγή ή τροποποίηση στο σύστημα (δημιουργία ή εγγραφή αρχείων) το superblock ενημερώνεται ώστε να αντικατοπτρίζει τις αλλαγές που έγιναν στον μονάδα αποθήκευσης [6].



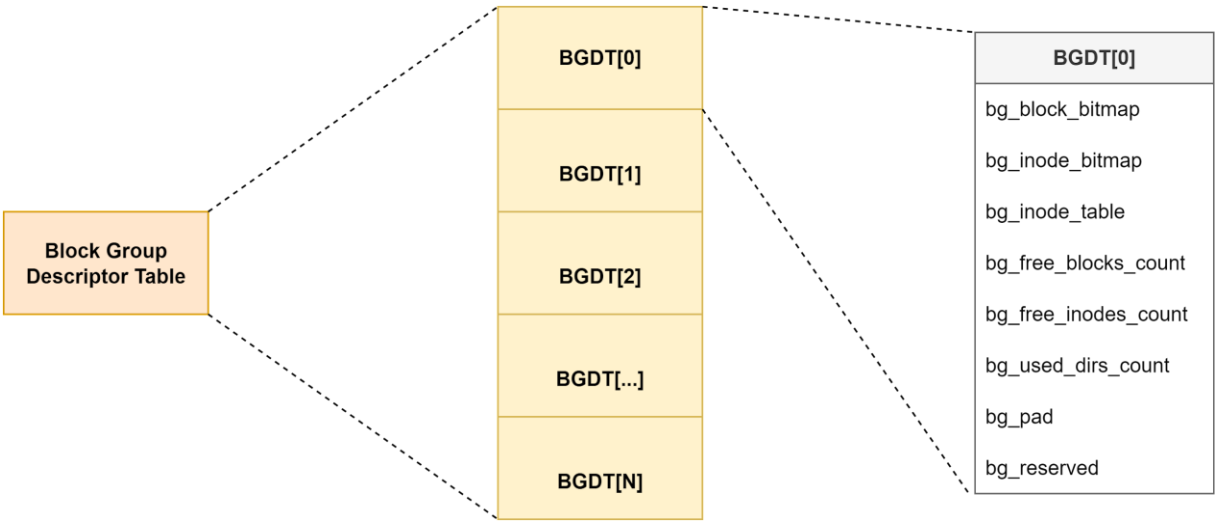
Εικόνα 2.2 : Εσωτερική δομή superblock

2.3.2 Block Group Descriptor Table

Το δεύτερο στοιχείο κατά σειρά είναι το block group descriptor table και βρίσκεται στο επόμενο block μετά το superblock [5]. Σύμφωνα με την ανάλυση του Poirier [6], το block group descriptor table είναι ένας πίνακας ο οποίος περιέχει όλα τα στοιχεία ενός block group και κάθε στοιχείο του πίνακα, αντιστοιχεί με ένα block group. Μερικές από τις πληροφορίες που έχει ο πίνακας , είναι το πλήθος των

[1] εδώ είναι σημαντικό να αναφερθεί ότι το μέγεθος του superblock είναι πάντα 1KiB ανεξάρτητα από το μέγεθος που έχουν τα data blocks.

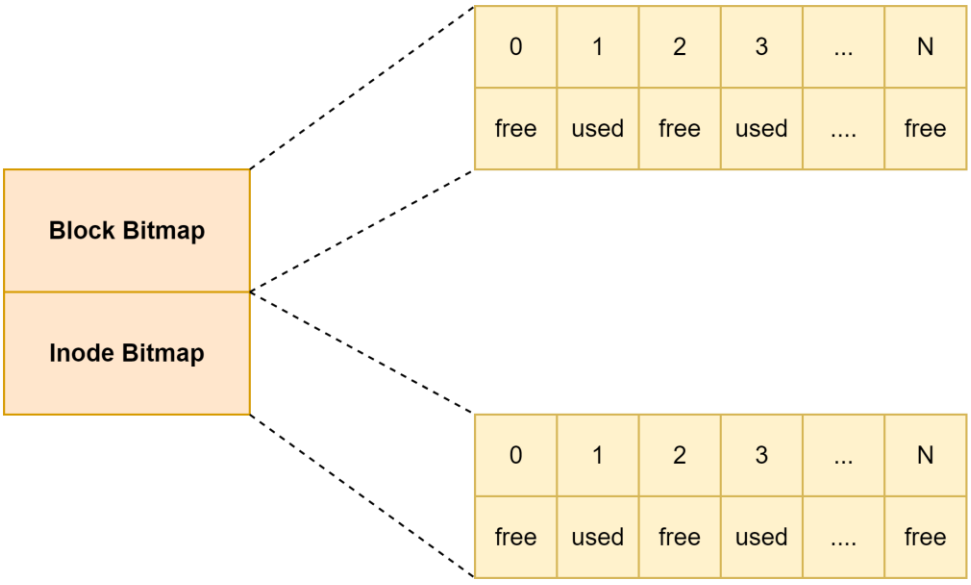
ελεύθερων inode και data block στο εν λόγω block group , το πρώτο ελεύθερο block στο block bitmap και τέλος το πρώτο ελεύθερο inode στο inode table και inode bitmap [8].



Εικόνα 2.3 : Δομή Block Group Descriptor Table

2.3.3 Bitmaps

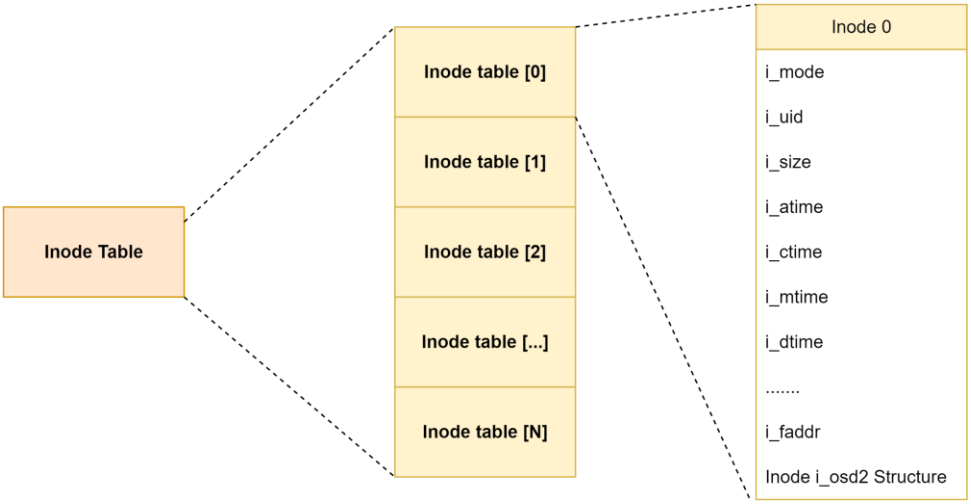
Τα επόμενα δύο στοιχεία στην Εικόνα 2.1 είναι τα inode και block bitmap [5]. Και τα δύο στοιχεία λειτουργούν ακριβώς με τον ίδιο τρόπο. Το bitmap είναι ένας πίνακας από bits και κάθε bit ανάλογα με την τιμή που έχει δηλώνει αν το inode/block είναι ελεύθερο ή δεσμευμένο [6]. Επίσης η κάθε θέση που έχει το bit στον πίνακα , αντιστοιχεί είτε στην θέση που έχει το inode στο inode table ή τον αριθμό του block στην μονάδα αποθήκευσης [6].



Εικόνα 2.4 : Αναπαράσταση Block και Inode Bitmap

2.3.4 Inode Table

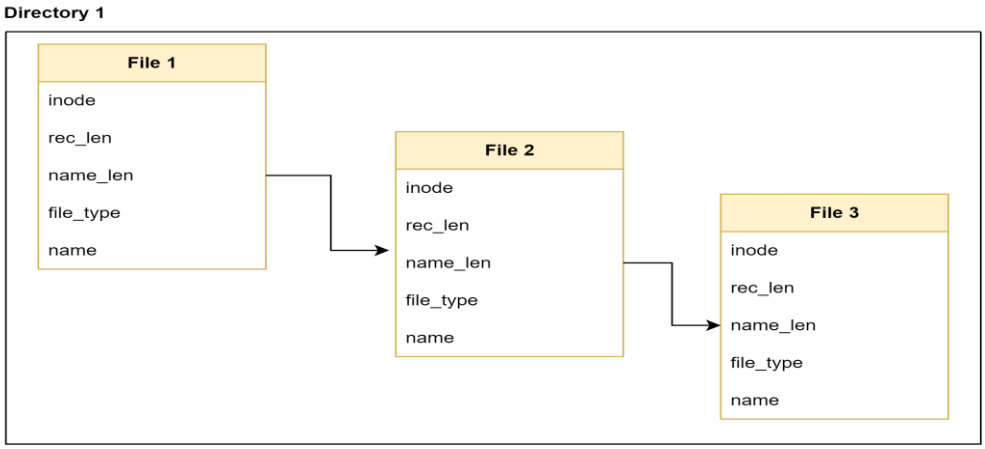
Το τελευταίο στοιχείο για την οργάνωση του συστήματος αρχείων στον δίσκο, είναι το inode table. Το inode table σύμφωνα με τον Poirier [6], είναι ένας πίνακας που αποτελείται από inodes και χρησιμοποιείται για την παρακολούθηση των αρχείων στο σύστημα. Το inode είναι η δομή που χρησιμοποιεί το σύστημα αρχείων για να περιγράψει ένα αρχείο στο σύστημα και περιέχει πληροφορίες όπως είναι το μέγεθος του αρχείου, ο τύπος του αρχείου (regular file,directory) καθώς και το σε ποια blocks βρίσκονται τα περιεχόμενα του αρχείου [5].



Εικόνα 2.5: Δομή Inode και Inode Table

2.3.5 Directory Linked List

Το directory linked list, είναι η τελευταία δομή που χρησιμοποιείται από το σύστημα αρχείων ext2. Το directory linked list όπως λέει και το όνομά του είναι μία συνδεδεμένη λίστα η οποία περιγράφει την εσωτερική δομή ενός καταλόγου στο σύστημα αρχείων. Οι περιγραφή αυτή περιέχει το όνομα του αρχείου το μέγεθος του ονόματος του αρχείου, το inode το οποίο περιγράφει το αρχείο και το τον τύπο του αρχείου [5]. Στης μετέπειτα εκδόσεις του συστήματος αρχείων ext2 η δομή αυτή αντικαταστάθηκε με το Indexed Directory Format καθώς όταν ο κατάλογος μεγάλωνε σε μέγεθος η διαδικασία της ανάκτησης του αρχείου γίνονταν χρονοβόρα [6].



Εικόνα 2.6 : Δομή directory linked list

Τυπικές Μέθοδοι

Οι Τυπικές Μέθοδοι γίνονται ολοένα και πιο δημοφιλείς στην ανάπτυξη λογισμικού λόγω της ικανότητάς τους να διασφαλίζουν την ορθότητα, την αξιοπιστία και την ασφάλεια των συστημάτων λογισμικού. Σε αυτό το κεφάλαιο, θα διερευνηθούν οι αρχές και οι πρακτικές των Τυπικών Μεθόδων, συμπεριλαμβανομένης της χρήσης Τυπικών Γλωσσών και τεχνικών για τον προσδιορισμό, την επαλήθευση και την επικύρωση συστημάτων λογισμικού. Επίσης στο κεφαλαιο αυτο θα συζητηθουν τα οφέλη από τη χρήση των τυπικών μεθόδων, συμπεριλαμβανομένης της αυξημένης εμπιστοσύνης στην ορθότητα του λογισμικού, της μείωσης των σφαλμάτων και της βελτιωμένης δυνατότητας για την συντήρηση του λογισμικού καθώς και διάφορες τεχνικές και εργαλεία που χρησιμοποιούνται σε Τυπικές Μεθόδους, όπως είναι ο έλεγχος μοντέλων, και η απόδειξη θεωρημάτων. Συνολικά, αυτό το κεφάλαιο παρέχει μια περιεκτική επισκόπηση των Τυπικών Μεθόδων στη μηχανική λογισμικού και υπογραμμίζει τη σημασία τους για την ανάπτυξη συστημάτων λογισμικού υψηλής ποιότητας και αξιοπιστίας.

3.1 Επισκόπηση Τυπικών Μεθόδων

Οι τυπικές μέθοδοι είναι ένα σύνολο από πρακτικές στη μηχανική λογισμικού που στοχεύουν στην αύξηση της αξιοπιστίας και της ορθότητας των συστημάτων λογισμικού [9]. Για να περιγράψει κάποιος ένα σύστημα με τυπικές μεθόδους πρέπει να κάνει χρήση μιας τυπικά ορισμένης γλώσσας της οποίας το συντακτικό, το λεξιλόγιο και το σημασιολογικό βασίζεται στα μαθηματικά [10]. Ο κλάδος των μαθηματικών που χρησιμοποιείται για τον ορισμό μιας τέτοιας γλώσσας είναι τα διακριτά μαθηματικά και οι μαθηματικές έννοιες της άλγεβρας, της λογικής και της θεωρίας συνόλων. Η χρήση των τυπικών μεθόδων παρέχει μια ακριβής σημασιολογία για το σύστημα στο αφηρημένο επίπεδο της αρχιτεκτονικής και μπορούν να εφαρμοστούν σε διαφορετικά στάδια της ανάπτυξης λογισμικού, όπως είναι π.χ. η προδιαγραφή απαιτήσεων, ο σχεδιασμός καθώς και η υλοποίηση και η δοκιμή του συστήματος [9]. Έτσι, δίνεται η δυνατότητα για μία αυστηρή, δικαιολογημένη ανάλυση των κρίσιμων ιδιοτήτων και η εξασφάλιση, της ορθότητας του συστήματος καθόλην την διάρκεια του κύκλου ζωής του λογισμικού [9,10].

3.2 Τυπικές Γλώσσες

Όπως αναφέρθηκε ήδη, οι τυπικές μέθοδοι χρησιμοποιούν μία τυπικά ορισμένη γλώσσα[2] της οποίας το συντακτικό , το λεξιλόγιο και η σημασιολογία της βασίζεται στα μαθηματικά.Το γεγονός ότι η τυπική γλώσσα βασίζεται στα μαθηματικά παρέχει μια πιο περιεκτική και σαφής περιγραφή του συστήματος καθώς και τα μέσα για την απόδειξη ότι το σύστημα είναι συνεπές και υλοποιήσιμο. Αυτό οφείλεται στο γεγονός ότι τα μαθηματικά παρέχουν σαφήνεια, ακρίβεια και προσφέρουν έναν τρόπο που μπορούν

[2] Συχνά,αναφέρεται και σαν γλώσσα τυπικής/επίσημης προδιαγραφής.

να δοθούν πειστικά επιχειρήματα για να δικαιολογηθούν οι λύσεις και η απόδειξη ότι η υλοποίηση είναι εφικτή [13].

3.2.1 Στοιχεία Τυπικής Γλώσσας

Σύμφωνα με τον ορισμό του Wing [12], μια τυπική γλώσσα αποτελείται από τα εξής στοιχεία:

- **Syn :** Σαν Syn ορίζεται το σύνολο του συντακτικού(syntax) της τυπικής γλώσσας. Το σύνολο αυτό αποτελείται από ένα σύνολο συμβόλων (γράμματα, σύμβολα πράξεων κλπ.) και ένα σύνολο γραμματικής για την δημιουργία τυπικών προτάσεων μέσα από τα σύμβολα
- **Sem :** Σαν Sem ορίζεται το σύνολο του σημασιολογικού(semantics) της γλώσσας. Το σύνολο αυτό, αποτελείται από έννοιες οι οποίες αποδίδουν νόημα ανάλογα με την κατηγορία της γλώσσας(αλγεβρικές γλώσσες,γλώσσες οριζόμενες από την διαδικασία).
- **Sat**: Σαν Sat ορίζεται η ικανοποίηση(Satisfaction) της σχέσης μεταξύ Sem και Syn. Μέσα από την ικανοποίηση δίνεται η συσχέτιση μεταξύ των στοιχείων που αποτελούν μία τυπική πρόταση.

Ορισμός κατα Wing : □*ί*□ □□*ί*□□□□ □□*ώ*□□□ < □□□, □□□, □□□ > □□□ □□ □□□(□□□, □□□) □*ό*□□ □□ □□□ □*ί*□□□ □□□ □□□□□□□□□□□□*ή* □□□ □□□ □□□ □□ □□□ *έ*□□ □□□□□□*ό* □□□ □□□ □□□ .

Παράδειγμα
Τυπική Γλώσσα: Προτασιακή λογική

Τυπικά καθορισμένη πρόταση: (p ∧ q) → r

Σύνταξη: ∧ αντιπροσωπεύει τον λογικό τελεστή AND, → αντιπροσωπεύει τον τελεστή λογικής συνεπαγωγής και τα p, q και r είναι προτασιακές μεταβλητές (δηλαδή, μεταβλητές Boolean που μπορούν να λάβουν τιμές είτε true είτε false).

Σημασιολογία: Αυτή η πρόταση ισχυρίζεται ότι αν και το p και το q είναι αληθή, τότε το r πρέπει επίσης να είναι αληθές.

Διευκρίνηση χρήσης της πρότασης

Έστω ότι υπάρχει ένα πρόγραμμα που πρέπει να ελέγξει εάν ένας πελάτης δικαιούται έκπτωση. Μπορούμε να χρησιμοποιήσουμε την πρόταση (p ∧ q) → r για να καθοριστούν τα κριτήρια επιλεξιμότητας. Σε αυτήν την περίπτωση, είναι το p αντιπροσωπεύει "ο πελάτης είναι μέλος", το q αντιπροσωπεύει "ο πελάτης έχει πραγματοποιήσει μια αγορά άνω των 100 ευρώ" και το r που αντιπροσωπεύει "ο πελάτης είναι επιλέξιμος για έκπτωση". Η πρόταση προσδιορίζει ότι εάν και τα δύο p και q είναι αληθή (δηλαδή, ο πελάτης είναι μέλος και έχει πραγματοποιήσει μια αγορά άνω των 100 ευρώ), τότε το r πρέπει επίσης να είναι αληθές (δηλαδή, ο πελάτης δικαιούται έκπτωση). Αυτή η πρόταση μπορεί να χρησιμοποιηθεί για να ελέγξει εάν ένας πελάτης πληροί τις προϋποθέσεις για έκπτωση με βάση την κατάσταση ιδιότητας μέλους και το ιστορικό αγορών του πελάτη.

3.2.2 Κατηγορίες Τυπικών Γλωσσών

Υπάρχουν διάφορες Τυπικές Γλώσσες με την καθεμία να έχει τα δικά της δυνατά και αδύνατα σημεία, και η επιλογή της κατάλληλης γλώσσας για μια συγκεκριμένη εφαρμογή είναι σημαντική. Ανάλογα με λοιπόν με τις ιδιότητες της γλώσσας προκύπτουν οι εξής κατηγορίες [13].

- **Γλώσσες με βάση το μοντέλο:** Οι γλώσσες που βασίζονται σε μοντέλα είναι μία κατηγορία από γλώσσες που εκφράζουν την προδιαγραφή σαν ένα μοντέλο με διαφορετικές καταστάσεις. Οι καταστάσεις του μοντέλου κατασκευάζονται χρησιμοποιώντας μαθηματικές έννοιες όπως σύνολα, σχέσεις, ακολουθίες και συναρτήσεις. Οι λειτουργίες του συστήματος που περιγράφει το μοντέλο, καθορίζονται ορίζοντας τον τρόπο που επηρεάζουν την κατάσταση του μοντέλου. Επίσης οι λειτουργίες περιγράφονται από τα κατηγορήματα που δίνονται ως προς τις προ και τις μεταγενέστερες συνθήκες του συστήματος.
- **Γλώσσες Αλγεβρικών Προδιαγραφών:** Οι γλώσσες αλγεβρικών προδιαγραφών περιγράφουν τη συμπεριφορά ενός συστήματος με βάση τα αξιώματα που χαρακτηρίζουν τις επιθυμητές ιδιότητές του συστήματος. Οι γλώσσες αυτές αποτελούνται κυρίως από ένα σύνολο συμβόλων που δηλώνουν τιμές κάποιου τύπου και ένα σύνολο λειτουργιών για την συσχέτιση των συμβόλων και των αλλαγών των τιμών.
- **Γλώσσες προσανατολισμένες στη διαδικασία:** Οι γλώσσες που είναι προσανατολισμένες στη διαδικασία βασίζονται σε διεργασίες που υποδηλώνονται και δημιουργούνται με εκφράσεις και στοιχειώδεις εκφράσεις.
- **Υβριδικές Γλώσσες:** Η υβριδικές γλώσσες χρησιμοποιούνται για να περιγράψουν συστήματα που έχουν αναλογικά και ψηφιακά συστατικά στοιχεία. Για την περιγραφή τέτοιων συστημάτων χρειάζεται μια γλώσσα που περιέχει στοιχεία τόσο από τα διακριτά όσο και τα συνεχή μαθηματικά.

3.3 Οι Πρακτικές Των Τυπικών Μεθόδων

Οι τυπικές μέθοδοι προσφέρουν μία πληθώρα από πρακτικές που μπορούν να χρησιμοποιηθούν για την προτυποποίηση και περιγραφή κάποιου συστήματος λογισμικού. Οι πρακτικές αυτές χρησιμοποιούνται σε συνδυασμό μεταξύ τους προσφέροντας μια καλύτερη εικόνα και εκτενέστερη ανάλυση του εκάστοτε συστήματος.

3.3.1 Τυπικές προδιαγραφές

Οι τυπικές προδιαγραφές (Formal Specifications), περιλαμβάνουν τον καθορισμό της συμπεριφοράς και των ιδιοτήτων του συστήματος χρησιμοποιώντας μια τυπική γλώσσα που έχει καλά καθορισμένη σύνταξη και σημασιολογία [9,10]. Οι ιδιότητες του συστήματος περιλαμβάνουν, την λειτουργική συμπεριφορά, την συμπεριφορά χρονισμού, και την εσωτερική δομή. Επίσης οι τυπικές προδιαγραφές μπορούν να συλλάβουν ανεπαίσθητες απαιτήσεις και περιορισμούς και μπορούν να αποφύγουν την ασάφεια και χρησιμοποιώντας την ακριβή σημασιολογία και σύνταξη που παρέχουν οι Τυπικές Γλώσσες [14].

3.3.2 Τυπική Επαλήθευση

Οι τυπική επαλήθευση (Formal Verification) είναι μία μέθοδος η οποία χρησιμοποιεί αυτοματοποιημένα εργαλεία για τον έλεγχο της ορθότητας της τυπικής προδιαγραφής όπου μέσα από τον έλεγχο αυτόν, μπορούν να εντοπιστούν σφάλματα, όπως αδιέξοδα(deadlocks), συνθήκες τερματισμού ή παραβιάσεις των ιδιοτήτων ασφαλείας [11]. Οι τεχνικές που χρησιμοποιούνται είναι τεχνικές ελέγχου μοντέλου (model checking) και τεχνικές απόδειξης θεωρήματος(Theorem proving) [14].

- **Έλεγχος Μοντέλου:** Ο έλεγχος μοντέλου βασίζεται στη δημιουργία ενός πεπερασμένου μοντέλο του συστήματος και στην συνέχεια ελέγχει το μοντέλο για την ύπαρξη μιας επιθυμητής ιδιότητας [14].
- **Απόδειξη Θεωρημάτων:** Η απόδειξη θεωρημάτων βασίζεται στην έκφραση των επιθυμητών ιδιοτήτων του συστήματος ως μαθηματικούς τύπους και τον ορισμό ενός συνόλου αξιωμάτων και ένα σύνολο κανόνων συμπερασμάτων [14].

3.3.3 Τυπικός Συλλογισμός

Ο τυπικός συλλογισμός (Formal Reasoning), είναι η διαδικασία εύρεσης καταστάσεων σε ένα μοντέλο που ικανοποιούν το σύνολο των ιδιοτήτων του μοντέλου και μέσα από αυτήν την διαδικασία, παρέχεται η δυνατότητα εξαγωγής συμπερασμάτων για την ασφάλεια και την ορθή λειτουργία του συστήματος σε όλες τις πιθανές του καταστάσεις [10,13]. Οι ιδιότητες που εξετάζονται με την μέθοδο αυτή, χωρίζονται σε δύο κατηγορίες:

- Η πρώτη κατηγορία είναι τα invariants [11] που είναι, οι ιδιότητες που προσδιορίζουν ότι τα χαρακτηριστικά του συστήματος θα παραμείνουν αμετάβλητα όσο και αν το μοντέλο αλλάζει καταστάσεις.
- Και η δεύτερη κατηγορία είναι οι ιδιότητες reaching [11] που διασφαλίζουν ότι κάποιες αλλαγές στο σύστημα θα συμβούν κάποια στιγμή μέσα σε ένα πεπερασμένο χρονικό διάστημα.

Συνοψίζοντας τα invariants παρέχουν την ασφάλεια ότι δεν θα γίνουν ανεπιθύμητες αλλαγές στο σύστημα και τα reaching ότι μόνον οι επιτρεπόμενες αλλαγές, θα γίνουν εξασφαλίζοντας την ορθότητα του συστήματος.

3.4 Πλεονεκτήματα Τυπικών Μεθόδων

Είναι γνωστό ότι ένα από τα μεγαλύτερα προβλήματα που αντιμετωπίζουν οι μηχανικοί λογισμικού, είναι η σωστή καταγραφή των απαιτήσεων που έχει το σύστημα που καλούνται να υλοποιήσουν. Η χρήση μιας απλής μη τυπικής γλώσσας όπως είναι τα Ελληνικά και τα Αγγλικά, δεν μπορούν να περιγράψουν ένα συστημα με σαφή και κατανοητό τρόπο όπως μπορεί μια τυπική γλώσσα. Μέσα από τις τυπικές μεθόδους ένας μηχανικός μπορεί να ορίσει με σαφήνεια τις απαιτήσεις ενός συστήματος χρησιμοποιώντας μια τυπική γλώσσα για την δημιουργία μιας τυπικής προδιαγραφής καί μέσα από τον τυπικό έλεγχο και τον τυπικό συλλογισμό μπορεί να εντοπίσει τα σφάλματα που ενδέχεται να προκύψουν αλλά και να επιβεβαιώσει ότι οι απαιτήσεις και οι ιδιότητες του συστήματος πληρούνται πριν καν φτάσει στο στάδιο της υλοποίησης κάτι δηλαδή που μέσα από τις παραδοσιακές μεθόδους δεν θα μπορούσε να είναι εφικτό. Σύμφωνα με την έρευνα των Batra κ.α [15] που δημοσιεύτηκε τον Μάιο του 2013, οι συγγραφείς εντόπισαν τα εξής πλεονεκτήματα:

1. **Μέτρο ορθότητας:** Η χρήση τυπικών μεθόδων παρέχει ένα μέτρο για τον έλεγχο της ορθότητας ενός συστήματος, σε αντίθεση με τα είδη υπάρχων μέτρα ελέγχου ποιότητας των συστημάτων.
2. **Έγκαιρη ανίχνευση ελαττωμάτων:** Με την χρήση τυπικών μεθόδων στα πρώιμα στάδια της σχεδίασης, ο εντοπισμός , η εύρεση και η διόρθωση σφαλμάτων γίνεται πιο γρήγορα και άμεσα.
3. **Εγγύηση ορθότητας:** Μέσα από την τυπική επαλήθευση ο ελεγκτής μοντέλου θα τρέξει το σύστημα με όλους τους πιθανούς τρόπους ώστε, να εντοπίσει όλα τα πιθανά σφάλματα. Προσφέροντας ένα ολοκληρωτικό επίπεδο κάλυψης.
4. **Επιρρεπής σε σφάλματα:** Οι τυπικές μέθοδοι έχουν την ιδιότητα της πληρότητας, δηλαδή μπορούν , να καλύψουν όλες τις πτυχές του συστήματος. Αυτό έχει ως αποτέλεσμα να βοηθά στη μείωση των σφαλμάτων που εμφανίζονται κατά τη διάρκεια ή μετά την συγγραφή του πηγαίου κώδικα καθώς στο στάδιο της σύνταξης και της καταγραφής των απαιτήσεων και των λειτουργιών της τυπικής προδιαγραφής ο μηχανικός χρειάζεται να αναλογιστεί και άλλες πτυχές του συστήματος πριν από την διαδικασία της υλοποίησης.

5. **Αφαιρετικότητα:** Μια τυπική προδιαγραφή, είναι μια περιγραφή που είναι αφηρημένη, ακριβής και σε υπό ορισμένες περιπτώσεις πλήρης. Η αφαίρεση επιτρέπει στον αναγνώστη της προδιαγραφής να κατανοήσει τη μεγάλη εικόνα του λογισμικού πιο εύκολα σε αντίθεση με ένα μεγάλο και περίπλοκο πρόγραμμα.
6. **Αυστηρή ανάλυση:** Η τυπικότητα της περιγραφής του συστήματος απο της τυπικές μεθόδους, επιτρέπει μια πιο αυστηρή ανάλυση του συστήματος και ο προσδιορισμός απαιτήσεων υψηλού επιπέδου ή της ορθότητα ενός προτεινόμενου σχεδίου γίνεται πιο εύκολα.
7. **Αξιοπιστία:** Οι τυπικές μέθοδοι παρέχουν το είδος των αποδεικτικών στοιχείων που απαιτούνται σε βιομηχανίες με αυστηρές διαρρυθμίσεις (π.χ έλεγχος συρμών τρένων). Και παρέχουν συγκεκριμένους λόγους για την επιλογή ενός τυπικά ορισμένου συστήματος.
8. **Αποτελεσματικές περιπτώσεις δοκιμών:** Μέσα από την τυπική προδιαγραφή, μπορούμε να αντλήσουμε συστηματικά αρκετές αποτελεσματικές περιπτώσεις δοκιμών απευθείας από την ίδια την προδιαγραφή. Καθώς είναι ένας οικονομικά αποδοτικός τρόπος δημιουργίας δοκιμαστικών περιπτώσεων.

Εργαλεία και μεθοδολογίες

Οι τυπικές μέθοδοι όπως αναφέρθηκε στο προηγούμενο κεφάλαιο είναι ένα σύνολο από πρακτικές που χρησιμοποιούνται στην μηχανική λογισμικού. Στο κεφάλαιο αυτό παρουσιάζονται δύο γλώσσες τυπικών προδιαγραφών που χρησιμοποιήθηκαν στην εν λόγω διπλωματική εργασία καθώς και ένα εργαλείο για την εξέταση των προδιαγραφών που υλοποιήθηκαν. Οι γλώσσες αυτές είναι η TLA+ και η PlusCal που υλοποιήθηκαν από τον Leslie Lamport και το εργαλείο που χρησιμοποιήθηκε αντίστοιχα για τον έλεγχο και την εξέταση των προδιαγραφών είναι το TLC model checker. Αυτό το κεφαλαίο περιέχει τις κατάλληλες πληροφορίες για την κατανόηση των ιδιοτήτων και τον χαρακτηριστικών των εργαλείων που χρησιμοποιήθηκαν. Στο τέλος του κεφαλαίου ο αναγνώστης θα είναι σε θέση να μπορεί να κατανοήσει τις προδιαγραφές που παρουσιάζονται στην εργασία αυτή.

4.1 Γλώσσα TLA+

Η γλώσσα TLA+ είναι μια γλώσσα τυπικών προδιαγραφών που δημιουργήθηκε από τον Leslie Lamport και χρησιμοποιείται για το σχεδιασμό, τη μοντελοποίηση, την τεκμηρίωση και την επαλήθευση προγραμμάτων, ιδιαίτερα των ταυτόχρονων και των κατανεμημένων συστημάτων [16]. Η γλώσσα έχει σκοπό την εύρεση ελαττωμάτων προτού ξεκινήσει η υλοποίηση του συστήματος και για την μοντελοποίηση ενός συστήματος η γλώσσα χρησιμοποιεί προδιαγραφές οι οποίες ορίζονται με μαθηματική λογική και αυτό δίνει την δυνατότητα εξέτασης του συστήματος σε βάθος κάνοντας έλεγχο πεπερασμένου μοντέλου, ώστε μέσα από τον έλεγχο αυτό, να μπορούν να εντοπιστούν όλες οι πιθανές συμπεριφορές του συστήματος μέχρι και κάποιον αριθμό βημάτων εκτέλεσης και να ελεγχθούν αν οι ιδιότητες αναλλοίωτης λειτουργίας(invariants) και ζωτικότητας(liveness) τηρούνται [17].

4.1.1 Ορολογίες TLA+

Για την κατανόηση των προδιαγραφών που θα παρουσιαστούν στην εν λόγω διπλωματική εργασία είναι σημαντικό να οριστούν κάποιες ορολογίες που χρησιμοποιούνται από την γλώσσα. Στον πίνακα που ακολουθεί παρέχεται μία επισκόπηση και ο ορισμός των πιο συνηθισμένων όρων που χρησιμοποιούνται από την γλώσσα TLA+ όπως εξηγούνται από το online εγχειρίδιο του Hillel Wayne LearnTLA [18]

Πίνακας 4.1 : Ορολογία TLA+

State	η εκχώρηση τιμών σε μεταβλητές
Behaviour	η ακολουθία των καταστάσεων(states)
Step	ένα ζεύγος διαδοχικών καταστάσεων σε μια συμπεριφορά
Stuttering step	ένα βήμα κατά το οποίο οι μεταβλητές δεν αλλάζουν
Next-state relation	μια σχέση που περιγράφει τον τρόπο με τον οποίο μπορούν να

	αλλάξουν οι μεταβλητές σε οποιοδήποτε βήμα
State function	μια παράσταση που περιέχει μεταβλητές και σταθερές που δεν αποτελεί σχέση επόμενης κατάστασης
State predicate	μια συνάρτηση κατάστασης με δυαδική τιμή
Invariant	ένα κατηγόρημα καταστάσεων που είναι αληθές σε όλες τις προσβάσιμες καταστάσεις
Temporal formula	μια έκφραση που περιέχει δηλώσεις σε χρονική λογική

4.1.2 Συντακτικό και Γραμματική

Η γλώσσα TLA+ όπως όλες οι γλώσσες έχει το δικό της συντακτικό και την δική της γραμματική. Το σύνολο του συντακτικού αποτελείται από σύμβολα που περιλαμβάνουν χειρισμό συνόλου όπως και τους τελεστές μέλους συνόλου, ένωση, διασταύρωση, διαφορά, σύνολο ισχύος και υποσύνολα. Περιλαμβάνονται επίσης λογικοί τελεστές πρώτης τάξης (∨, ∧, ¬, ⇒, ↔, ≡) καθώς και καθολικοί και υπαρξιακή ποσοτικοί δείκτες ∀ και ∃. Επίσης η γλώσσα παρέχει τελεστές χρονικής λογικής (temporal logic operators), την δυνατότητα στους χρήστες να μπορούν να ορίσουν τους δικούς τους τελεστές όπως και ειδικούς τελεστές που παρέχονται από τις βιβλιοθήκες της γλώσσας[3]. Για την αναπαράσταση των τελεστών , χρησιμοποιείται η αναπαράσταση ascii [17]. Στο παράδειγμα που ακολουθεί παρουσιάζεται η σύνταξη μιας έκφρασης σε TLA+.

Παράδειγμα

VARIABLE x

Increment == x' = x + 1

Σε αυτό το παράδειγμα, δηλώνεται πρώτα μια μεταβλητή x χρησιμοποιώντας τη λέξη-κλειδί VARIABLE. Στη συνέχεια ορίζεται η πράξη Increment χρησιμοποιώντας τον τελεστή ==. Η πράξη αυξάνει την τιμή του x κατά 1 και εκχωρεί το αποτέλεσμα στο x' (προφέρεται "x prime"), που είναι η τιμή του x στην επόμενη κατάσταση.

4.1.3 Δομές Δεδομένων

Η γλώσσα TLA+ παρέχει και τις δικές της δομές δεδομένων [17]. Οι δομές που παρέχει η γλώσσα είναι οι εξής.

1. **Sets (Σύνολα):** Τα σύνολα είναι η βασική δομή δεδομένων που χρησιμοποιεί η γλώσσα TLA+. Τα σύνολα, είτε απαριθμούνται ρητά είτε κατασκευάζονται από άλλα σύνολα χρησιμοποιώντας τελεστές.

[3] Στο παράρτημα 1 βρίσκεται ο πίνακας με τους τελεστές, τις βιβλιοθήκες και τις δεσμευμένες λέξεις της γλώσσας TLA+

2. **Functions (Συναρτήσεις):** Οι συναρτήσεις στην TLA+ αντιστοιχούν μια τιμή σε κάθε στοιχείο του πεδίου ορισμού τους.
3. **Records (Εγγραφές):** Οι εγγραφές είναι μία δομή δεδομένων που επιτρέπει την ομαδοποίηση σχετικών τιμών κάτω από ένα μόνο όνομα.
4. **Tuples (πλειάδες):** Μια πλειάδα είναι μια διατεταγμένη ακολουθία τιμών που είναι αμετάβλητη.

4.2 Γλώσσα PlusCal

Η PlusCal είναι μια γλώσσα τυπικών προδιαγραφών που δημιουργήθηκε από τον Leslie Lamport για την δημιουργία αλγορίθμων σε ταυτόχρονα και κατανεμημένα συστήματα [19]. Η γλώσσα αυτή έχει ως σκοπό να κάνει την δημιουργία αλγορίθμων πιο εύκολη και να αντικαταστήσει την χρήση ψευδογλώσσας παρέχοντας την απλότητα που έχει μια ψευδογλώσσα μέσα από μία πιο τυπικά καθορισμένη και επαληθεύσιμη γλώσσα [20]. Η γλώσσα PlusCal σε αντίθεση με μία παραδοσιακή γλώσσα προγραμματισμού δεν μεταφράζεται σε γλώσσα μηχανής αλλά σε προδιαγραφές TLA+ όπου οι προδιαγραφές αυτές ελέγχονται και επαληθεύονται από το εργαλείο TLC model checker και αυτό καθιστά την γλώσσα ιδανική για την περιγραφή και τον σχεδιασμό αλγορίθμων καθώς οι προδιαγραφές TLA+ έχουν σχεδιαστεί με σκοπό την απόδειξη των ιδιοτήτων ενός συστήματος οπότε οι ιδιότητες που περιγράφονται στον εκάστοτε αλγόριθμο μπορούν να επαληθευτούν και να αποδειχθούν μέσα από την μετάφραση τους σε TLA+ προδιαγραφές [19].

4.2.1 Χαρακτηριστικά PlusCal

Η γλώσσα PlusCal μοιράζεται αρκετά χαρακτηριστικά με την TLA+ καθώς οι εκφράσεις σε PlusCal είναι όλες οι εκφράσεις σε TLA+ που δεν χρησιμοποιούν κάποια δεσμευμένη λέξη PlusCal [19]. Παρόλα τα κοινά χαρακτηριστικά πού έχει η γλώσσα με την TLA+, η PlusCal έχει και κάποια δικά της που την διαφοροποιούν. Το πρώτο χαρακτηριστικό, είναι το διαφορετικό συντακτικό που έχει. Το συντακτικό της PlusCal πέρα από το συντακτικό της TLA+ εμπεριέχει και δικές της δηλώσεις όπως δηλώσεις επανάληψης (while) και δηλώσεις διακλάδωσης (If). Επίσης η PlusCal δίνει την δυνατότητα στον χρήστη για δημιουργία διεργασιών, διαδικασιών και μακροεντολών [19]. Τέλος το δεύτερο χαρακτηριστικό της PlusCal είναι ότι χρησιμοποιεί ένα σύνολο από κανόνες[4] για την συγγραφή των προδιαγραφών.

4.3 TLC Model Checker

Το τελευταίο εργαλείο που χρησιμοποιείται στην διπλωματική εργασία είναι το TLC model checker. Το εργαλείο αυτό είναι ένα εργαλείο που χρησιμοποιείται για την ανίχνευση σφαλμάτων στις

[4] Στο παράρτημα 1 βρίσκονται όλες οι δηλώσεις της γλώσσας καθώς και οι κανόνες που πρέπει να τηρούνται.

προδιαγραφές TLA+ και αναπτύχθηκε από τους Yuan Yu, Leslie Lamport, Mark Hayden, και Mark Tuttle[17]. Το TLC δημιουργεί ένα σύνολο από πεπερασμένες καταστάσεις και ελέγχει τη συνέπεια της προδιαγραφής, εντοπίζει σφάλματα και δημιουργεί αντιπαραδείγματα όταν εντοπίζει παραβίαση μιας ιδιότητας προδιαγραφής [21].

4.4 Χρήση Εργαλείων

Τα εργαλεία που παρουσιάστηκαν στο εν λόγω κεφάλαιο είναι τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση των τυπικών προδιαγραφών που ακολουθούν στο επόμενο κεφάλαιο. Πιο συγκεκριμένα για την υλοποίηση των δομών και των ιδιοτήτων invariants και temporal properties χρησιμοποιήθηκε η γλώσσα TLA+ και για την υλοποίηση των διαδικασιών και των διεργασιών χρησιμοποιήθηκε η γλώσσα PlusCal και για το κομμάτι του ελέγχου των προδιαγραφών χρησιμοποιήθηκε το εργαλείο TLC model checker. Επίσης χρησιμοποιήθηκαν οι βιβλιοθήκες:

- Integers
- TLC
- Sequences
- Finite Sets

Και στο αρχείο με την υλοποίηση της διαδικασίας δημιουργίας αρχείων χρησιμοποιήθηκε και η βιβλιοθήκη inode η οποία περιέχει όλες της δομές που χρησιμοποιεί το σύστημα αρχείων Ext2.

Υλοποίηση Διαδικασίας Δημιουργίας Αρχείου

Στην παρούσα διπλωματική εργασία, σαν περίπτωση μελέτης των τυπικών μεθόδων επιλέχθηκε η διαδικασία δημιουργίας αρχείων του συστήματος αρχείων ext2 όπως πραγματοποιείται στην πρώτη του έκδοση (ext2 fs revision 0). Με βάση την ανάλυση του Poirier [6] και τον πηγαίο κώδικα του ext2 [22], προκύπτει ότι η διαδικασία δημιουργίας αρχείων δημιουργεί τρεις βασικούς τύπους αρχείων:

- Regular file
- Directory file
- Symlink (symbolic link)

Όμως για των κάθε τύπο αρχείων, υπάρχουν κάποιες **διεργασίες** οι οποίες είναι υπεύθυνες για την δημιουργία των αρχείων που αυτές με την σειρά τους διαμορφώνουν **μικρότερες διαδικασίες** που εντάσσονται μέσα στην διαδικασία υλοποίησης αρχείων. Επίσης οι επιμέρους διαδικασίες αυτές, αποτελούνται από διάφορες δομές δεδομένων και ιδιότητες που πρέπει να τηρούνται. Για την κατανόηση και την προτυποποίηση της διαδικασίας δημιουργίας αρχείων είναι σημαντικό να κατανοηθούν οι επιμέρους διαδικασίες και τα υπόλοιπα στοιχεία που τις απαρτίζουν έτσι ώστε οι τυπικές προδιαγραφές που θα προκύψουν να είναι ορισμένες ορθά και με σαφήνεια. Μέσα στο κεφάλαιο αυτό, παρουσιάζεται η ανάλυση όλων των επιμέρους στοιχείων που απαρτίζουν την διαδικασία δημιουργίας αρχείων και στο τέλος παρουσιάζεται η σύνθεση των τυπικών προδιαγραφών.

5.1 Διαδικασία Δημιουργίας Αρχείων

Το σύστημά αρχείων όταν ο χρήστης θέλει να δημιουργήσει ένα συγκεκριμένο αρχείο από τους διαθέσιμους τύπους αρχείων (regular,directory.symlink) που προσφέρει το σύστημα αρχείων ext2, ξεκινά να εκτελεί μια σειρά από διεργασίες οι οποίες κατά την διάρκεια της εκτέλεσης τους, τηρούν ένα σύνολο από κανόνες και ιδιότητες. Το σύνολο όλων των στοιχείων αυτών, ονομάζεται διαδικασία δημιουργίας αρχείων. Για την προτυποποίηση της διαδικασίας αυτής χρησιμοποιώντας τις πρακτικές των τυπικών μεθόδων θα πρέπει αρχικά να εξεταστούν οι διεργασίες που εκτελούνται για την κάθε περίπτωση δημιουργίας αρχείου ξεχωριστά ώστε να υπάρχει μια σαφή και ξεκάθαρη εικόνα. Οπότε με βάση τους τύπους των αρχείων προκύπτουν οι εξής τρεις διαδικασίες:

1. Δημιουργία Regular File
2. Δημιουργία Directory File
3. Δημιουργία Symlink

5.1.1 Δημιουργία Regular File

Η πρώτη περίπτωση δημιουργίας αρχείου που θα εξεταστεί στην παράγραφο αυτή, είναι η διαδικασία δημιουργίας Regular File. Για την δημιουργία ενός regular file το σύστημα αρχείων εκτελεί τις εξής ενέργειες:

1. Το σύστημα αρχείων βρίσκει ελευθερό inode στο inode bitmap.
2. Το σύστημα αρχείων δεσμεύει το inode που βρήκε.
3. Το σύστημα αρχείων εισάγει τα μεταδεδομένα του αρχείου στο inode που δέσμευσε.
4. Το σύστημα αρχείων καταχωρεί στο directory linked list το νέο αρχείο.
5. Το σύστημα αρχείων ενημερώνει το block group descriptor table με τις αλλαγές που έγινε στο block group που δημιουργήθηκε το αρχείο.
6. Το σύστημα αρχείων ενημερώνει το superblock για της αλλαγές που προέκυψαν.

5.1.2 Δημιουργία Directory

Η δεύτερη κατά σειρά περίπτωση δημιουργίας αρχείου είναι η δημιουργία Directory. Η διαδικασία δημιουργίας directory αποτελείται από τις εξής διεργασίες:

1. Το σύστημα αρχείων βρίσκει ελεύθερο inode στο inode bitmap.
2. Το σύστημα αρχείων δεσμεύει ελεύθερο block απο το block bitmap
3. Το σύστημα αρχείων δεσμεύει το inode που βρήκε.
4. Το σύστημα αρχείων εισάγει τα μεταδεδομένα του αρχείου στο inode που δέσμευσε.
5. Το σύστημα αρχείων καταχωρεί στο directory linked list το νέο αρχείο.
6. Το σύστημα αρχείων ενημερώνει το block group descriptor table με τις αλλαγές που έγινε στο block group που δημιουργήθηκε το αρχείο.
7. Το σύστημα αρχείων ενημερώνει το superblock για της αλλαγές που προέκυψαν.

5.1.3 Δημιουργία Symlink

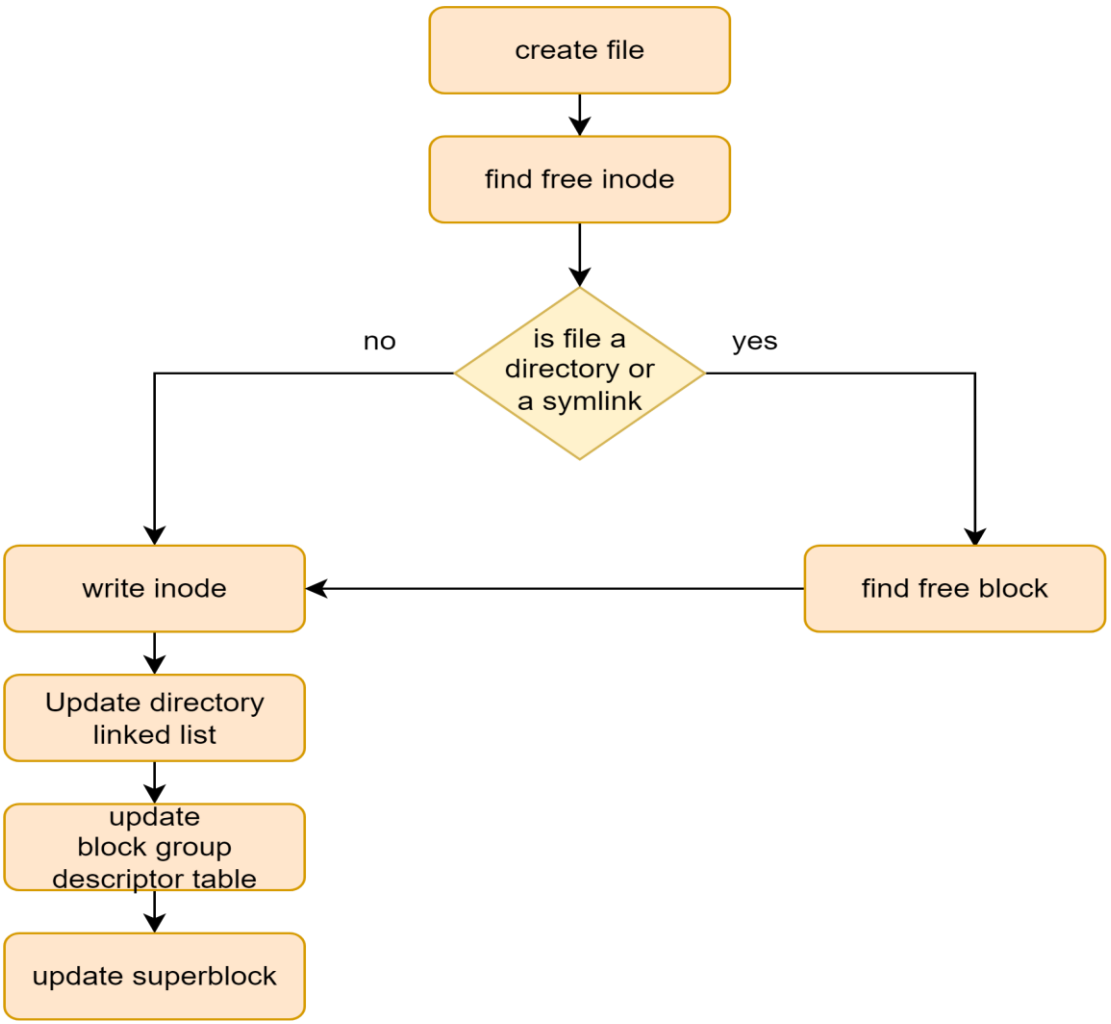
Η τελευταία κατά σειρά περίπτωση δημιουργίας αρχείου είναι η δημιουργία Symbolic link. Η διαδικασία δημιουργίας ενος symbolic link αποτελείται από τις εξής διεργασίες:

1. Το σύστημα αρχείων βρίσκει ελεύθερο inode στο inode bitmap.
2. Το σύστημα αρχείων δεσμεύει ελεύθερο block απο το block bitmap
3. Το σύστημα αρχείων δεσμεύει το inode που βρήκε.
4. Το σύστημα αρχείων εισάγει τα μεταδεδομένα του αρχείου στο inode που δέσμευσε.
5. Το σύστημα αρχείων καταχωρεί στο directory linked list το νέο αρχείο.
6. Το σύστημα αρχείων ενημερώνει το block group descriptor table με τις αλλαγές που έγινε στο block group που δημιουργήθηκε το αρχείο.
7. Το σύστημα αρχείων ενημερώνει το superblock για της αλλαγές που προέκυψαν.

5.1.4 Σύνοψη Και Περιγραφή Διαδικασίας Δημιουργίας Αρχείου

Εφόσον παρουσιάστηκαν οι περιπτώσεις των διαφόρων αρχείων που μπορούν να υλοποιηθούν από το σύστημα αρχείων ext2 η συνολική διαδικασία δημιουργίας αρχείων έχει ως εξής:

1. Όταν ο χρήστης δημιουργεί ένα αρχείο, το σύστημα αρχείων ψάχνει να βρει ελεύθερο inode από το inode bitmap.
2. **Αν** το αρχείο είναι symbolic link ή directory τότε το σύστημα αρχείων δεσμεύει ελεύθερο block απο το block bitmap **Αλλιώς** συνεχίζει κανονικά στην επόμενη διεργασία.
3. Το σύστημα αρχείων δεσμεύει το ελεύθερο inode και εισάγει τα μεταδεδομένα του αρχείου που δημιουργείται.
4. Το σύστημα αρχείων καταχωρεί στο directory linked list το νέο αρχείο.
5. Το σύστημα αρχείων ενημερώνει το block group descriptor table με τις αλλαγές που έγινε στο block group που δημιουργήθηκε το αρχείο.
6. Το σύστημα αρχείων ενημερώνει το superblock για της αλλαγές που προέκυψαν.



Εικόνα 5.1 : Διάγραμμα Ροής διαδικασίας αρχείων

Παρατηρώντας τα βήματα της δημιουργίας αρχείων προκύπτει η εξής παρατήρηση: Το σύστημα αρχείων **πρέπει να γνωρίζει** πότε ένα αρχείο είναι Regular file και πότε Symlink ή Directory έτσι ώστε το σύστημα να προβεί σε εύρεση ελεύθερου block. Για να επιτευχθεί αυτό θα πρέπει το σύστημα να ζητά από των χρήστη ένα αναγνωριστικό έτσι ώστε να ξεχωρίζει αν το αρχείο είναι Regular file,

Symlink ή Directory. Στην προκειμένη περίπτωση το αναγνωριστικό το οποίο θα πρέπει ο χρήστης να χρησιμοποιεί είναι μία από τις παρακάτω επιλογές:

- **touch**: Όταν ο χρήστης θέλει να φτιάξει ένα Regular file
- **mkdir**: Όταν ο χρήστης θέλει να φτιάξει Directory
- **slink**: Όταν ο χρήστης θέλει να φτιάξει Symlink

Επίσης με βάση όσα ειπώθηκαν στο 2.3.5 και με την εικόνα 2.6 όταν γίνεται μία νέα καταχώρηση στο directory linked list καταχωρείτε και το όνομα του αρχείου κάτι το οποίο **δίνεται από τον χρήστη** στο σύστημα αρχείων. Οπότε όταν ο χρήστης θέλει να δημιουργήσει ένα αρχείο πρέπει να καταχωρήσει και το όνομα του αρχείου μαζί με το αναγνωριστικό για τον τύπο αρχείου. Συνοψίζοντας για την δημιουργία του αρχείου το σύστημα αρχείων θα πρέπει **να ζητά αναγνωριστικό και όνομα αρχείου**. Εφόσον πλέον υπάρχει μία ξεκάθαρη εικόνα για την διαδικασία δημιουργίας αρχείου στην εικόνα[5] που ακολουθεί παρουσιάζεται η τυπική προδιαγραφή σε κώδικα PlusCal και η κλήση του συστήματος για δημιουργία αρχείου.

```
procedure CreateFile(command, name)
begin
  FindInode: call FindFreeInode();
  FindBlock :
   if command = "slink" then
         call FindFreeBlock()
    elsif command = "mkdir" then
          call FindFreeBlock()
   end if ;
   writeinode:  call UpdateInode(command)      ;
   update_BGDT: call UpdateBGDT(command);
   createdirentry:     call addDirEntry(name);
   updatesuperblock :     call updateSuperblock(command);
  goback:  return;
end procedure ;

process MAIN = "Main Process" begin
symbolic_link:
call CreateFile("slink", "symlink");
regularfile:
call CreateFile("touch", "regfile");
directory:
call CreateFile("mkdir", "directory1");
```

Εικόνα 5.2 : Τυπική προδιαγραφή δημιουργίας αρχείων και κλήσης συστήματος

5.2 Δομές

Για την προτυποποίηση της διαδικασίας δημιουργίας αρχείων δεν αρκεί μόνο η δημιουργία της ίδιας της διαδικασίας αλλά χρειάζεται να γίνει και προτυποποίηση των διεργασιών που εκτελούνται από την

[5] Στο παράρτημα 2 βρίσκεται ο κώδικας και σε μορφή εικόνας και σε μορφή ascii

ίδια την διαδικασία αλλά και η ίδιες οι δομές δεδομένων που χρησιμοποιούνται από το σύστημα αρχείων. Στο σημείο αυτό παρουσιάζονται αναλυτικά όλες οι δομές δεδομένων και οι τυπικές προδιαγραφές που τις περιγράφουν.

5.2.1 Δομή Inode

Η πρώτη δομή δεδομένων είναι η δομή inode. Όπως αναφέρθηκε και στο 2.3.4 το inode είναι μία δομή δεδομένων που χρησιμοποιεί το σύστημα αρχείων για να περιγράψει ένα αρχείο [5]. Οι πληροφορίες που έχει μέσα το inode είναι πληροφορίες που σχετίζονται με τα μεταδεδομένα του αρχείου που αντιπροσωπεύει [6]. Για την προτυποποίηση της δομής αυτής χρησιμοποιήθηκε η γλώσσα TLA+ και ορίστηκε ως Record. Για λόγους απλότητας στην προδιαγραφή χρησιμοποιήθηκαν μόνο τα πεδία του inode που επηρεάζονται κατά την διάρκεια της δημιουργίας ενός αρχείου. Αναλυτικότερα τα πεδία που χρησιμοποιήθηκαν είναι :

1. **iNum**: Το νούμερο του inode που σχετίζεται με το inode table και το inode bitmap.
2. **mode**: Ο τύπος του αρχείου
3. **uid**: Το αναγνωριστικό του χρήστη όπου ανήκει το αρχείο
4. **size**: Το μέγεθος του αρχείου
5. **atime**: Η τελευταία που φορά που έγινε προσπέλαση στο αρχείο
6. **ctime**: Η τελευταία φορά που έγινε κάποια αλλαγή στο αρχείο
7. **mtime**: Η τελευταία φορά που έγινε κάποια τροποποίηση στο αρχείο
8. **dtime**: Πότε έγινε η τελευταία διαγραφή του αρχείου
9. **gid**: Σε ποιο group χρηστών ανήκει το αρχείο
10. **links_count**: Το πλήθος των hardlinks που έχει το αρχείο
11. **blocks**: Το πλήθος το blocks που έχει το αρχείο
12. **flags**: Αναγνωριστικό flag που έχει το αρχείο
13. **osd1**: Από ποιο λειτουργικό σύστημα εξαρτάται το αρχείο
14. **directAddr**: Πίνακας με της διεύθυνσης των block που έχουν τα περιεχόμενα του αρχείου
15. **generation**: Έκδοση αρχείου
16. **file_acl**: Λίστα με τα δικαιώματα του αρχείου
17. **dir_acl**: Λίστα με τα δικαιώματα που έχει μέσα στον κατάλογο το αρχείο

$$
\begin{aligned}
Inode \triangleq\ & [\\
& iNum \mapsto 0, \quad Inode \text{ number is equal to its index in the array} \\
& mode \mapsto \text{“none”}, \quad \text{File type and permission bits} \\
& uid \mapsto 0, \quad \text{User } ID \text{ of the file's owner} \\
& size \mapsto 0, \quad \text{File size in bytes, initially set to 1024 bytes} \\
& atime \mapsto 0, \quad \text{Last time the file was accessed} \\
& ctime \mapsto 0, \quad \text{Last time the file was changed} \\
& mtime \mapsto 0, \quad \text{Last time the file was modified} \\
& dtime \mapsto 0, \quad \text{Last time the file was deleted} \\
& gid \mapsto 0, \quad \text{Group } ID \text{ of the file's owner} \\
& links\_count \mapsto 0, \quad \text{Number of hard links to the file} \\
& blocks \mapsto 0, \quad \text{Number of blocks used by the file} \\
& flags \mapsto 0, \quad \text{File flags} \\
& osd1 \mapsto 0, \quad \text{Operating system dependent field} \\
& directAddr \mapsto [b \in 0 \,..\, 11 \mapsto -1], \quad \text{Direct block addresses,} \\
& \text{initialized with } -1 \text{ indicating that it's not pointing to any block} \\
& generation \mapsto 0, \quad \text{File version (used for } NFS) \\
& file\_acl \mapsto -1, \quad \text{File access control list} \\
& dir\_acl \mapsto -1 \quad \text{Directory} \\
&]
\end{aligned}
$$

Εικόνα 5.3 : Τυπική προδιαγραφή Inode

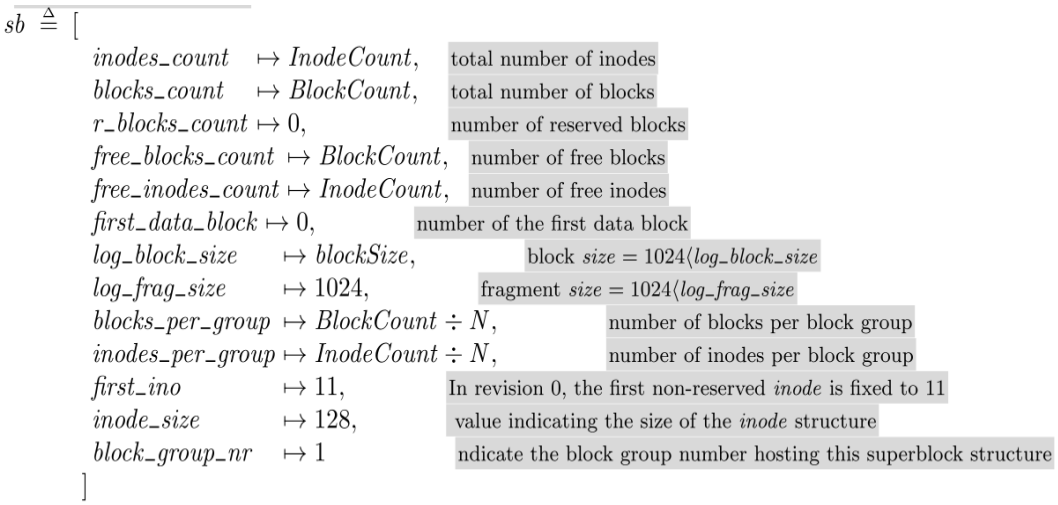
Όλα τα πεδία εκτός των **mode** και **directAddr** είναι ορισμένα σαν integers και αρχικοποιούνται με την τιμή 0 εκτός των file_acl και dir_acl που αρχικοποιούνται με την τιμή -1. Το πεδίο **mode** ορίζεται σαν string και αρχικοποιείται με την τιμή “none”. Τέλος το πεδίο **directAddr** είναι δηλωμένο σαν πίνακας. Οι πρόταση [□ ∈ *0..11* ↦ *−1*] υποδηλώνει ότι ο πίνακας **directAddr** για κάθε στοιχείο του πίνακα που ανήκει στο εύρος 0 εώς 11 το στοιχείο b του πίνακα έχει τιμή -1. Η τιμή αυτή δηλώνει ότι δεν χρησιμοποιείται κάποιο block από το αρχείο.

5.2.2 Δομή Superblock

Η δεύτερη κατά σειρά δομή είναι η δομή του superblock. Όπως και με το inode έτσι και εδώ η δομή που χρησιμοποιήθηκε για την δημιουργία της τυπικής προδιαγραφής είναι η δομή του Record και τα πεδία που χρησιμοποιήθηκαν είναι:

1. **inodes_count**: Το πλήθος των inode που υπάρχουν στο σύστημα αρχείων
2. **blocks_count**: Το πλήθος των block που υπάρχουν στο σύστημα αρχείων
3. **r_blocks_count**: Το πλήθος των δεσμευμένων block για τον super user
4. **free_blocks_count**: Το πλήθος των ελευθέρων block
5. **free_inodes_count**: Το πλήθος των ελευθέρων inode
6. **first_data_block**: Το νούμερο του πρώτου ελευθέρου block στο block που βρίσκεται το superblock

7. **log_block_size**: Το μέγεθος του block σε bytes
8. **log_frag_size**: Το μέγεθος των Fragments
9. **blocks_per_group**: Το πλήθος των blocks ανα block group
10. **inodes_per_group**: Το πλήθος των inodes ανα block group
11. **first_ino**: Το πρώτο ελεύθερο block στο σύστημα αρχείων
12. **inode_size**: Το μέγεθος του inode
13. **block_group_nr**: Το block group που φιλοξενεί το superblock



Εικόνα 5.4 : Τυπική προδιαγραφή Superblock

Κάθε πεδίο στην προδιαγραφή του **superblock** έχει οριστεί ως integer και αρχικοποιείται είτε με default τιμές, είτε με τις σταθερές που έχουν οριστεί από το σύστημα αρχείων όταν αυτό δημιουργήθηκε.

5.2.3 Δομή Block Group Descriptor Table

Η επόμενη δομή προς προτυποποίηση είναι η δομή του Block group descriptor table. Το block group descriptor table είναι ένας πίνακας που περιέχει όλες τις πληροφορίες για τα block groups που υπάρχουν στο σύστημα αρχείων, δηλαδή το block group descriptor table είναι ένας πίνακας που αποτελείται από block group descriptor entries[6]. Οπότε για την δημιουργία του block descriptor table χρειάζεται πρώτα να δημιουργηθεί η δομή block descriptor entry. Οι δομή έχει τα εξής πεδία:

1. **bg_block_bitmap**: Το πεδίο αυτό δείχνει στο πρώτο ελεύθερο data block στο block bitmap
2. **bg_inode_bitmap**: Το πεδίο αυτό δείχνει στο πρώτο ελεύθερο inode στο inode bitmap
3. **bg_inode_table**: Το πεδίο αυτό δείχνει στο πρώτο ελεύθερο inode στο inode table
4. **bg_free_block_count**: Το πεδίο αυτό δείχνει πλήθος των ελευθέρων block που ανήκουν στο block group
5. **bg_free_inode_count**: Το πεδίο αυτό δείχνει πλήθος των ελευθέρων inode που ανήκουν στο block group

6. **bg_used_dir_count**: Το πεδίο αυτό δείχνει το πλήθος των καταλόγων που χρησιμοποιούνται
7. **bg_pad**: Το πεδίο αυτό χρησιμοποιείται για το padding
8. **bg_reserved**: Το πεδίο αυτό χρησιμοποιείται για μελλοντική χρήση

$$
\begin{aligned}
BGD\_entry \triangleq\ & [\\
& \quad bg\_block\_bitmap \mapsto 0, && \text{the firts free data block in the block bitmap} \\
& \quad bg\_inode\_bitmap \mapsto 0, && \text{the first free } inode \text{ in the } inode \text{ bitmap} \\
& \quad bg\_inode\_table \mapsto 0, && \text{the first free } inode \text{ in the } inode \text{ table} \\
& \quad bg\_free\_block\_count \mapsto sb.blocks\_per\_group, && \text{the count of the free blocks} \\
& \quad bg\_free\_inode\_count \mapsto sb.inodes\_per\_group, && \text{the count of the free inodes} \\
& \quad bg\_used\_dir\_count \mapsto 1, && \text{the count of used directories} \\
& \quad bg\_pad \mapsto 0, && \text{used for padding the structure on a } 32bit \text{ boundary} \\
& \quad bg\_reserved \mapsto 10 && \text{reserved for future purpose} \\
&]
\end{aligned}
$$

$$BlockGroupDescriptorTable \triangleq [i \in 0 \,..\, N-1 \mapsto BGD\_entry]$$

Εικόνα 5.5 : Τυπική προδιαγραφή Block group descriptor entry και Block descriptor table

Όπως και στην περιγραφή του inode και του superblock τα πεδία του **BGD_entry** έχουν δηλωθεί σαν integers και αρχικοποιούνται με default τιμές. Τα πεδία **bg_free_block_count** και **bg_free_inode_count** αρχικοποιούνται με βάση την τιμή της δομής **superblock blocks_per_group** και **inodes_per_group**. Η δήλωση του **BlockGroupDescriptorTable** όπως και η πρόταση directAddr είναι δηλωμένη σαν πίνακας και υποδηλώνει ότι κάθε στοιχείο i στον πίνακα που ανήκει στο εύρος 0 εώς N-1 ειναι **BGD_Entry.**

5.2.4 Δομή Directory Entry

Η τελευταία δομή που χρειάζεται να προτυποποιηθεί είναι η δομή του directory entry. Το directory entry είναι η δομή που χρησιμοποιείται για την δημιουργία του directory linked list [6]. Τα πεδία του directory entry είναι τα εξής:

1. **inode**: Το πεδίο αυτό δείχνει το inode που σχετίζεται με το αρχείο
2. **rec_len**: Το πεδίο αυτό δείχνει το μήκος του αρχείου σε byte
3. **name_len**: Το πεδίο αυτό δείχνει το μήκος του ονόματος του αρχείου
4. **filename**: Το πεδίο αυτό αποθηκεύει το όνομα του αρχείου

$$
\begin{aligned}
dir\_entry \quad \triangleq\ & [\\
& \quad inode \mapsto 0, \quad inode \text{ number associated with the file} \\
& \quad rec\_len \mapsto 0, \quad \text{length of the record} \\
& \quad name\_len \mapsto 0, \quad \text{length of filename} \\
& \quad filename \ \mapsto \text{“”} \quad \text{name of the file} \\
&]
\end{aligned}
$$

Εικόνα 5.6 : Τυπική προδιαγραφή directory linked list

Όλα τα πεδία στο **dir_entry** είναι δηλωμενα σαν integer εκτός από το **filename** που είναι string

5.3 Διεργασίες

Εφόσον παρουσιάστηκαν οι δομές δεδομένων που χρειάζονται από την διαδικασία δημιουργίας αρχείων στο σημείο αυτό θα παρουσιαστούν οι διεργασίες που εκτελούνται για την δημιουργία ενός αρχείου. Οι διεργασίες αυτές είναι οι εξής:

1. Διεργασία Εύρεσης Inode
2. Διεργασία Εύρεσης Block
3. Διεργασία Δέσμευσης Inode
4. Διεργασία Ενημέρωσης BGDT
5. Διεργασία Ενημέρωσης Directory Linked List
6. Διεργασία Ενημέρωσης SuperBlock

5.3.1 Διεργασία Εύρεσής Inode

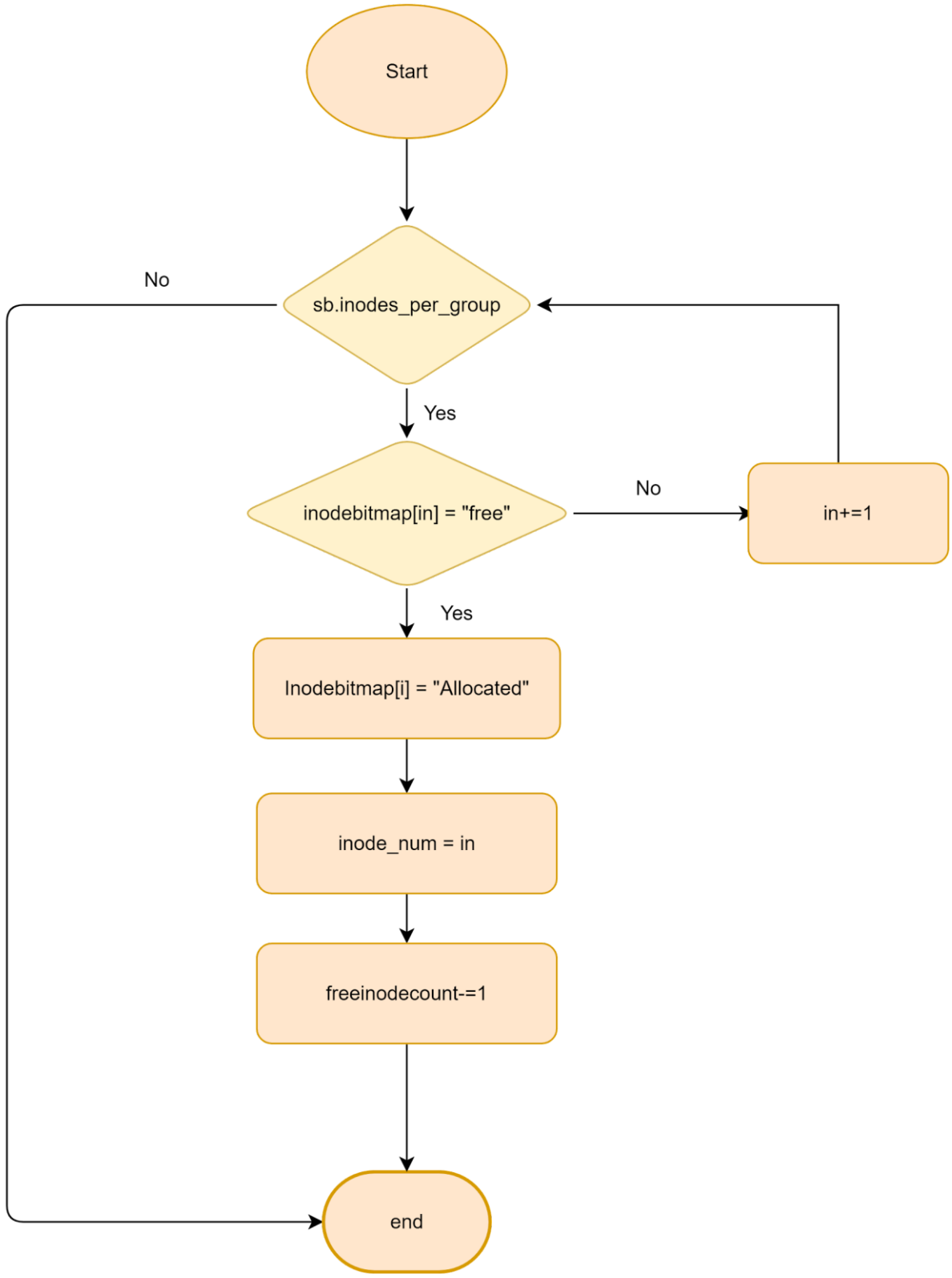
Η διεργασία εύρεσής inode έχει ως εξής:

“**Όσο υπάρχουν inode στο inode bitmap κάνε το εξής . Αν το inode είναι ελεύθερο τότε άλλαξε το σε κατάσταση δεσμευμένου. Στη συνέχεια μείωσε κατά ένα το πλήθος των ελεύθερων inode και επέστρεψε το inode που βρήκες. Αλλιώς, προχώρα στο επόμενο κατά σειρά inode**.”

Από την **μη τυπική** περιγραφή που μόλις παρουσιάστηκε γίνεται αντιληπτό ότι η διεργασία εύρεσης inode είναι μία επαναληπτική ενέργεια που γίνεται μέχρι να βρεθεί ένα ελεύθερο inode καθώς και ότι χρειάζεται να οριστούν κάποιες μεταβλητές ώστε να μπορεί να πραγματοποιηθεί η διεργασία. Οπότε με βάση λοιπόν την πληροφορία αυτήν οι μεταβλητές που πρέπει να οριστούν είναι:

1. **inodeBitmap**: Πίνακας με μέγεθος το συνολικό πλήθος inode του block group που ανήκει. Κάθε στοιχείο του πίνακα παίρνει την τιμή “Free” αν το inode είναι ελεύθερο και την τιμή “Allocated” αν είναι δεσμευμένο
2. **in**: Μεταβλητή που χρησιμοποιείται για την διάσχιση του πίνακα inodeBitmap
3. **inode_num**: Μεταβλητή που χρησιμοποιείται για την αποθήκευση του ελεύθερου inode
4. **freeinodecount**: Το σύνολο των ελεύθερων inode στο block group

Λαμβάνοντας υπόψη τις μεταβλητές που αναφέρθηκαν οι διεργασία εύρεσης ελεύθερου inode παρουσιάζεται στο διάγραμμα ροής που ακολουθεί(Εικόνα 5.7).



Εικόνα 5.7 : Διάγραμμα ροής διεργασίας ευρέσεως ελεύθερου inode

Έχοντας πλέον μία ξεκάθαρη εικόνα για την διεργασία εύρεσης ελεύθερου inode στους πίνακες που ακολουθούν παρουσιάζονται ο ορισμός και η δήλωση των μεταβλητών σε γλώσσα PlusCal και στη συνέχεια, η τυπική προδιαγραφή της διεργασίας .

Πίνακας 5.1 : Δηλώσεις μεταβλητών FindFreeInode

Μεταβλητή	**Ορισμος PlusCal**
inodeBitmap	□□□□□□□□□□□□ = [□ ∈ 0..□□.□□□□□□_□□□_□□□□□ − 1 ↦ "□□□□"]
freeInodeCount	□□□□□□□□□□□□□□□ = {□ ∈ 0..□□.□□□□□□_□□□_□□□□□ − 1 : □□□□□□□□□□□□[□] = "□□□□"}
in	in = 0
inode_num	inode_num = -1

Πίνακας 5.2 : Επεξηγήσεις Μεταβλητών

Μεταβλητή	**Επεξήγηση δήλωσης**
inodeBitmap	Η πρόταση αυτή υποδηλώνει ότι κάθε στοιχείο i του πίνακα που ανήκει στο εύρος 0 εώς sb.inodes_per_group έχει τιμή “FREE”
freeInodeCount	Η πρόταση αυτή υποδηλώνει ότι το freeInodeCount είναι το σύνολο όλων των στοιχείων του πίνακα inodeBitmap που το inodeBitmap[i] = “FREE” και το i παίρνει τιμές από 0 εώς sb.inodes_per_group
in	Μεταβλητή integer που χρησιμοποιείται για την διάσχιση του πίνακα inodeBitmap
inode_num	Μεταβλητή τύπου integer που αρχικοποιείται με την τιμή -1. Σε περίπτωση που η μεταβλητή δεν αλλάξει τιμή σημαίνει ότι δεν υπάρχει ελεύθερο inode στο σύστημα αρχείων.

```
procedure FindFreeInode()begin
 find:
  while in < sb.inodes_per_group do
    if (inodeBitmap[in] = "FREE") then
        inodeBitmap[in] := "ALLOCATED" ;
        inode_num := in ;
        FreeInodeCount := FreeInodeCount \ {in}
        goto final ;
     else
         in := in + 1 ;
    end if ;
  end while ;
final:   return ;
end procedure ;
```

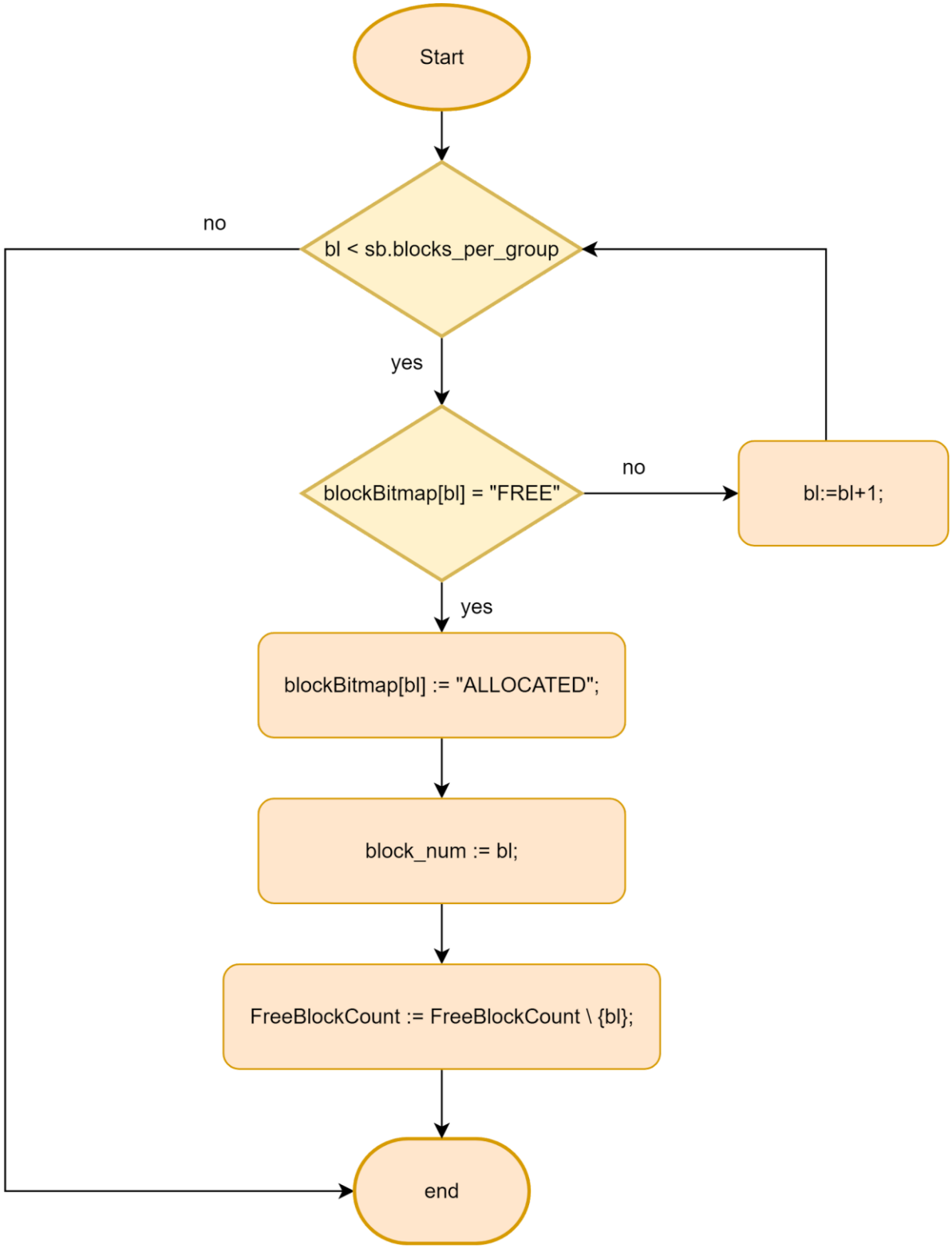
Εικόνα 5.8 : Τυπική προδιαγραφή διεργασίας εύρεσης ελεύθερου inode

5.3.2 Διεργασία Εύρεσης Block

Η διεργασία εύρεσης block έχει ως εξής:

" **Όσο υπάρχουν blocks στο block bitmap κάνε το εξής . Αν το block είναι ελεύθερο τότε άλλαξε το σε κατάσταση δεσμευμένου. Στην συνέχεια μείωσε κατά ένα το πλήθος των ελεύθερων block και επέστρεψε σε το block που βρήκες. Αλλιώς , προχώρα στο επόμενο κατά σειρά block**."

Από την **μη τυπική** περιγραφή που μόλις παρουσιάστηκε γίνεται αντιληπτό ότι η διεργασία εύρεσης block είναι παρόμοια με την διεργασία εύρεσης inode οπότε για την υλοποίηση της τυπικής προδιαγραφής θα χρησιμοποιηθούν παρόμοιες μεταβλητές με αυτές που χρησιμοποιήθηκαν και στην διεργασία εύρεσης inode.



Εικόνα 5.9 : Διάγραμμα ροής διεργασίας εύρεσης ελεύθερου block

Ακολουθώντας λοιπόν την ίδια λογική με την διεργασία εύρεσης inode στους πίνακες που ακολουθούν παρουσιάζονται οι δηλώσεις και οι επεξηγήσεις των μεταβλητών της διεργασίας ευρέσεως block.

Πίνακας 5.3 : Δηλώσεις μεταβλητών FindFreeBlock

Μεταβλητή	Ορισμος PlusCal
blockBitmap	□□□□□□□□□□□□ = [□ ∈ 0..□□.□□□□□□_□□□_□□□□□ − 1 ↦ "□□□□"]
freeBlockCount	□□□□□□□□□□□□□□□ = {□ ∈ 0..□□.□□□□□□_□□□_□□□□□ − 1 : □□□□□□□□□□□□[□] = "□□□□"}
bl	bl = 0
block_num	block_num = -1

Πίνακας 5.4 : Επεξηγήσεις Μεταβλητών

Μεταβλητή	Επεξήγηση δήλωσης
blockBitmap	Η πρόταση αυτή υποδηλώνει ότι κάθε στοιχείο i του πίνακα που ανήκει στο εύρος 0 εώς sb.blocks_per_group έχει τιμή "FREE"
freeBlockCount	Η πρόταση αυτή υποδηλώνει ότι το freeBlockCount είναι το σύνολο όλων των στοιχείων του πίνακα blockBitmap που το blockBitmap[i] = "FREE" και το i παίρνει τιμές από 0 εώς sb.blocks_per_group
bl	Μεταβλητή integer που χρησιμοποιείται για την διάσχιση του πίνακα blockBitmap
block_num	Μεταβλητή τύπου integer που αρχικοποιείται με την τιμή -1. Σε περίπτωση που η μεταβλητή δεν αλλάξει τιμή σημαίνει ότι δεν υπάρχει ελεύθερο block στο σύστημα αρχείων.

```
procedure FindFreeBlock()begin
 find:
  while bl < sb.blocks_per_group do
    if (blockBitmap[bl] = "FREE") then
        blockBitmap[bl] := "ALLOCATED" ;
        block_num := bl ;
        FreeBlockCount := FreeBlockCount \ {bl} ;
        goto final ;
     else
         bl := bl + 1 ;
    end if ;
  end while ;
final:   return ;
end procedure ;
```

Εικόνα 5.10 : Τυπική προδιαγραφή διεργασίας εύρεσης ελεύθερου block

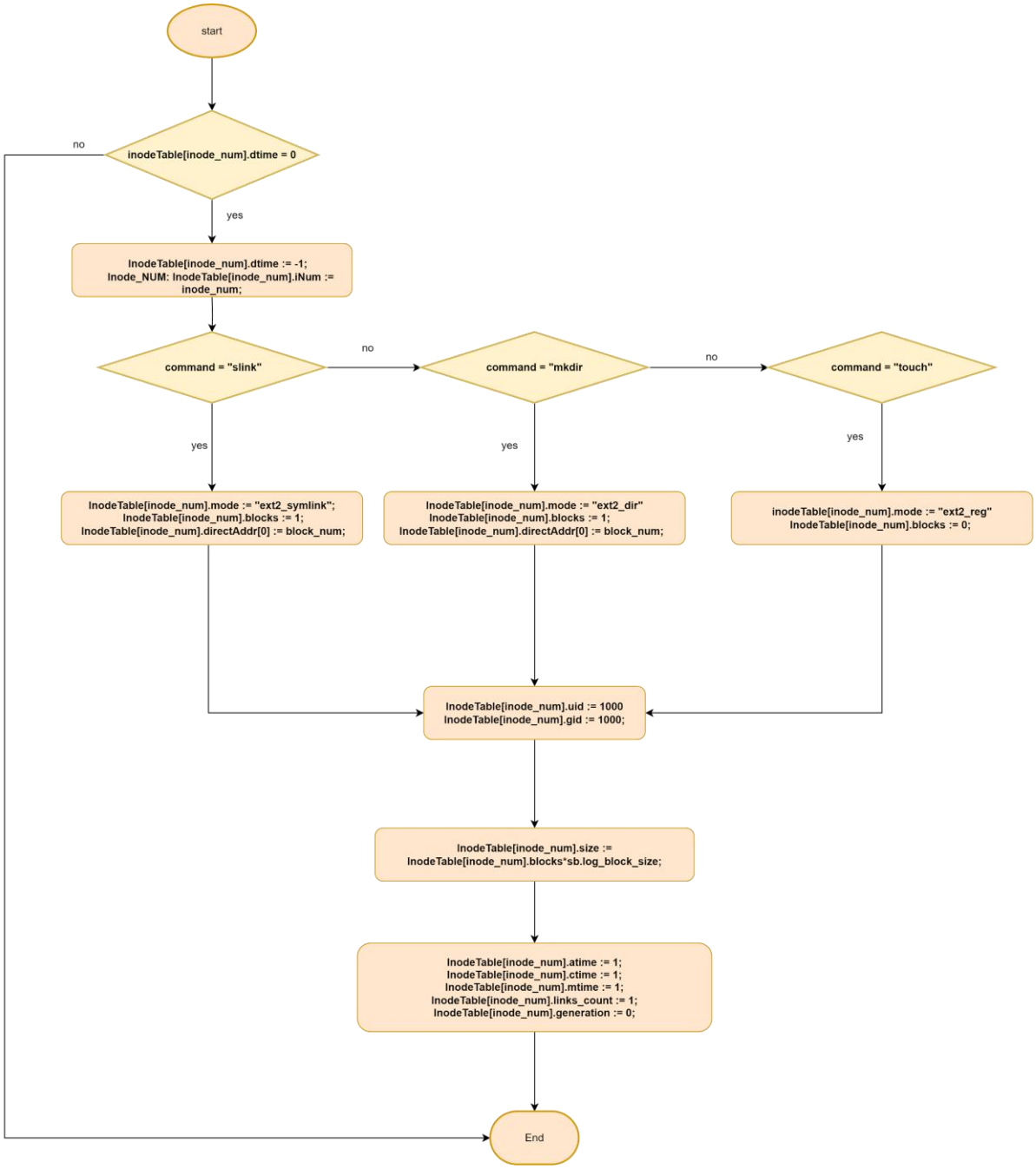
5.3.3 Διεργασία Δέσμευσης Inode

Η διεργασία δέσμευσής inode έχει ως εξής.

"Αν το inode έχει στο πεδίο dtime την τιμή 0 τότε κάνε τις εξής αλλαγές.

- **Άλλαξε το πεδίο dtime απο 0 σε -1**
- **Ενημέρωσε το πεδίο iNum με την τιμή του inode_num που βρήκες από το inodeBitmap**
- **Ανάλογα με τον τύπο αρχείου που δημιουργείται ενημέρωσε το πεδίο mode είτε σε ext2_sylink αν το αρχείο είναι symlink είτε σε ext2_dir αν είναι directory ή σε ext2_reg αν είναι regular file**
- **Αν το αρχείο είναι symlink ή directory ανέβασε κατα 1 το πεδίο blocks**
- **Αν το αρχείο είναι symlink ή directory ενημέρωσε την πρώτη θέση του πίνακα dirrectAddr με την τιμή του block_num που βρήκες απο το blockBitmap**
- **Ενημέρωσε τα πεδία guid και uid με τα κατάλληλα αναγνωριστικά**
- **Ενημέρωσε το πεδίο size με το νέο μέγεθος του αρχείου**
- **Αρχικοποίησε τα πεδία atime,ctime,mtime και linkscount με την τιμή 1**
- **Αρχικοποίησε το πεδίο generation με την τιμή 0**

Αν το πεδίο dtime δεν είναι 0 τότε το inode είναι δεσμευμένο και μην προχωρήσεις σε καμία αλλαγή". Παρατηρώντας την μη τυπική περιγραφή της διεργασίας δέσμευσης inode παρατηρείται ότι σύστημα αρχείων πρέπει να γνωρίζει τι αρχείο δημιουργείται όχι μόνο για την εύρεση του block αλλά και για την σωστή ενημέρωση των πεδίων του inode. Οπότε για την τυπική προδιαγραφή θα πρέπει σαν είσοδο η διεργασία δέσμευσης inode να δέχεται το αντίστοιχο αναγνωριστικό για τον τύπο αρχείου.



Εικόνα 5.11 : Διάγραμμα ροής διεργασία δέσμευσης inode

```
procedure UpdateInode(command)
begin
writeInode:
    if InodeTable[inode_num].dtime = 0 then
      InodeTable[inode_num].dtime := −1;
      Inode_NUM: InodeTable[inode_num].iNum := inode_num;
      change_mode:
      if command = "slink" then
          InodeTable[inode_num].mode := "ext2_symlink";
       elsif command = "mkdir" then
          InodeTable[inode_num].mode := "ext2_dir";
       elsif command = "touch" then
          InodeTable[inode_num].mode := "ext2_reg";
       end if;
      init_block:
      if command = "slink" then
           InodeTable[inode_num].blocks := 1;
       elsif command = "mkdir" then
           InodeTable[inode_num].blocks := 1;
       elsif command = "touch" then
           InodeTable[inode_num].blocks := 0;
      end if;
      fstblock:
      if command = "slink" then
            InodeTable[inode_num].directAddr[0] := block_num;
       elsif command = "mkdir" then
            InodeTable[inode_num].directAddr[0] := block_num;
      end if;
      change_uid: InodeTable[inode_num].uid := 1000;
      change_gid: InodeTable[inode_num].gid := 1000;
      change_size: InodeTable[inode_num].size := InodeTable[inode_num].blocks * sb.log_block_size;
      change_access_time: InodeTable[inode_num].atime := 1;
      change_creation_time: InodeTable[inode_num].ctime := 1;
      change_modification_time: InodeTable[inode_num].mtime := 1;
      change_links_count: InodeTable[inode_num].links_count := 1;
      generation_type: InodeTable[inode_num].generation := 0;
       else
      print "Error: inode already in use";
    end if;
goback: return;
```

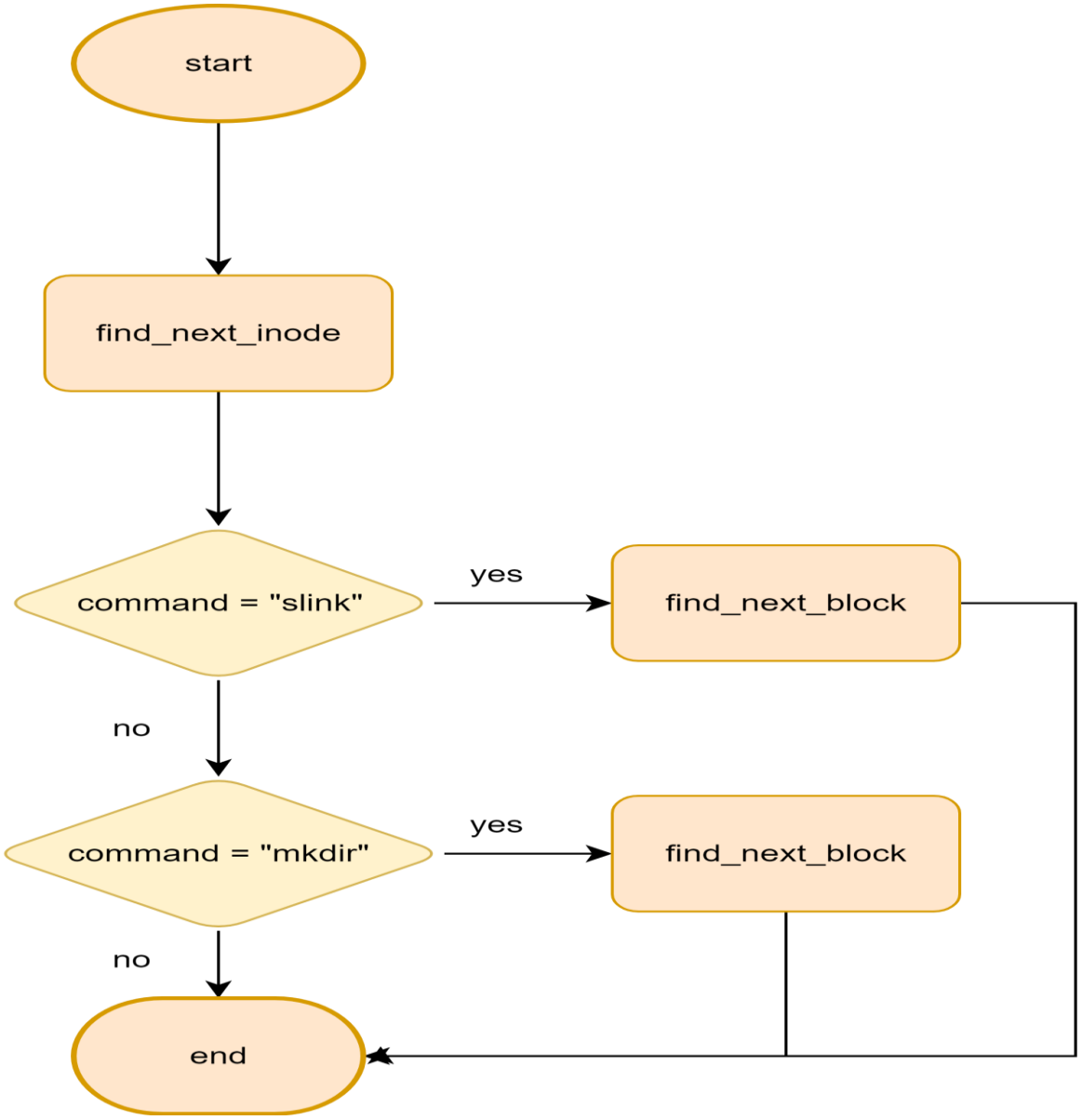
Εικόνα 5.12 : Τυπική προδιαγραφή διεργασίας δέσμευσης inode

5.3.4 Διεργασία Ενημέρωσης BGDT

Η διεργασία ενημέρωσης BGDT (block group descriptor table) έχει ως εξής.

"Όταν το αρχείο έχει δημιουργηθεί πήγαινε στο block group descriptor table και βρες το επόμενο ελεύθερο inode. Εφόσον το βρεις μείωσε κατά 1 το πεδίο bg_free_inode_count και ενημέρωσε τα πεδία bg_inode_table και nbg inode bitmap με το νούμερο του καινούργιου inode. Αν το αρχείο που δημιουργήθηκε είναι symlink ή directory μείωσε κατά 1 το πεδίο bg_free_block_count και βρες το επόμενο ελεύθερο block. Όταν το βρεις ενημέρωσε το πεδίο bg block bitmap ". Με μία πρώτη ματιά παρατηρείται ότι η διεργασία ενημέρωσης BGDT αποτελείται από δύο ακόμα διεργασίας. Μία διεργασία που βρίσκει το επόμενο ελεύθερο inode και το επόμενο ελεύθερο block. Επίσης όπως

και η διεργασία δέσμευσης inode έτσι και η διεργασία αυτή, χρειάζεται να λάβει σαν είσοδο το αναγνωριστικό δημιουργίας αρχείου.



Εικόνα 5.13 : Διάγραμμα ροής διεργασία ενημέρωσης BGDT

```
procedure UpdateBGDT(command)
begin
find_next_free_inode : call find_next_free_inode() ;
find_next_free_block :
if command = "slink" then
            call find_next_free_block()
    elsif command = "mkdir" then
            call find_next_free_block()
   end if ;
final:
return ;
end procedure ;
```

Εικόνα 5.14 : Τυπική προδιαγραφή διεργασίας ενημέρωσης BGDT

Οι διεργασίες **find_next_free_inode** και **find_next_free_block**, είναι πανομοιότυπες με τις διεργασίες FindFreeInode και FindFreeBlock που παρουσιάστηκαν προηγουμένως. Οι διαφορές που έχουν είναι ότι **δεν** αλλάζουν το πλήθος των ελευθερών block και inode αντίστοιχα και ενημερώνουν τα κατάλληλα πεδία του block group descriptor table. Οπότε με βάση αυτές της αλλαγές οι τυπικές προδιαγραφές των διεργασιών αυτών είναι οι ακόλουθες.

```
procedure find_next_free_block()
begin
blvalue: bl := BGDT[0].bg_block_bitmap ;
find_next__free_block:
  while bl < sb.blocks_per_group do
    if (blockBitmap[bl] = "FREE") then
        BGDT[0].bg_block_bitmap := bl ;
        bl := sb.blocks_per_group ;
     else
         bl := bl + 1 ;
    end if ;
  end while ;
change_free_blockcount:
BGDT[0].bg_free_block_count := BGDT[0].bg_free_block_count − 1 ;
blnewvalue: bl := BGDT[0].bg_block_bitmap ;
final:
return ;
end procedure ;
```

Εικόνα 5.15 : Τυπική προδιαγραφή διεργασίας εύρεσης επόμενου ελεύθερου block

```
procedure find_next_free_inode()
 begin
 valueupdate:
 in := BGDT[0].bg_inode_bitmap ;
 find_next__free_inode:
  while in < sb.inodes_per_group do
    if (inodeBitmap[in] = "FREE") then
        BGDT[0].bg_inode_bitmap := in ;
        change_inodetablevalue:
        BGDT[0].bg_inode_table := in ;
        in := sb.inodes_per_group ;
     else
         in := in + 1 ;
    end if ;
  end while ;
change_free_inode_count:
BGDT[0].bg_free_inode_count := BGDT[0].bg_free_inode_count − 1 ;
rstval: in := BGDT[0].bg_inode_bitmap ;
final:
return ;
end procedure ;
```

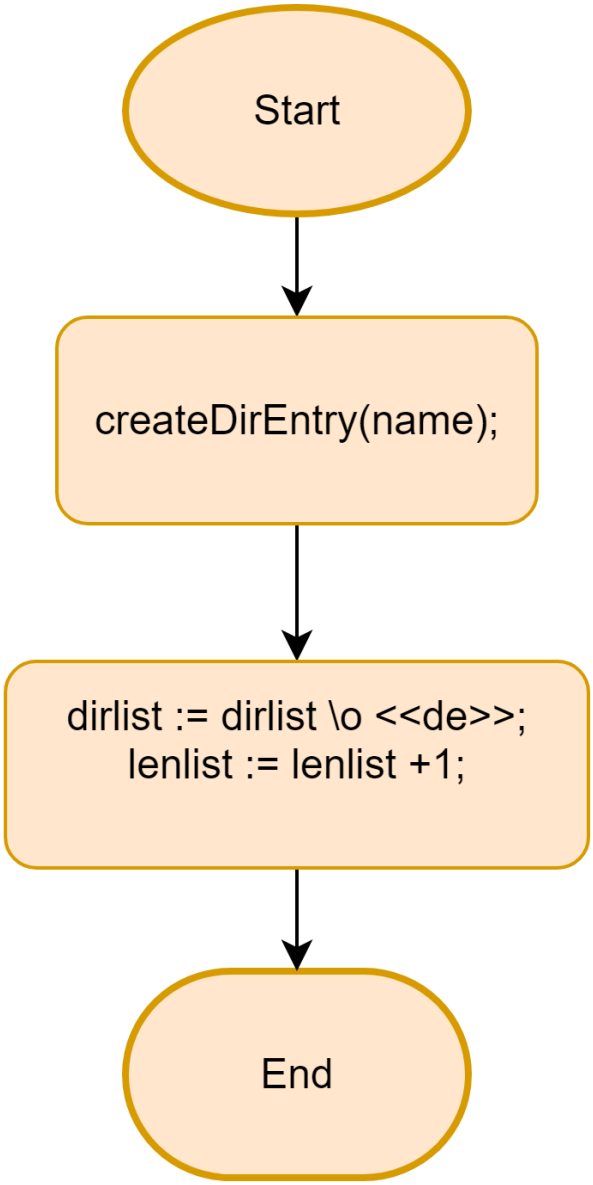
Εικόνα 5.16 : Τυπική προδιαγραφή διεργασίας εύρεσης επόμενου ελεύθερου inode

Οι μεταβλητές **rstval** και **bl newvalue** χρησιμοποιούνται για την συνέπεια και την διατήρηση της διάσχισης των πινάκων.

5.3.5 Διεργασία Ενημέρωσης Directory Linked List

Η διεργασία ενημέρωσης Directory Linked List έχει ως εξής.

"Δημιούργησε μία καινούργια καταχώρηση. Όταν την φτιάξεις τοποθέτησε την στην λίστα. Στη συνέχεια ενημέρωσε το μέγεθος της λίστας". Από την μη τυπική περιγραφή της διεργασίας ενημέρωσης directory linked list, προκύπτει ότι για την εκτέλεση της θα πρέπει πρώτα το σύστημα αρχείων να εκτελέσει μία νέα διεργασία. Η διεργασία αυτή, είναι υπεύθυνη για τη δημιουργία directory entry.



Εικόνα 5.17 : Διάγραμμα ροής διεργασία ενημέρωσης Directory Linked List

Με βάση το διάγραμμα ροής της διεργασίας ενημέρωσης Directory Linked List (Εικόνα 5.17) χρειάζεται να λάβει σαν είσοδο το όνομα του αρχείου που δημιουργείται. Επίσης χρειάζεται να δηλωθούν δύο νέες μεταβλητές. Η πρώτη μεταβλητή είναι μία μεταβλητή που αποθηκεύει το μέγεθος της λίστας και η δεύτερη είναι η μεταβλητή που αντιπροσωπεύει την λίστα.

Πίνακας 5.5 : Δηλώσεις μεταβλητών add DirEntry

Μεταβλητή	Ορισμος PlusCal
dirlist	dirlist = << >>
len list	len list = 0

```
procedure createDirEntry(name)begin
create:
    de.inode := inode_num ;
nanmelen:
    de.name_len := Len(name) ;
reclen:
    de.rec_len := ((8 + de.name_len + 3) * 4) ÷ 4 ;
filename:
    de.filename := name ;
back:
return ;
end procedure ;
```

Εικόνα 5.17 : Τυπική προδιαγραφή διεργασίας ενημέρωσης Directory Linked List

Η διεργασία δημιουργιας directory entry από το διάγραμμα ροής της διεργασίας ενημέρωσης Directory Linked List για την εκτέλεσή της πρέπει να δεχθεί και αυτή σαν είσοδο το όνομα του αρχείου που δημιουργήθηκε. Οπότε η τυπική προδιαγραφή της διεργασίας παρουσιάζεται στην εικόνα 5.18.

```
dir_entry  ≜ [
               inode ↦ 0,    inode number associated with the file
               rec_len ↦ 0,  length of the record
               name_len ↦ 0, length of filename
               filename  ↦ "" name of the file
             ]
```

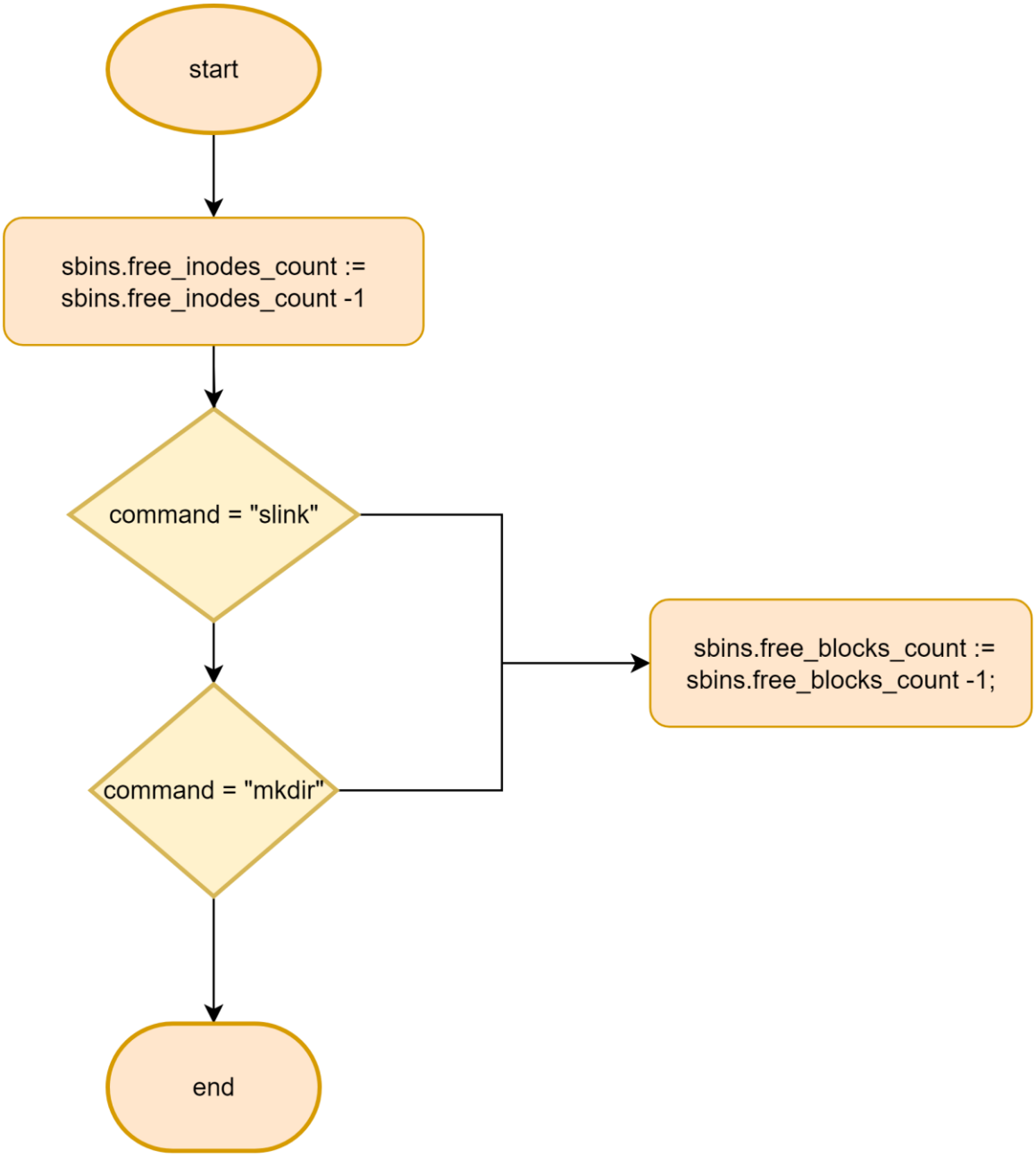
Εικόνα 5.18 : Τυπική προδιαγραφή διεργασίας δημιουργιας directory entry

5.3.6 Διεργασία Ενημέρωσης SuperBlock

Η διεργασία ενημέρωσης superblock έχει ως εξής

"Αφαίρεσε από το συνολικό πλήθος των ελεύθερων inode που υπάρχουν στο σύστημα το inode που μόλις δέσμευσες. Αν το αρχείο που δημιουργήθηκε είναι symlink ή directory αφαίρεσε το block που δέσμευσες από το πλήθος των ελεύθερων block που υπάρχουν στο σύστημα ". Από την

περιγραφή αυτή γίνεται κατανοητό ότι η διεργασία ενημέρωσης superblock χρειάζεται και αυτή να δεχτεί σαν είσοδο το αναγνωριστικό για τον τύπο αρχείου που δημιουργείται.



Εικόνα 5.19 : Διάγραμμα ροής διεργασίας ενημέρωσης Superblock

```
procedure updateSuperblock(command)
begin
updatefreeinode:
sbins.free_inodes_count := sbins.free_inodes_count − 1 ;
updatefreeblocks:
if command = "slink" then
                sbins.free_blocks_count := sbins.free_blocks_count − 1 ;
       elsif   command = "mkdir" then
                sbins.free_blocks_count := sbins.free_blocks_count − 1 ;
      end if ;
goback:
return ;
end procedure ;
```

Εικόνα 5.20 : Τυπική προδιαγραφή διεργασίας ενημέρωσης Superblock

5.4 Invariants Και Temporal Properties

Για την ομαλή λειτουργία και την διασφάλιση ότι η διαδικασία δημιουργίας αρχείου θα λειτουργεί σωστά θα πρέπει η διαδικασία και οι διεργασίες να τηρούν ένα σύνολο από κανόνες και ιδιότητες. Στο σημείο αυτό παρουσιάζονται το σύνολο όλων των ιδιοτήτων και τον κανόνων που χρειάζονται για την διαδικασία δημιουργίας αρχείου και τον διεργασιών που την αποτελούν.

5.4.1 Invariants

Οι ιδιότητες αναλλοίωτης λειτουργίας (invariants όπως αναφέρθηκε στο Κεφάλαιο 3), είναι οι ιδιότητες που προσδιορίζουν ότι τα χαρακτηριστικά του συστήματος θα παραμείνουν αμετάβλητα κατά την μετάβαστου μοντέλου διαμέσου διαφορετικών του καταστάσεων. Επίσης σύμφωνα με τον Leslie Lamport [17] οι ιδιότητες αναλλοίωτης λειτουργίας χρησιμοποιούνται ώστε να αποφύγουμε τις ανεπιθύμητες αλλαγές και τα ανεπιθύμητα αποτελέσματα στο σύστημα μας. Με βάση λοιπόν τα κριτήρια αυτά οι ιδιότητες αναλλοίωτης λειτουργίας για την διαδικασία δημιουργίας αρχείου και των επιμέρους διεργασιών της είναι οι εξής:

1. **Free node Exists Invariant**: Στο σύστημα υπάρχει ελεύθερο inode
2. **InodeInRange**: Η τιμή του πεδίου inode της δομής directory entry πρέπει να είναι στο εύρος του συνολικού πλήθους των inode του συστήματος
3. **BlockNum**; Αν υπάρχει inode που έχει δεσμευμένο block τότε για κάθε δεσμευμένο block οι τιμές του πίνακα direct Addr πρέπει να ανήκουν στο εύρος των διαθέσιμων block που έχει το blockgroup που ανήκει το αρχείο
4. **FreeInodeCountProp**: Το σύνολο των ελευθέρων inode στον inodeBitmap είναι ίσο με το σύνολο των ελευθέρων inode στην μεταβλητή Free Inode Count
5. **FreeBlockCountProp** : Το σύνολο των ελευθέρων block στον blockBitmap είναι ίσο με το σύνολο των ελευθέρων inode στην μεταβλητή Free Block Count
6. **Valid RecLen**: Για κάθε καταχώρηση στο directory linked list το μήκος του αρχείου σε byte πρέπει να είναι μεγαλύτερο του 0

7. **Valid NameLen**: Για κάθε καταχώρηση στο directory linked list το μήκος του ονόματος του αρχείου πρέπει να είναι μεγαλύτερο του 0
8. **Valid Filename**: Για κάθε καταχώρηση στο directory linked list το όνομα του αρχείου δεν πρέπει να είναι κενό
9. **max_file_size**: Για κάθε αρχείο στο σύστημα το πεδίο size του inode θα πρέπει να μεταξύ 0 και sb.blocks per group * sb.log_block_size
10. **Inode Count Invariant**: Το πλήθος των inode στο superblock πρέπει να είναι ίσο με το πλήθος των inode που έχει το σύστημα
11. **Block Count Invariant**: Το πλήθος των block στο superblock πρέπει να είναι ίσο με το πλήθος των block που έχει το σύστημα
12. **FreeBlockCountInvariant**: Το πλήθος των ελευθέρων inode δεν πρέπει να είναι μεγαλύτερο από το πλήθος των συνολικών inode του συστήματος
13. **FreeInodeCountInvariant**: Το πλήθος των ελευθέρων block δεν πρέπει να είναι μεγαλύτερο από το πλήθος των συνολικών block του συστήματος

Πίνακας 5.6 : Δηλώσεις invariants

Invariants	Ορισμος σε TLA+
FreeInodeExistsInvariant	∃ □ ∈ 0..□□.□□□□□□ □□□ □□□□□ − 1 : □□□□□□□□□□□□[□] = "□□□□"
Inode In Range	□□.□□□□□ ∈ 0..□□□□□ □□□□□ − 1
BlockNum	∃ □ ∈ 0..□□.□□□□□□ □□□ □□□□□ − 1 : (□□□□□□□□□□□[□].□□□□□□ > 0 => ∀ □ ∈ 0..□□□□□□□□□□□[□].□□□□□□: □□□□□□□□□□□[□].□□□□□□□□□□□[□] ∈ 0..□□.□□□□□□_□□□_□□□□□ − 1)
Free Inode Count Prop	∀ □ ∈ □□□□□□□□□□□□□□□: □□□□□□□□□□□□[□] = "□□□□"
Free Block Count Prop	∀ □ ∈ □□□□□□□□□□□□□□□: □□□□□□□□□□□□[□] = "□□□□"
Valid RecLen	∀ □ ∈ 1..□□□ □□□□ : □□□□□□□□[□].□□□ □□□ > 0
Valid NameLen	∀ □ ∈ 1..□□□□□□□□ : □□□□□□□□[□].□□□□_□□□ > 0
Valid Filename	∀ □ ∈ 1..□□□□□□□□ : □□□□□□□□[□].□□□□□□□□ ≠ ""
max_file_size	∀ □ ∈ 0..□□.□□□□□□ □□□ □□□□□ − 1 : □□□□□□□□□□□[□].□□□□ <= □□.□□□□□□_□□□_□□□□□ * □□.□□□_□□□□□_□□□□
Inode Count Invariant	□□.□□□□□□_□□□□□ = □□□□□□□□□□
BlockCount Invariant	□□.□□□□□□_□□□□□ = □□□□□□□□□□
FreeBlockCount Invariant	□□.□□□□_□□□□□□_□□□□□ <= □□□□□□□□□□

FreeInodeCountInvariant	□□.□□□□_□□□□□□_□□□□□ <= □□□□□□□□□□

5.4.2 Temporal Properties

Οι χρονικές ιδιότητες (temporal properties), αναφέρονται σε δηλώσεις ή συνθήκες που περιγράφουν πώς εξελίσσεται η συμπεριφορά ενός συστήματος με την πάροδο του χρόνου [17]. Συγκεκριμένα για την διαδικασία δημιουργίας αρχείου οι χρονικές ιδιότητες που χρησιμοποιούνται, έχουν να κάνουν με τις επιθυμητές αλλαγές που γίνονται στο σύστημα (π.χ. η τιμή του πεδίου mode στο inode). Οπότε με βάση λοιπόν το κριτήριο αυτό οι ιδιότητες έχουν ως εξής :

1. **AllInodesCovered:** Τα στοιχεία του inode Bitmap μπορούν να πάρουν τιμές free ή allocated
2. **All blocks Covered:** Τα στοιχεία του block Bitmap μπορούν να πάρουν τιμές free ή allocated
3. **ValidMode:**Το πεδίο mode του inode μπορεί να πάρει τιμές none, ext2_reg ,ext2_symlink,ext2_dir
4. **Inode Allocation:** Το πεδίο dtime του inode μπορεί να πάρει τιμές 0 ή -1.

Πίνακας 5.7 : Δηλώσεις temporal properties

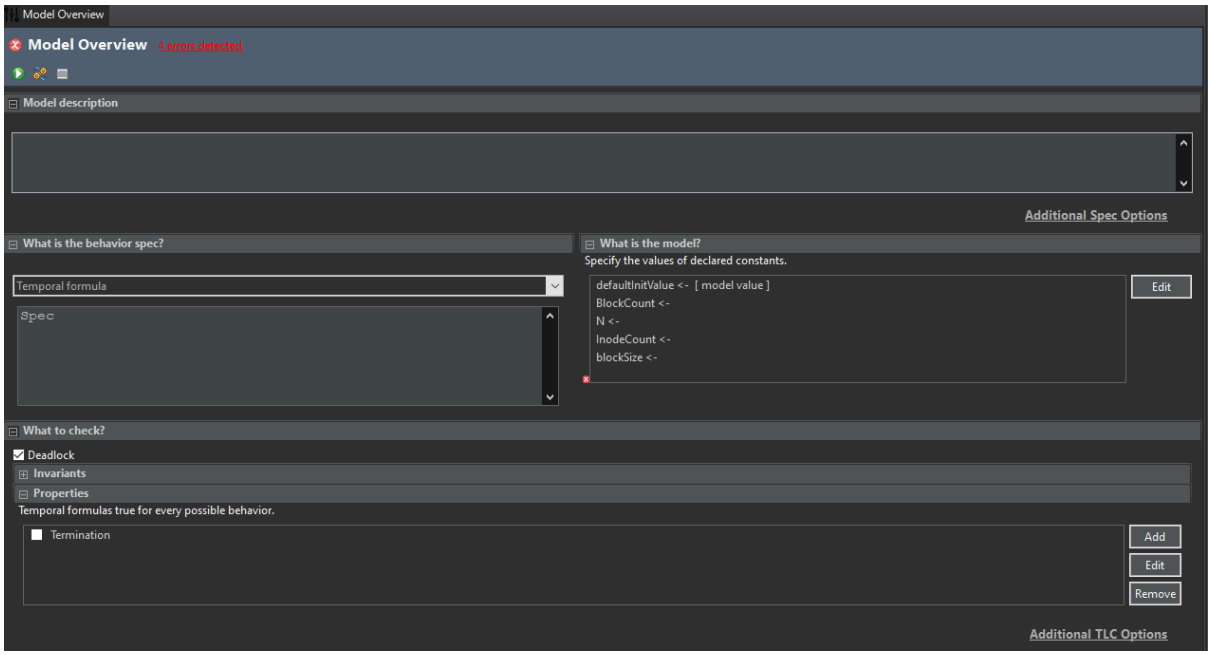
temporal properties	Ορισμος σε TLA+
AllInodesCovered	[]($\forall$ □ $\in$ *0*..□□.□□□□□□ □□□ □□□□□ − *1* : □□□□□ □□□□□□□[□] $\in$ {"□□□□","□□□□□□□□□□"})
All blocks Covered	[]($\forall$ □ $\in$ *0*..□□.□□□□□□_□□□_□□□□□ − *1* : □□□□□ □□□□□□□[□] $\in$ {"□□□□","□□□□□□□□□□"})
Valid Mode	[]($\forall$ □ $\in$ *0*..□□.□□□□□□ □□□ □□□□□ − *1* : □□□□□ □□□□□[□□].□□□□ $\in$ {"□□□*2*_□□□","□□□*2*_□□□","□□□*2*_□□□□□□□","□□□□"})
Inode Allocation	[]($\forall$ □ $\in$ *0*..□□.□□□□□□ □□□ □□□□□ − *1* : □□□□□□□□□□[□].□□□□□ $\in$ {*0*, −*1*})

Ενδεικτικές Εκτελέσεις και Αποτελέσματα

Στο σημείο αυτό, παρουσιάζονται ενδεικτικές εκτελέσεις πάνω σε διάφορα use cases και τα αποτελέσματά τους. Τα use case που παρουσιάζονται αφορούν την παραβίαση κάποιων invariants και temporal properties για την επιβεβαίωση ότι οι ιδιότητες και οι κανόνες που ορίστηκαν τηρούνται καθώς και ένα use case που δείχνει την σωστή συμπεριφορά του συστήματος.

6.1 Δημιουργία Μοντέλου

Το πρώτο βήμα που πρέπει να γίνει για την εξέταση των use cases πρέπει αρχικά ο κώδικας plus Cal να μεταφραστεί σε κώδικα TLA+. Για να γίνει αυτό στο περιβάλλον TLA+ toolbox χρησιμοποιούνται τα πλήκτρα ctrl + t. Όταν η μετάφρασή ολοκληρωθεί το IDE θα υποδείξει ότι η κατάσταση της προδιαγραφής έχει αναλυθεί(spec status parsed). Στην συνέχεια πηγαίνοντας στο πάνω μέρος του ide επιλέγοντας την επιλογή tlc model checker και στην συνέχεια new model το IDE δημιουργεί ένα νέο μοντέλο. Το παράθυρο του μοντέλου έχει ως εξής.



Εικόνα 6.1 : Παράθυρο μοντέλου

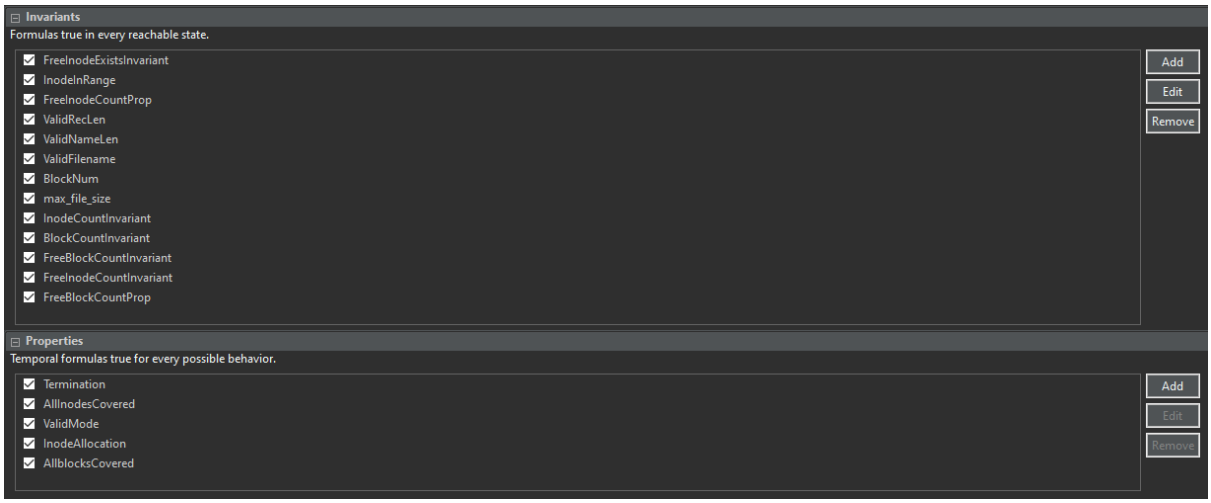
Το επόμενο βήμα για την δημιουργία μοντέλου αφορά την αρχικοποίηση των σταθερών που έχουν οριστεί από την προδιαγραφή. Οι τιμές[6] αυτές έχουν ως εξής:

- **BlockCount:** 50
- **N:** 5

[6] Οι τιμές είναι ενδεικτικές καθώς όσο μεγαλώνουν σε μέγεθος τόσο περισσότερους ελέγχους θα χρειαστεί να κάνει ο ελεγκτής μοντέλου

- **Inode Count:** 50
- **block Size:** 1024

Τέλος για την ολοκλήρωση του μοντέλου στα παράθυρα invariants και properties τοποθετούνται τα ονόματα των invariants και των temporal properties που έχουν οριστεί.



Εικόνα 6.2 : invariants και temporal properties

Πριν την εκτέλεση του μοντέλου πρέπει πρώτα να επεξηγηθούν οι νέες ιδιότητες που υπάρχουν στο παράθυρο του μοντέλου καθώς και η επιλογή deadlock. Η πρώτη ιδιότητα προς επεξήγηση είναι η ιδιότητα termination. Η ιδιότητα αυτή, είναι μία ιδιότητα που δημιουργείται αυτόματα από τον μεταφραστή η οποία δηλώνει ότι κάποια στιγμή όλες οι διεργασίες θα τερματίσουν. Η ιδιότητα termination χρησιμοποιείται για την διασφαλίση ότι οι διαδικασίες θα είναι “δίκαιες” η μία απέναντι στην άλλη. Για την εξασφάλιση ότι η ιδιότητα αυτή πρέπει να ισχύει θα πρέπει στην δήλωση της διαδικασίας που είναι υπεύθυνη για την κλήση της διαδικασίας δημιουργίας αρχείου, να τοποθετηθεί η λέξη fair. Παρόλα αυτά δεν έχει γίνεται η χρήση της λέξης fair καθώς επιτηδευμένα πρέπει να παραβιαστεί η ιδιότητα αυτή για την εξέταση όλων των use cases μέσα από το παράθυρο error trace που προσφέρει το tlc model checker έτσι ώστε σε περίπτωση που το σύστημα λειτουργεί σωστά, να δοθεί ένα αντι-παράδειγμα που να παρουσιάζει την διαδρομή που ακολούθησε ο ελεγκτής για να βρει το επιτηδευμένο σφάλμα αυτό. Έτσι σαν αποτέλεσμα παρέχετε μία διαδρομή με όλα τα επιθυμητά αποτελέσματα καθώς και η δυνατότητα εξετάσεις ότι οι τιμές των μεταβλητών της διαδικασίας έχουν λάβει τις τιμές που εκτιμήθηκαν ότι θα λάβουν.

Η δεύτερη κατά σειρά ιδιότητα είναι η ιδιότητα που εμφανίζεται στο παράθυρο what is the behaviour spec? . Η ιδιότητα αυτή είναι μία ιδιότητα που δημιουργείται αυτόματα από τον μεταφραστή και υποδηλώνει στον ελεγκτή την συμπεριφορά της προδιαγραφής. Πιο συγκεκριμένα η ιδιότητα περιγράφει τον τρόπο με τον οποίο γίνεται η αλλαγή των καταστάσεων όταν ο ελεγκτής τρέχει το μοντέλο.

Τέλος η επιλογή deadlock είναι μία επιλογή που υποδηλώνει στον ελεγκτή να εξετάσει το μοντέλο και να βρει κατάσταση αδιεξόδου. Δηλαδή να βρει καταστάσεις που το μοντέλο σταματά για κάποιον ανεξήγητό λόγο. Η επιλογή αυτή είναι σημαντική καθώς ένα σύστημα πρέπει πάντα να είναι σε λειτουργία και να μην σταματάει ποτέ.

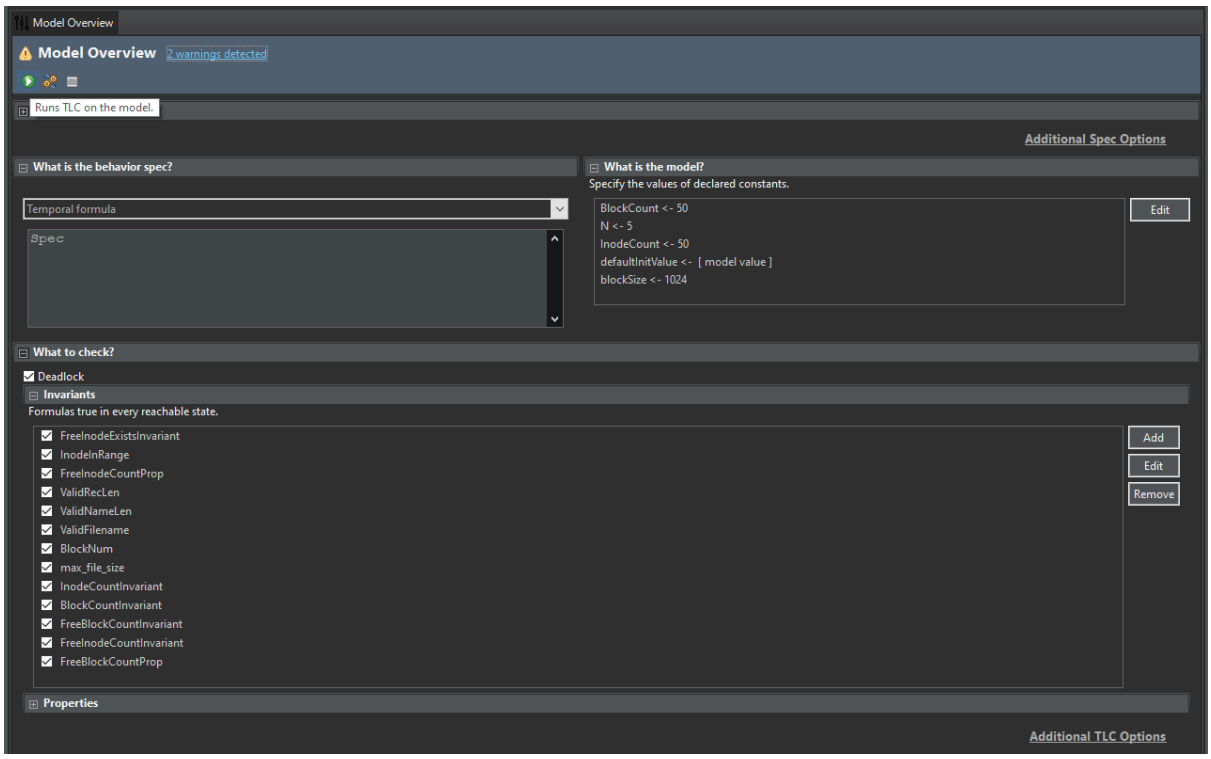
6.2 Έλεγχος Παραβίασης Invariant

Σαν πρώτο use case επιλέχθηκε η παραβίαση του invariant **FreenodeExistsInvariant**. Όπως αναφέρθηκε στο 5.4.1 το invariant Free node Exists Invariant υποδηλώνει ότι για την δημιουργία του αρχείου πρέπει πάντα να υπάρχει ένα ελεύθερο inode, κάτι το οποίο καθιστά το invariant αυτό σημαντικό για την ορθή και σωστή λειτουργία μοντέλου.
Συνεχίζοντας με το use case το πρώτο πράγμα που πρέπει να γίνει να αλλάξει η τιμή των inode στο inodeBitmap από **"FREE"** σε **"ALLOCATED"**. Αυτό θα "αναγκάσει" τον ελεγκτή να σταματήσει τον έλεγχο από την πρώτη κιόλας κατάσταση.

$$inodeBitmap = [i \in 0 \,..\, sb.inodes\_per\_group - 1 \mapsto \text{"ALLOCATED"}],$$

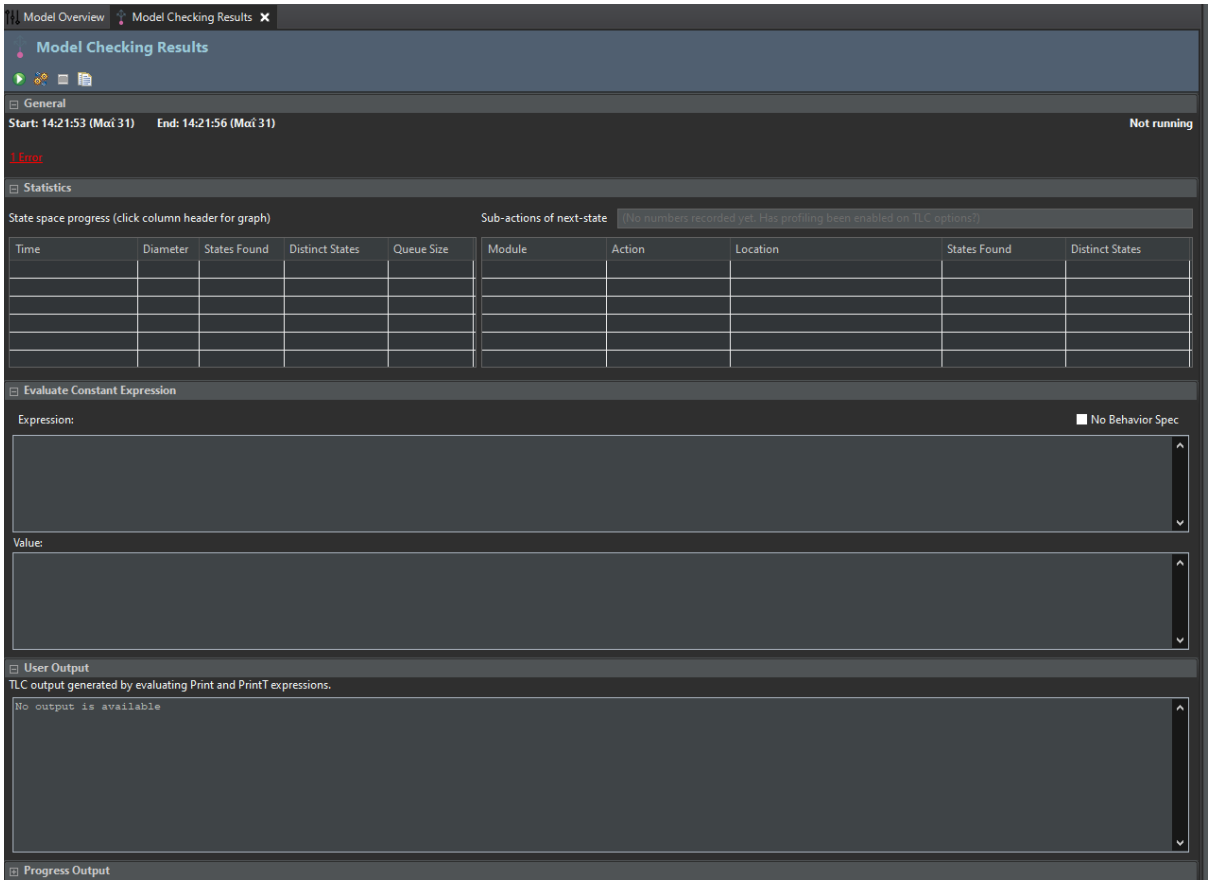
Εικόνα 6.3 : Αλλαγή τιμής free σε allocated.

Εφόσον γίνει η αλλαγή το επόμενο βήμα είναι να τρέξει ο ελεγκτής μοντέλου.



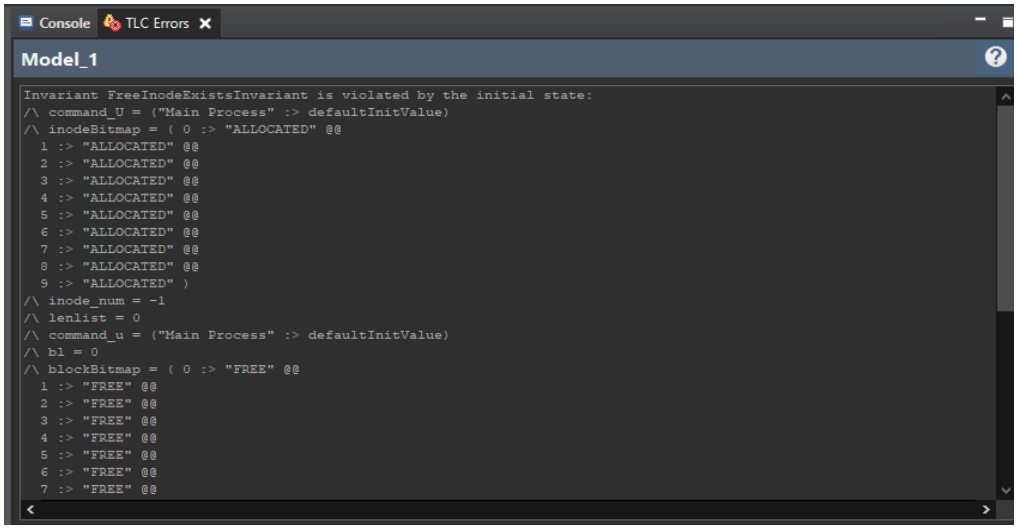
Εικόνα 6.4 : Τρέξιμο μοντέλου

Πατώντας το κουμπί start, όταν ο ελεγκτής ολοκληρώσει τον έλεγχο, ανοίγει το παράθυρό που εμφανίζεται στην παρακάτω εικόνα.



Εικόνα 6.5 : Παράθυρο αποτελεσμάτων.

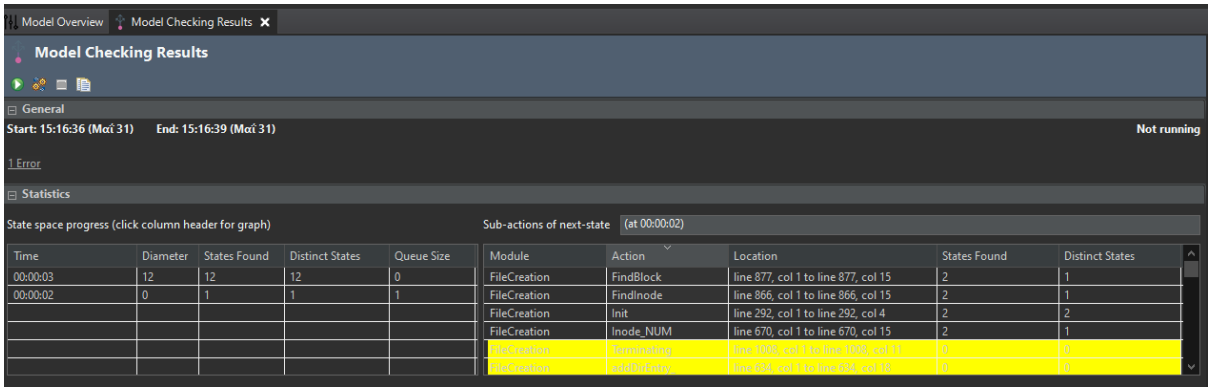
Το παράθυρο αυτό δίνει της απαραίτητες πληροφορίες σχετικά με τον έλεγχο που μόλις διεξάχθηκε και μπορούμε να δούμε ότι έχει εντοπίσει ένα πρόβλημα. και ακριβώς από δίπλα στο παράθυρο της κονσόλας βλέπουμε το αντίστοιχο μήνυμα. Επίσης στο παράθυρο αποτελεσμάτων, εμφανίζονται δύο πίνακες. Ο πρώτος πίνακας δείχνει τον χρόνο που ο ελεγκτής έκανε για να ολοκληρώσει τον έλεγχο, την διάμετρο του δέντρου καταστάσεων, της καταστάσεις που βρήκε (states found) και τέλος τις καταστάσεις που επισκέφτηκε (distinct states) και το μέγεθος της ουράς. Στην προκειμένη περίπτωση δεν υπάρχει καμία καταχώρηση στους πίνακες πράγμα που σημαίνει ότι ο ελεγκτής σταμάτησε στην αρχική κατάσταση κάτι το οποίο ήταν επιθυμητό. Πηγαίνοντας στην συνέχεια στο παράθυρο μηνυμάτων, παρατηρείται το εξής μήνυμα “Invariant FreeInodeExistsInvariant is violated by the initial state:”. Δηλαδή το invariant **FreenodeExistsInvariant** έχει παραβιαστεί στην αρχική κατάσταση του μοντέλου.



Εικόνα 6.6 : Παράθυρο μηνυμάτων.

6.3 Έλεγχος Παραβίασης Temporal Property

Το επόμενο κατά σειρά use case είναι το use case παραβίασης temporal property. Η ιδιότητα που θα χρησιμοποιηθεί για την παραβίαση είναι η ιδιότητα ValidMode. Πιο συγκεκριμένα η τιμή που θα πάρει το InodeTable[inode_num].mode := "ext2_symlink" , θα αλλάξει σε “symlink”. Η αλλαγή αυτή έχει σκοπό ο ελεγκτής να σταματήσει όταν γίνει η αλλαγή του πεδίου mode απο “none” σε “symlink”.
Όπως και στο προηγούμενο use case έτσι και εδώ, στην εικόνα που ακολουθεί παρουσιάζονται τα αποτελέσματα από τον έλεγχο που διεξήχθη.

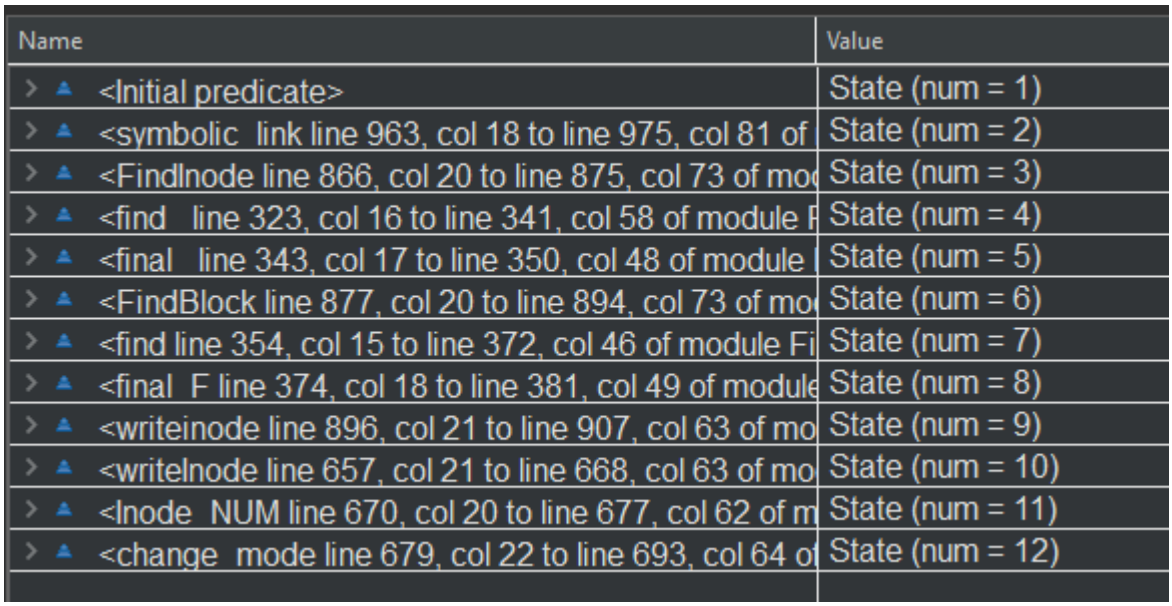


Εικόνα 6.7 : Παράθυρο αποτελεσμάτων use case 2

Σε αντίθεση με το use case 1 βλέπουμε ότι στον use case αυτό ο ελεγκτής έχει βρει 12 καταστάσεις, κάτι το οποίο υποδηλώνει το γεγονός ότι ο ελεγκτής έτρεξε μέχρι ένα σημείο. Στον διπλανό πίνακα παρουσιάζονται οι ενέργειες που έγιναν κατά την διάρκεια του ελέγχου. Παρατηρώντας λοιπόν τις ενέργειες αυτές, παρατηρείται ότι κάποιες από αυτές είναι τονισμένες με ένα κίτρινο χρώμα κάτι που

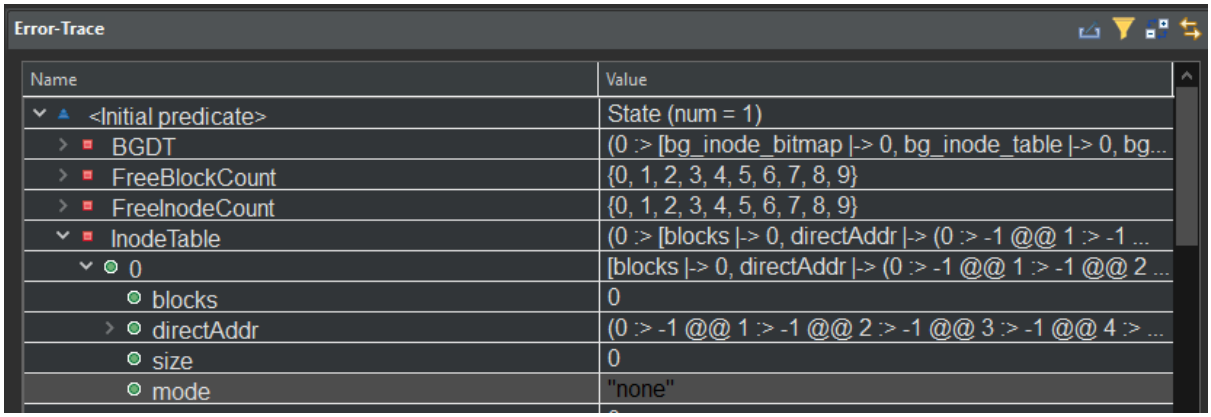
υποδηλώνει ότι οι ενέργειες αυτές είναι ενέργειες που δεν εκτελέστηκαν. Πηγαίνοντας στο παράθυρο μηνυμάτων εμφανίζεται το μήνυμα ότι "Invariant[7] ValidMode is violated. ".

Συνεχίζοντας λοιπόν την εξέταση των αποτελεσμάτων το επόμενο βήμα είναι η εξέταση του παραθύρου error trace.



Εικόνα 6.8 : Παράθυρο error trace

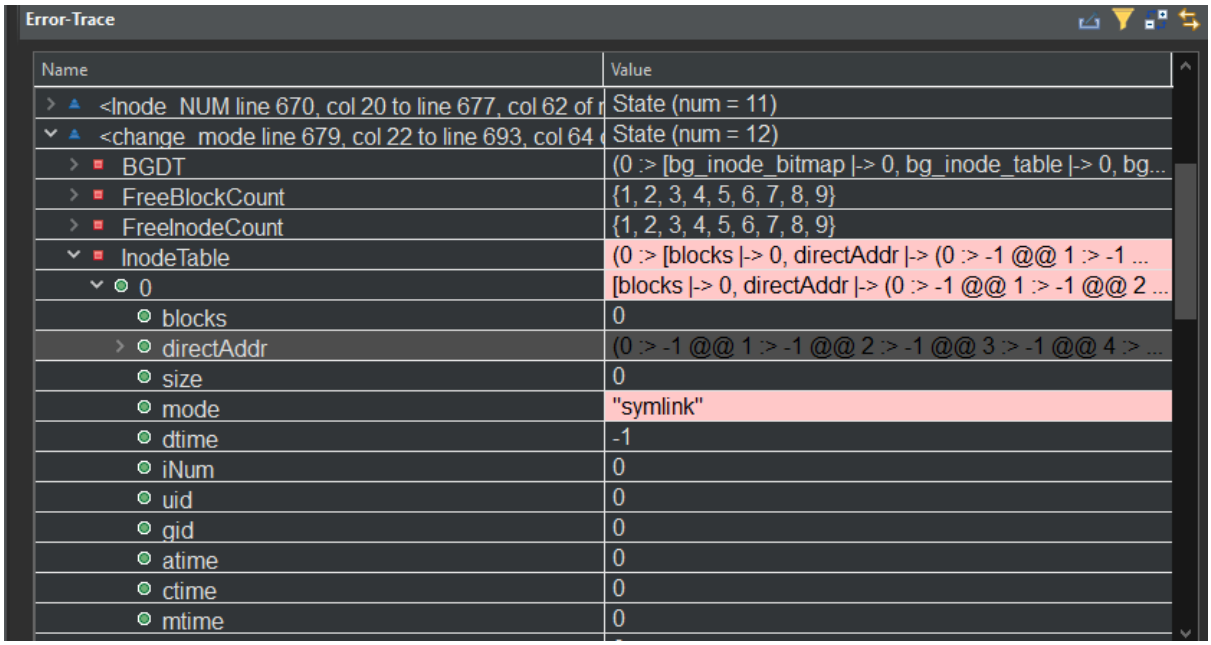
Το παράθυρο αυτό δίνει την διαδρομή που ακολούθησε ο ελεγκτής μέχρι να τερματίσει και μέσα σε αυτό παρέχονται όλες οι πληροφορίες σχετικά με τα περιεχόμενα των στοιχείων που απαρτίζουν το μοντέλο που ελέγχθηκε. Κάνοντας επέκταση της πρώτης κατάστασης (initial predicate), και πηγαίνοντας στην επιλογή InodeTable και ανοίγοντας το πρώτο inode παρατηρείται ότι τα πεδία του inode έχουν τις αρχικές τιμές που είχαν οριστεί απο την προδιαγραφή.



Εικόνα 6.9 : Παράθυρο error trace

[7] Το tlc model checker έχει συγκεκριμένα ονόματα για να προσδιορίσει τα invariants και τα temporal properties σε περίπτωση που το όνομα δεν ταιριάζει με το όνομα που περιμένει το tlc model checker εμφανίζει όλα τα ονόματα που έχει δώσει ο χρήστης σαν invariants.

Στην συνέχεια κάνοντας ακριβώς τα ίδια βήματα αλλά στην τελευταία κατάσταση (change mode) παρατηρείται ότι το πεδίο που σχετίζεται με το mode είναι κόκκινο κάτι που υποδηλώνει την αλλαγή που υπάρχει σε σχέση με την προηγούμενη κατάσταση. Επίσης βλέποντας το πεδίο block παρατηρείται ότι δεν έχει λάβει κάποια τιμή που σημαίνει ότι ο ελεγκτής σταμάτησε και δεν προχώρησε σε άλλη κατάσταση καθώς η κατάσταση αυτή δεν είναι επιθυμητή κατάσταση με βάση τις ιδιότητες που έχουν οριστεί.



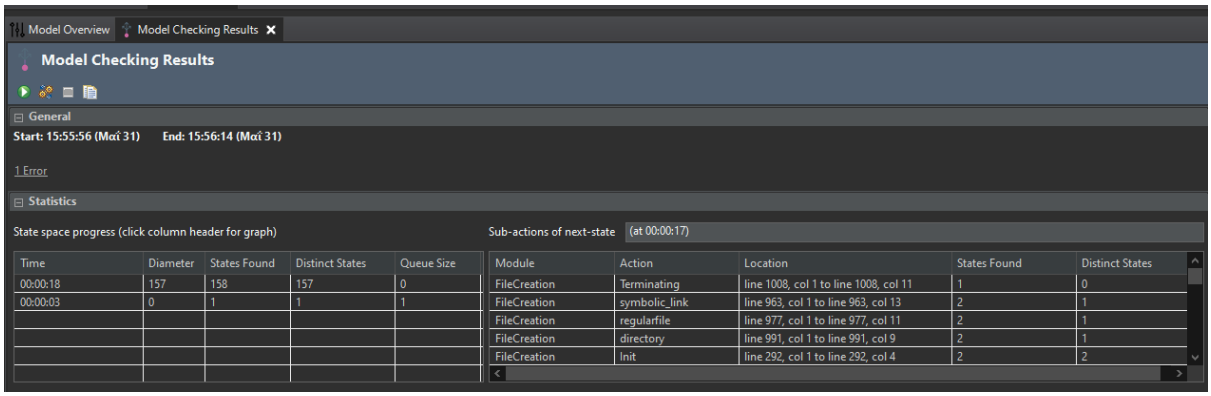
Εικόνα 6.10 : Παράθυρο error trace

6.4 Τελικός Έλεγχος Μοντέλου

Το τελευταίο use case είναι ο τελικός έλεγχος του μοντέλου. Σε αυτήν την περίπτωση θα παραβιαστεί η ιδιότητα termination. Όπως αναφέρθηκε και στο 6.1 η ιδιότητα termination είναι η ιδιότητα που χρησιμοποιείται ώστε οι διαδικασίες να είναι "δίκαιες" η μία απέναντι στην άλλη. Ο όρος δίκαιες σημαίνει ότι μία διαδικασία κάποια στιγμή πρέπει να παραχωρήσει προτεραιότητα σε μία άλλη ώστε να τερματίσει. Ο σκοπός της παραβίασης αυτής γίνεται εσκεμμένα με σκοπό την δημιουργία ενός αντιπαραδείγματος που να περιέχει όλες τις καταστάσεις μέχρι να βρεθεί η μη αποδεκτή κατάσταση δηλαδή η κατάσταση που η ιδιότητα termination παραβιάζεται. Μέσα από αυτό το αντιπαράδειγμα θα εξασφαλιστεί, ότι ο ελεγκτής θα βρεί όλες τις καταστάσεις που τηρούν όλες τις ιδιότητες που έχουν οριστεί. Οι αλλαγές που πρέπει να παρατηρηθούν είναι οι εξής:

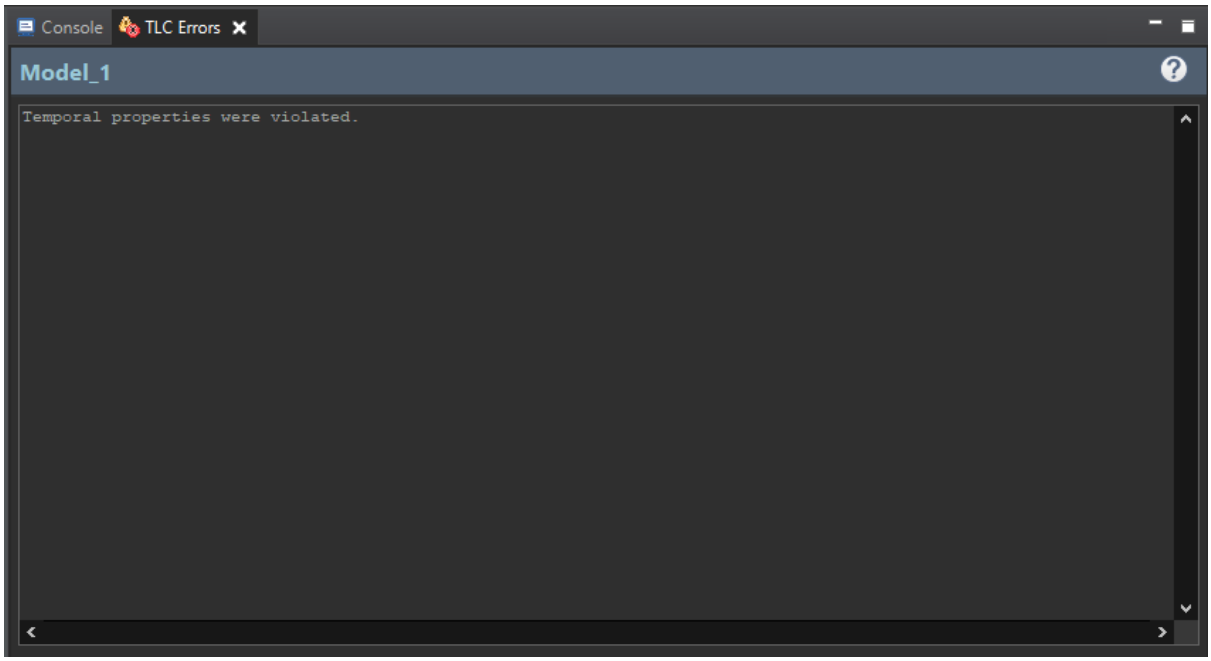
- Μικρότερο πλήθος ελεύθερων inode
- Μικρότερο πλήθος ελεύθερων block
- 3 αρχεία (ένα για κάθε τύπο) καταχωρημένα στο inode table
- Αλλαγμένες τιμές σε superblock και BGDT

- Τρεις καταχωρήσεις στο directory entry list



Εικόνα 6.11 : Παράθυρο αποτελεσμάτων

Παρατηρώντας τους πίνακες στο παράθυρο των αποτελεσμάτων, βλέπουμε ότι ο ελεγκτής βρήκε 158 καταστάσεις και από αυτές προσπέλασε 157. Επίσης στο παράθυρο των σφαλμάτων παρατηρήται ότι η ιδιότητα termination έχει παραβιαστεί.



Εικόνα 6.12 : Παράθυρο σφαλμάτων

Παρατηρώντας το παράθυρο error trace και ακολουθώντας τα βήματα που εφαρμόστηκαν στο 6.3 και συγκρίνοντας τα αποτελέσματα[8] της κατάστασης initial predicate και την κατάσταση goback, παρατηρούνται οι εξής αλλαγές:

- Το πλήθος των blogs είναι μικρότερο από το αρχικό
- Το πλήθος των inode είναι μικρότερο από το αρχικό
- Το inode 0,1,3 έχουν δεσμευτει με τα κατάλληλα στοιχεία
- Υπάρχουν τρεις καταχωρήσεις στο directory entry list
- Ενημερωμένες τιμές σε BGDT και Superblock

Οι αλλαγές αυτές υποδηλώνουν ότι ο ελεγκτής πέρασε από όλες τις επιθυμητές καταστάσεις ώστε να δημιουργηθούν τα τρία αρχεία που έχουν οριστεί στην προδιαγραφή. Επίσης αυτό σημαίνει ότι οι ιδιότητες και οι κανόνες που έχουν οριστεί τηρούνται οπότε η προδιαγραφή για την δημιουργία αρχείου συμπεριφέρεται σύμφωνα με τις προδιαγραφές που έχουν οριστεί.

[8] Τα screenshot με τα αποτελέσματα βρίσκονται στο παράρτημα 3

Συμπεράσματα

Η παρούσα διπλωματική εργασία διερεύνησε την εφαρμογή των τυπικών μεθόδων χρησιμοποιώντας τον καθορισμό της διαδικασίας δημιουργίας αρχείων του συστήματος αρχείων ext2 σαν περίπτωση μελέτης. Χρησιμοποιώντας τις γλώσσες τυπικών προδιαγραφών pluscal και tla+ δημιουργήθηκε μια σαφής και ξεκάθαρη προδιαγραφή που αποτυπώνει με ακρίβεια τις βασικές πτυχές της διαδικασίας δημιουργίας αρχείων. Και μέσα από την εφαρμογή των τεχνικών τυπικής επαλήθευσης, πραγματοποιήθηκε μία ενδελεχής και εξαντλητική ανάλυση της προδιαγραφής για την επαλήθευση ότι το σύστημα συμπεριφέρεται σωστά αλλά, και στον εντοπισμό και την εξάλειψη πιθανών σφαλμάτων και ελαττωμάτων σχεδιασμού. Επιπλέον, οι γνώσεις που αποκτήθηκαν από αυτή την έρευνα συμβάλλουν στο ευρύτερο πεδίο των τυπικών μεθόδων και της εφαρμογής τους σε συστήματα αρχείων αλλά και σε συστήματα λογισμικού γενικότερα και το πώς μπορούν να συνεισφέρουν στον σχεδιασμό αξιόπιστων και ασφαλών συστημάτων λογισμικού. Τέλος τα ευρήματα της έρευνας αυτής, μπορούν να επεκταθούν και να εφαρμοστούν σε άλλα κομμάτια του ίδιου του συστήματος αρχείων, ή και για την προτυποποίηση και δημιουργία ενός νέου συστήματος αρχείων που βασίζεται στο ext2.

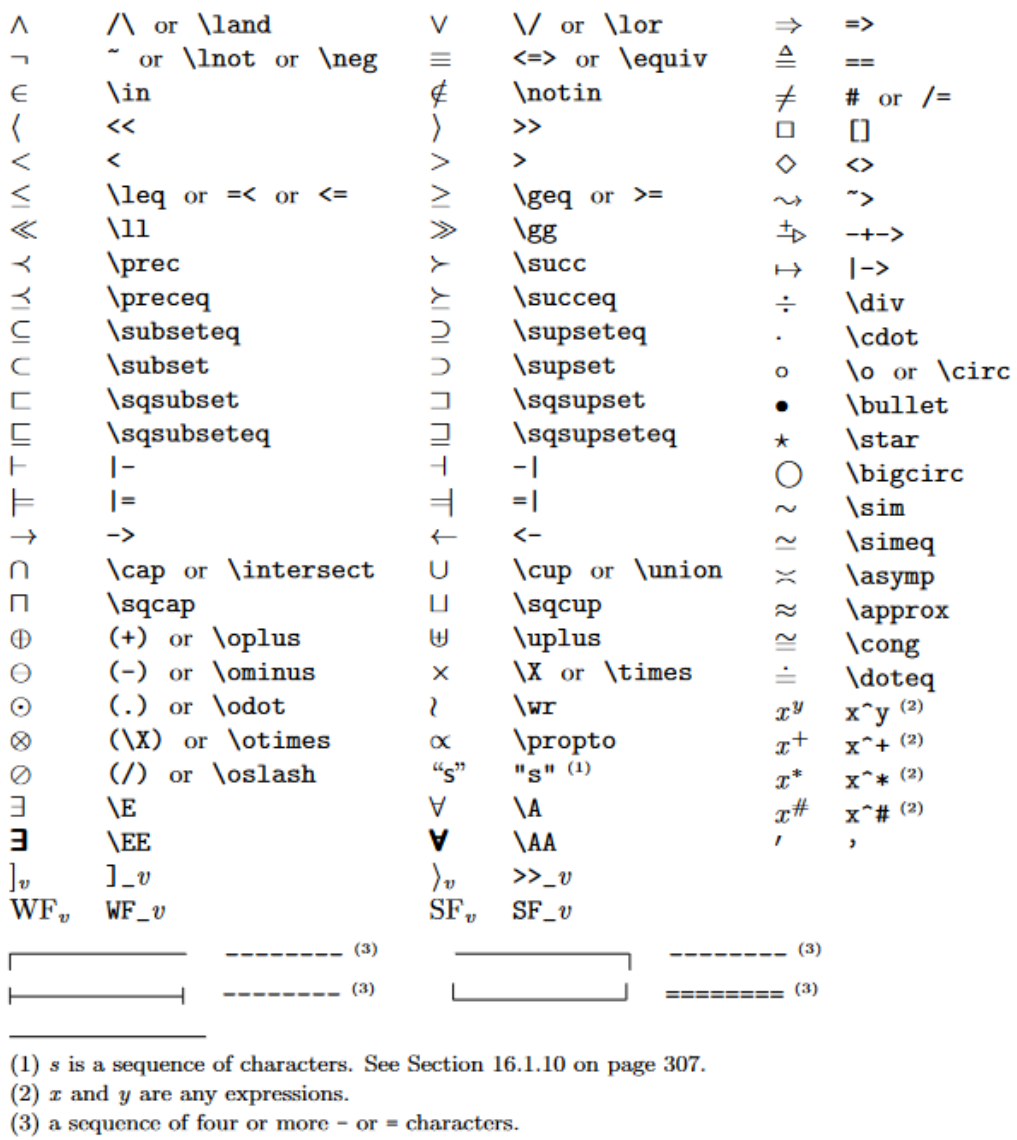
Παράρτημα Ι

Πίνακας με δεσμευμένες λέξεις TLA+

ASSUME	ELSE	LOCAL
UNION	ASSUMPTION	ENABLED
MODULE	VARIABLE	AXIOM
EXCEPT	OTHER	VARIABLES
CASE	EXTENDS	SF_
WF_	CHOOSE	IF
SUBSET	WITH	CONSTANT
IN	THEN	CONSTANTS
INSTANCE	THEOREM	DOMAIN
LET	UNCHANGED	

Modules TLA+

Naturals
Integers
Reals
Sequences
FiniteSets
Bags
RealTime
TLC



(1) s is a sequence of characters. See Section 16.1.10 on page 307.
(2) x and y are any expressions.
(3) a sequence of four or more - or = characters.

Εικόνα 8.1: TLA+ operators Πηγή : [17]

PlusCal Δεσμευμένες λέξεις

If	Print
Either	Assert
While	Call and Return
Await	Got
With	
Skip	

Κανόνες συγγραφής αλγορίθμου σε γλώσσα pluscal

"Ο κώδικας για κάθε διεργασία (και ένα πρόγραμμα uniprocess) πρέπει να ξεκινά με μία ετικέτα.

- Μια δήλωση while πρέπει να φέρει ετικέτα.
- Η ρήτρα do μιας δήλωσης με δεν μπορεί να περιέχει καμία επισημασμένη κατάσταση-εντολές. Αυτός ο κανόνας περιορίζει ό,τι μπορεί να εμφανίζεται σε μια δήλωση with— για παράδειγμα, απαγορεύει να φωλιάζουμε για λίγο μέσα σε ένα με. Αυτοί οι περιορισμοί-ισχύουν ακόμη και όταν αφήνετε τον μεταφραστή να προσθέσει ετικέτες. Το πλήρες λίστα τέτοιων περιορισμών δίνεται στην Ενότητα 3.2.6 στη σελίδα 23.
- Μια δήλωση if που περιέχει μια ετικέτα μέσα σε αυτή πρέπει να ακολουθείται από μία ετικέτα. Για παράδειγμα, η ετικέτα l5 του αλγορίθμου FastMutex δεν μπορεί να είναι παραλείπεται.
- Σε οποιαδήποτε διαδρομή ελέγχου από τη μία ετικέτα στην άλλη, δεν μπορούν να υπάρχουν δύο ξεχωριστές δηλώσεις ανάθεσης στην ίδια μεταβλητή. Για παράδειγμα, αυτός ο κανόνας θα παραβιαζόταν από δύο εκχωρήσεις στη μεταβλητή b εάν ετικέτες Τα l1, l2 και l3 καταργήθηκαν όλα—ακόμα κι αν μία από τις εργασίες ήταν ένα διαφορετικό στοιχείο του πίνακα. Ωστόσο, μια ενιαία πολλαπλή ανάθεση όπως x[1] := 1 || x[2] := 2 μπορεί να αντιστοιχίσει σε διαφορετικά στοιχεία της ίδιας μεταβλητής.
- Στη σύνταξη c, δεν μπορεί να προηγηθεί και να προηγηθεί ένα ανοιγόμενο στήριγμα χαμηλώνεται από μια ετικέτα—δηλαδή, δεν μπορείτε να γράψετε κάτι σαν "l1: { l2: "

[19, σ. 17]

Παράρτημα ΙΙ

Κώδικας createfile

MODULE *FileCreation*

EXTENDS *Integers*, *TLC*, *Sequences*, *FiniteSets*, *inode*

--**algorithm** *Createfile*

variables
inodeBitmap = [*i* ∈ 0 .. *sb.inodes_per_group* − 1 ↦ "FREE"],
inode bitmap, with each bit representing the allocation status of an *inode*
blockBitmap = [*i* ∈ 0 .. *sb.blocks_per_group* − 1 ↦ "FREE"],
block_num = −1,
bl = 0,
in = 0,
FreeInodeCount = {*i* ∈ 0 .. *sb.inodes_per_group* − 1 : *inodeBitmap*[*i*] = "FREE"},
FreeBlockCount = {*i* ∈ 0 .. *sb.blocks_per_group* − 1 : *blockBitmap*[*i*] = "FREE"},
InodeTable = [*i* ∈ 0 .. *sb.inodes_per_group* − 1 ↦ *Inode*],
de = *dir_entry*,
lenlist = 0,
dirlist = ⟨⟩,
BGDT = *BlockGroupDescriptorTable*,
sbins = *sb*,
inode_num = −1 ;

define
FreeInodeExistsInvariant ≜ ∃ *i* ∈ 0 .. *sb.inodes_per_group* − 1 : *inodeBitmap*[*i*] = "FREE"
InodeInRange ≜ *de.inode* ∈ 0 .. *InodeCount* − 1
BlockNum ≜ ∃ *i* ∈ 0 .. *sb.inodes_per_group* − 1 : (*InodeTable*[*i*].*blocks* > 0 ⇒ ∀ *j* ∈
0 .. *InodeTable*[*i*].*blocks* : *InodeTable*[*i*].*directAddr*[*j*] ∈ 0 .. *sb.blocks_per_group* − 1)
FreeInodeCountProp ≜ ∀ *i* ∈ *FreeInodeCount* : *inodeBitmap*[*i*] = "FREE"
FreeBlockCountProp ≜ ∀ *i* ∈ *FreeBlockCount* : *blockBitmap*[*i*] = "FREE"
ValidRecLen ≜ ∀ *i* ∈ 1 .. *lenlist* : *dirlist*[*i*].*rec_len* > 0
ValidNameLen ≜ ∀ *i* ∈ 1 .. *lenlist* : *dirlist*[*i*].*name_len* > 0
ValidFilename ≜ ∀ *i* ∈ 1 .. *lenlist* : *dirlist*[*i*].*filename* ≠ ""
max_file_size ≜ ∀ *i* ∈ 0 .. *sb.inodes_per_group* − 1 : *InodeTable*[*i*].*size* ≤ *sb.blocks_per_group* ∗ *sb.log_block_size*
AllInodesCovered ≜ □(∀ *i* ∈ 0 .. *sb.inodes_per_group* − 1 : *inodeBitmap*[*i*] ∈ {"FREE", "ALLOCATED"})
AllblocksCovered ≜ □(∀ *i* ∈ 0 .. *sb.blocks_per_group* − 1 : *blockBitmap*[*i*] ∈ {"FREE", "ALLOCATED"})
ValidMode ≜ □(∀ *i* ∈ 0 .. *sb.inodes_per_group* − 1 : *InodeTable*[*i*].*mode* ∈ {"ext2_reg", "ext2_dir", "ext2_symlink", "none"})
InodeAllocation ≜ □(∀ *i* ∈ 0 .. *sb.inodes_per_group* − 1 : *InodeTable*[*i*].*dtime* ∈ {0, −1})
end define ;

procedure *FindFreeInode*()**begin**
find:
while *in* < *sb.inodes_per_group* **do**
if (*inodeBitmap*[*in*] = "FREE") **then**
inodeBitmap[*in*] := "ALLOCATED" ;
inode_num := *in* ;
FreeInodeCount := *FreeInodeCount* \ {*in*} ;
goto *final* ;
else
in := *in* + 1 ;
end if ;
end while ;
final: **return** ;
end procedure ;

procedure *FindFreeBlock*()**begin**
find:

```
  while bl < sb.blocks_per_group do
    if (blockBitmap[bl] = "FREE") then
        blockBitmap[bl] := "ALLOCATED";
        block_num := bl;
        FreeBlockCount := FreeBlockCount \ {bl};
        goto final;
     else
         bl := bl + 1;
    end if ;
  end while ;
final:   return;
end procedure ;


procedure find_next_free_inode()
 begin
 valueupdate:
 in := BGDT[0].bg_inode_bitmap;
 find_next__free_inode:
  while in < sb.inodes_per_group do
    if (inodeBitmap[in] = "FREE") then
        BGDT[0].bg_inode_bitmap := in;
        change_inodetablevalue:
        BGDT[0].bg_inode_table := in;
        in := sb.inodes_per_group;
     else
         in := in + 1;
    end if ;
  end while ;
change_free_inode_count:
BGDT[0].bg_free_inode_count := BGDT[0].bg_free_inode_count − 1;
rstval: in := BGDT[0].bg_inode_bitmap;
final:
return;
end procedure ;

procedure find_next_free_block()
begin
blvalue: bl := BGDT[0].bg_block_bitmap;
find_next__free_block:
  while bl < sb.blocks_per_group do
    if (blockBitmap[bl] = "FREE") then
        BGDT[0].bg_block_bitmap := bl;
        bl := sb.blocks_per_group;
     else
         bl := bl + 1;
    end if ;
  end while ;
change_free_blockcount:
BGDT[0].bg_free_block_count := BGDT[0].bg_free_block_count − 1;
blnewvalue: bl := BGDT[0].bg_block_bitmap;
final:
return;
end procedure ;
```

```
procedure UpdateBGDT(command)
begin
find_next_free_inode : call find_next_free_inode() ;
find_next_free_block :
if command = "slink" then
            call find_next_free_block()
    elsif command = "mkdir" then
            call find_next_free_block()
   end if ;
final:
return ;
end procedure ;

procedure createDirEntry(name)begin
create:
     de.inode := inode_num ;
nanmelen:
     de.name_len := Len(name) ;
reclen:
     de.rec_len := ((8 + de.name_len + 3) * 4) ÷ 4 ;
filename:
     de.filename := name ;
back:
return ;
end procedure ;

procedure addDirEntry(name)
begin
create_dir_entry:
call createDirEntry(name) ;
addDirEntry:
dirlist := dirlist ∘ ⟨de⟩ ;
lenlist := lenlist + 1 ;
back:
return ;
end procedure ;

procedure UpdateInode(command)
begin
writeInode:
     if InodeTable[inode_num].dtime = 0 then
       InodeTable[inode_num].dtime := −1 ;
       Inode_NUM : InodeTable[inode_num].iNum := inode_num ;
       change_mode:
       if command = "slink" then
           InodeTable[inode_num].mode := "ext2_symlink" ;
        elsif command = "mkdir" then
           InodeTable[inode_num].mode := "ext2_dir" ;
        elsif command = "touch" then
           InodeTable[inode_num].mode := "ext2_reg" ;
        end if ;
       init_block:
       if command = "slink" then
           InodeTable[inode_num].blocks := 1 ;
        elsif command = "mkdir" then
           InodeTable[inode_num].blocks := 1 ;
        elsif command = "touch" then
```

```
            InodeTable[inode_num].blocks := 0 ;
        end if ;
        fstblock:
        if command = "slink" then
                InodeTable[inode_num].directAddr[0] := block_num ;
         elsif command = "mkdir" then
                InodeTable[inode_num].directAddr[0] := block_num ;
        end if ;
        change_uid: InodeTable[inode_num].uid := 1000 ;
        change_gid: InodeTable[inode_num].gid := 1000 ;
        change_size: InodeTable[inode_num].size := InodeTable[inode_num].blocks * sb.log_block_size ;
        change_access_time: InodeTable[inode_num].atime := 1 ;
        change_creation_time: InodeTable[inode_num].ctime := 1 ;
        change_modification_time: InodeTable[inode_num].mtime := 1 ;
        change_links_count: InodeTable[inode_num].links_count := 1 ;
        generation_type: InodeTable[inode_num].generation := 0 ;
         else
        print "Error: inode already in use" ;
      end if ;
goback: return ;

end procedure ;

procedure updateSuperblock(command)
begin
updatefreeinode:
sbins.free_inodes_count := sbins.free_inodes_count − 1 ;
updatefreeblocks:
if command = "slink" then
                sbins.free_blocks_count := sbins.free_blocks_count − 1 ;
        elsif   command = "mkdir" then
                sbins.free_blocks_count := sbins.free_blocks_count − 1 ;
       end if ;
goback:
return ;
end procedure ;

procedure CreateFile(command, name)
begin
  FindInode: call FindFreeInode() ;
  FindBlock :
   if command = "slink" then
          call FindFreeBlock()
    elsif command = "mkdir" then
          call FindFreeBlock()
   end if ;
   writeinode:  call UpdateInode(command)      ;
   update_BGDT: call UpdateBGDT(command) ;
   createdirentry:     call addDirEntry(name) ;
   updatesuperblock :     call updateSuperblock(command) ;
  goback:  return ;
end procedure ;


process MAIN = "Main Process"begin
symbolic_link:
call CreateFile("slink", "symlink") ;




regularfile:
call CreateFile("touch", "regfile") ;
directory:
call CreateFile("mkdir", "directory1") ;

end
process ;
```

Κώδικας createfile μορφή ascii

```
------EXTENDS Integers,TLC,Sequences,FiniteSets,inode --------

(* --algorithm Createfile

variables
        inodeBitmap = [i \in 0..sb.inodes_per_group-1 |-> "FREE"] ,
        (* inode bitmap, with each bit representing the allocation status of an inode *)
        blockBitmap = [i \in 0..sb.blocks_per_group-1 |-> "FREE"],
        block_num = -1,
        bl = 0,
        in = 0,
        FreeInodeCount = {i \in 0..sb.inodes_per_group-1 : inodeBitmap[i] = "FREE"},
        FreeBlockCount = {i \in 0..sb.blocks_per_group-1 : blockBitmap[i] = "FREE"},
        InodeTable = [i \in 0..sb.inodes_per_group-1|->Inode] ,
        de = dir_entry,
        lenlist = 0,
        dirlist = << >> ,
        BGDT=BlockGroupDescriptorTable,
        sbins=sb,
        inode_num = -1;

define
FreeInodeExistsInvariant == \E i \in 0..sb.inodes_per_group-1 : inodeBitmap[i] = "FREE"
InodeInRange == de.inode \in 0..InodeCount-1
BlockNum == \E i \in 0..sb.inodes_per_group-1 : (InodeTable[i].blocks > 0 => \A j \in
0..InodeTable[i].blocks: InodeTable[i].directAddr[j] \in 0..sb.blocks_per_group-1)
FreeInodeCountProp == \A i \in FreeInodeCount: inodeBitmap[i] = "FREE"
FreeBlockCountProp == \A i \in FreeBlockCount: blockBitmap[i] = "FREE"
ValidRecLen == \A i \in 1..lenlist :dirlist[i].rec_len > 0
ValidNameLen == \A i \in 1..lenlist : dirlist[i].name_len >0
ValidFilename == \A i \in 1..lenlist : dirlist[i].filename # ""
max_file_size == \A i \in 0..sb.inodes_per_group-1 : InodeTable[i].size <= sb.blocks_per_group *
sb.log_block_size
AllInodesCovered == [](\A i \in 0..sb.inodes_per_group-1 : inodeBitmap[i]\in
{"FREE","ALLOCATED"})
AllblocksCovered == [](\A i \in 0..sb.blocks_per_group-1 : blockBitmap[i]\in
{"FREE","ALLOCATED"})
ValidMode == [](\A i \in 0..sb.inodes_per_group-1 : InodeTable[i].mode \in {"ext2_reg", "ext2_dir",
"ext2_symlink","none"})
InodeAllocation == [](\A i \in 0..sb.inodes_per_group-1 : InodeTable[i].dtime \in {0, -1})
end define;

procedure FindFreeInode () begin
```

```
 find:
  while in < sb.inodes_per_group do
            if (inodeBitmap[in] = "FREE") then
            inodeBitmap[in] := "ALLOCATED";
            inode_num := in;
            FreeInodeCount := FreeInodeCount \ {in};
            goto final;
            else
            in:=in+1;
            end if;
  end while;
final: return ;
end procedure;

procedure FindFreeBlock () begin
 find:
  while bl < sb.blocks_per_group do
            if (blockBitmap[bl] = "FREE") then
            blockBitmap[bl] := "ALLOCATED";
            block_num := bl;
            FreeBlockCount := FreeBlockCount \ {bl};
            goto final;
            else
            bl:=bl+1;
            end if;
  end while;
final: return ;
end procedure;


procedure find_next_free_inode()
 begin
 valueupdate:
 in := BGDT[0].bg_inode_bitmap;
 find_next__free_inode:
  while in < sb.inodes_per_group do
            if (inodeBitmap[in] = "FREE") then
            BGDT[0].bg_inode_bitmap := in;
            change_inodetablevalue:
            BGDT[0].bg_inode_table := in;
            in := sb.inodes_per_group;
            else
            in:=in+1;
            end if;
  end while;
change_free_inode_count:
BGDT[0].bg_free_inode_count :=BGDT[0].bg_free_inode_count - 1;
rstval: in:=BGDT[0].bg_inode_bitmap;
```

```
final:
return;
end procedure;

procedure find_next_free_block()
begin
blvalue: bl:= BGDT[0].bg_block_bitmap;
find_next__free_block:
  while bl < sb.blocks_per_group do
            if (blockBitmap[bl] = "FREE") then
            BGDT[0].bg_block_bitmap := bl;
            bl := sb.blocks_per_group;
            else
            bl:=bl+1;
            end if;
  end while;
change_free_blockcount:
BGDT[0].bg_free_block_count := BGDT[0].bg_free_block_count - 1;
blnewvalue: bl:= BGDT[0].bg_block_bitmap;
final:
return;
end procedure;

procedure UpdateBGDT(command)
begin
find_next_free_inode : call find_next_free_inode();
find_next_free_block :
if command = "slink" then
            call find_next_free_block ()
   elsif command = "mkdir" then
            call find_next_free_block ()
   end if;
final:
return;
end procedure;

procedure createDirEntry (name) begin
create:
            de.inode := inode_num;
nanmelen:
            de.name_len := Len(name);
reclen:
            de.rec_len := ((8 + de.name_len + 3) * 4 )\div 4;
filename:
            de.filename := name;
back:
return;
end procedure;
```

```
procedure addDirEntry(name)
begin
create_dir_entry:
call createDirEntry(name);
addDirEntry:
dirlist := dirlist \o <<de>>;
lenlist := lenlist +1;
back:
return;
end procedure;

procedure UpdateInode (command)
begin
writeInode:
        if InodeTable[inode_num].dtime = 0 then
        InodeTable[inode_num].dtime := -1;
        Inode_NUM: InodeTable[inode_num].iNum := inode_num;
        change_mode:
        if command = "slink" then
        InodeTable[inode_num].mode := "ext2_symlink";
        elsif command = "mkdir" then
        InodeTable[inode_num].mode := "ext2_dir";
        elsif command = "touch" then
        InodeTable[inode_num].mode := "ext2_reg";
        end if;
        init_block:
        if command = "slink" then
        InodeTable[inode_num].blocks := 1;
        elsif command = "mkdir" then
        InodeTable[inode_num].blocks := 1;
        elsif command = "touch" then
        InodeTable[inode_num].blocks := 0;
        end if;
        fstblock:
        if command = "slink" then
        InodeTable[inode_num].directAddr[0] := block_num;
        elsif command = "mkdir" then
        InodeTable[inode_num].directAddr[0] := block_num;
        end if;
        change_uid: InodeTable[inode_num].uid := 1000;
        change_gid: InodeTable[inode_num].gid := 1000;
        change_size: InodeTable[inode_num].size :=
InodeTable[inode_num].blocks*sb.log_block_size;
        change_access_time: InodeTable[inode_num].atime := 1;
        change_creation_time: InodeTable[inode_num].ctime := 1;
        change_modification_time: InodeTable[inode_num].mtime := 1;
        change_links_count: InodeTable[inode_num].links_count := 1;
```

```
            generation_type: InodeTable[inode_num].generation := 0;
            else
            print "Error: inode already in use";
            end if;
goback: return;

end procedure;

procedure updateSuperblock(command)
begin
updatefreeinode:
sbins.free_inodes_count := sbins.free_inodes_count -1;
updatefreeblocks:
if command = "slink" then
            sbins.free_blocks_count := sbins.free_blocks_count -1;
            elsif command = "mkdir" then
            sbins.free_blocks_count := sbins.free_blocks_count -1;
            end if;
goback:
return;
end procedure;

procedure CreateFile(command, name)
begin
  FindInode: call FindFreeInode();
  FindBlock :
   if command = "slink" then
            call FindFreeBlock ()
   elsif command = "mkdir" then
            call FindFreeBlock ()
   end if;
   writeinode: call UpdateInode(command);
   update_BGDT: call UpdateBGDT(command);
   createdirentry: call addDirEntry(name);
   updatesuperblock : call updateSuperblock(command);
  goback: return;
end procedure;


process MAIN= "Main Process" begin
symbolic_link:
call CreateFile("slink","symlink");
regularfile:
call CreateFile("touch","regfile");
directory:
call CreateFile("mkdir","directory1");

end
```

process;

Κώδικας δομών ext2fs

MODULE *inode*

EXTENDS *Integers*, *TLC*, *Sequences*, *FiniteSets*
CONSTANTS *InodeCount*, *BlockCount*, *N*, *blockSize*

Inode $\triangleq$ [
iNum $\mapsto 0$, *Inode* number is equal to its index in the array
mode $\mapsto$ "none", File type and permission bits
uid $\mapsto 0$, User *ID* of the file's owner
size $\mapsto 0$, File size in bytes, initially set to 1024 bytes
atime $\mapsto 0$, Last time the file was accessed
ctime $\mapsto 0$, Last time the file was changed
mtime $\mapsto 0$, Last time the file was modified
dtime $\mapsto 0$, Last time the file was deleted
gid $\mapsto 0$, Group *ID* of the file's owner
links_count $\mapsto 0$, Number of hard links to the file
blocks $\mapsto 0$, Number of blocks used by the file
flags $\mapsto 0$, File flags
*osd*1 $\mapsto 0$, Operating system dependent field
directAddr $\mapsto [b \in 0 \,..\, 11 \mapsto -1]$, Direct block addresses,
initialized with -1 indicating that it's not pointing to any block
generation $\mapsto 0$, File version (used for *NFS*)
file_acl $\mapsto -1$, File access control list
dir_acl $\mapsto -1$ Directory
]

superblock data structure
sb $\triangleq$ [
inodes_count $\mapsto$ *InodeCount*, total number of inodes
blocks_count $\mapsto$ *BlockCount*, total number of blocks
r_blocks_count $\mapsto 0$, number of reserved blocks
free_blocks_count $\mapsto$ *BlockCount*, number of free blocks
free_inodes_count $\mapsto$ *InodeCount*, number of free inodes
first_data_block $\mapsto 0$, number of the first data block
log_block_size $\mapsto$ *blockSize*, block *size* $= 1024\langle$*log_block_size*
log_frag_size $\mapsto 1024$, fragment *size* $= 1024\langle$*log_frag_size*
blocks_per_group $\mapsto$ *BlockCount* $\div N$, number of blocks per block group
inodes_per_group $\mapsto$ *InodeCount* $\div N$, number of inodes per block group
first_ino $\mapsto 11$, In revision 0, the first non-reserved *inode* is fixed to 11
inode_size $\mapsto 128$, value indicating the size of the *inode* structure
block_group_nr $\mapsto 1$ ndicate the block group number hosting this superblock structure
]

$$
\begin{array}{l}
dir\_entry \triangleq [\\
\quad inode \mapsto 0, \quad inode \text{ number associated with the file} \\
\quad rec\_len \mapsto 0, \quad \text{length of the record} \\
\quad name\_len \mapsto 0, \quad \text{length of filename} \\
\quad filename \mapsto \text{""} \quad \text{name of the file} \\
] \\
BGD\_entry \triangleq [\\
\quad bg\_block\_bitmap \mapsto 0, \quad \text{the firts free data block in the block bitmap} \\
\quad bg\_inode\_bitmap \mapsto 0, \quad \text{the first free } inode \text{ in the } inode \text{ bitmap} \\
\quad bg\_inode\_table \mapsto 0, \quad \text{the first free } inode \text{ in the } inode \text{ table} \\
\quad bg\_free\_block\_count \mapsto sb.blocks\_per\_group, \quad \text{the count of the free blocks} \\
\quad bg\_free\_inode\_count \mapsto sb.inodes\_per\_group, \quad \text{the count of the free inodes} \\
\quad bg\_used\_dir\_count \mapsto 1, \quad \text{the count of used directories} \\
\quad bg\_pad \mapsto 0, \quad \text{used for padding the structure on a } 32bit \text{ boundary} \\
\quad bg\_reserved \mapsto 10 \quad \text{reserved for future purpose} \\
] \\
BlockGroupDescriptorTable \triangleq [i \in 0 \,..\, N-1 \mapsto BGD\_entry]
\end{array}
$$

Invariants

$$
\begin{array}{l}
InodeCountInvariant \triangleq sb.inodes\_count = InodeCount \\
BlockCountInvariant \triangleq sb.blocks\_count = BlockCount \\
FreeBlockCountInvariant \triangleq sb.free\_blocks\_count \leq BlockCount \\
FreeInodeCountInvariant \triangleq sb.free\_inodes\_count \leq InodeCount
\end{array}
$$

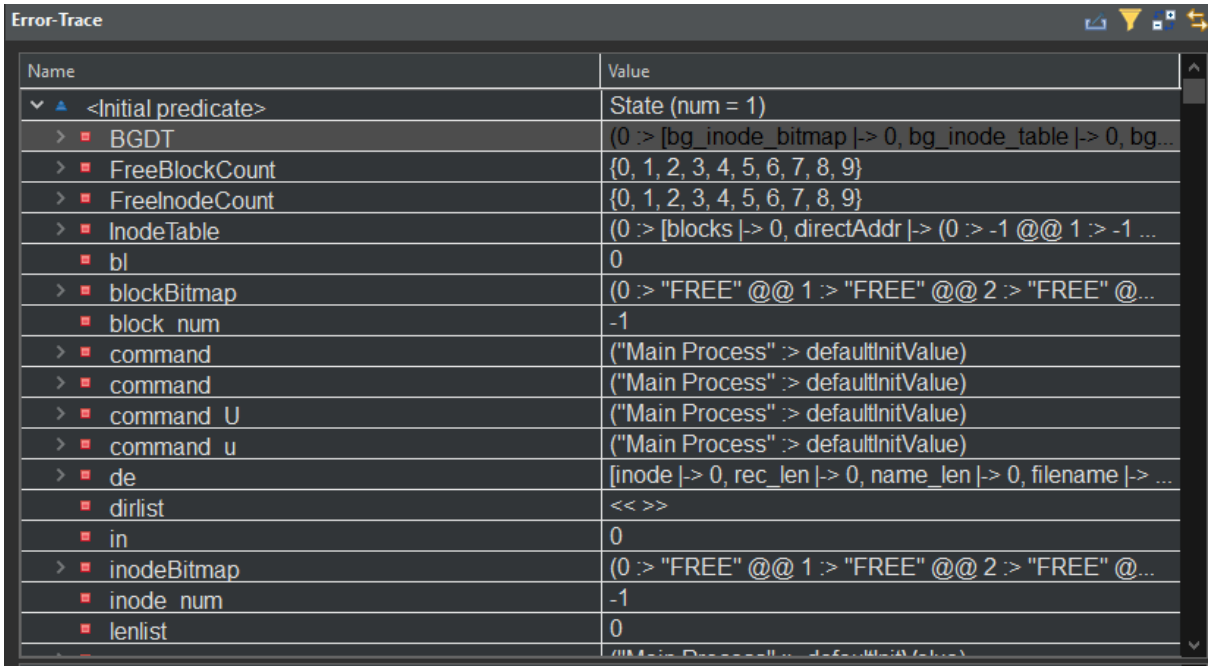
Κώδικας δομών extfs σε μορφή ascii

```
EXTENDS Integers,TLC,Sequences,FiniteSets
CONSTANTS InodeCount, BlockCount,N,blockSize
Inode == [
        iNum    |-> 0, (* Inode number is equal to its index in the array *)
        mode    |-> "none", (* File type and permission bits *)
        uid     |-> 0, (* User ID of the file's owner *)
        size    |-> 0, (* File size in bytes, initially set to 1024 bytes *)
        atime   |-> 0, (* Last time the file was accessed *)
        ctime   |-> 0, (* Last time the file was changed *)
        mtime   |-> 0, (* Last time the file was modified *)
        dtime   |-> 0, (* Last time the file was deleted *)
        gid     |-> 0, (* Group ID of the file's owner *)
        links_count  |-> 0, (* Number of hard links to the file *)
        blocks  |-> 0, (* Number of blocks used by the file *)
        flags   |-> 0, (* File flags *)
```

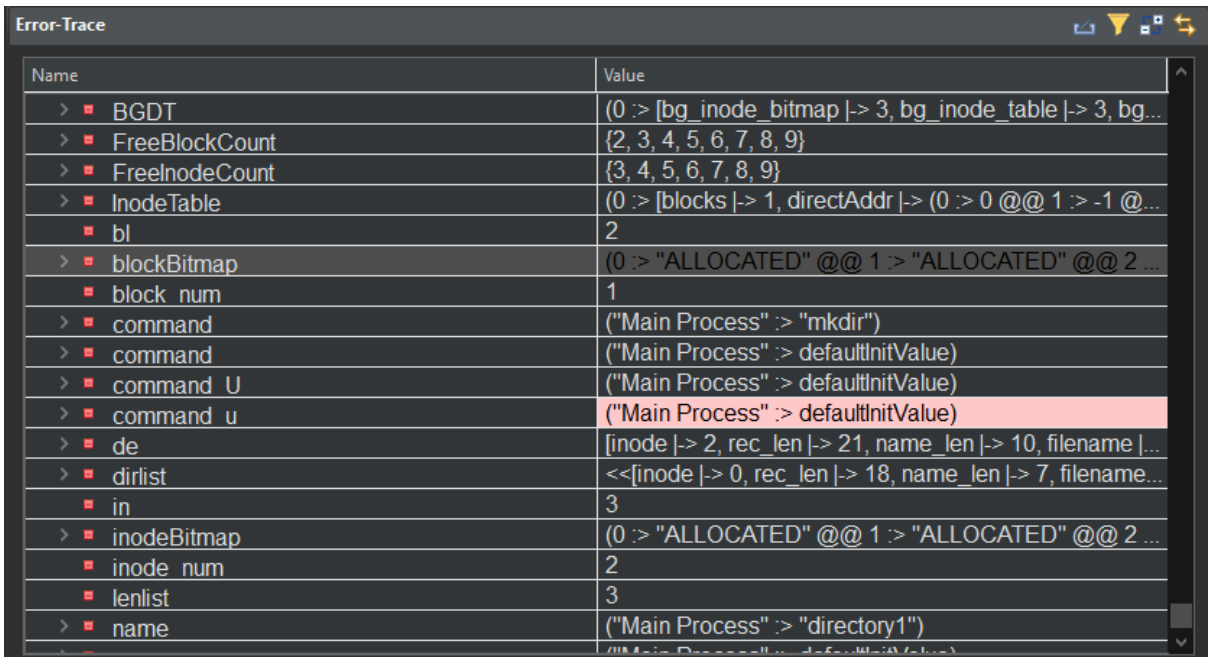
```
        osd1     |-> 0, (* Operating system dependent field *)
        directAddr  |-> [b \in 0..11 |-> -1],  (* Direct block addresses,
        initialized with -1 indicating that it's not pointing to any block *)
        generation  |-> 0, (* File version (used for NFS) *)
        file_acl |-> -1, (* File access control list *)
        dir_acl  |-> -1 (* Directory*)
        ]
(* superblock data structure *)
sb == [
        inodes_count   |-> InodeCount,  (* total number of inodes *)
        blocks_count   |-> BlockCount,  (* total number of blocks *)
        r_blocks_count |-> 0,              (* number of reserved blocks *)
        free_blocks_count |-> BlockCount, (* number of free blocks *)
        free_inodes_count |-> InodeCount, (* number of free inodes *)
        first_data_block |-> 0,            (* number of the first data block *)
        log_block_size  |-> blockSize,            (* block size = 1024 << log_block_size *)
        log_frag_size   |-> 1024,          (* fragment size = 1024 << log_frag_size *)
        blocks_per_group|-> BlockCount \div N,          (* number of blocks per block group *)
        inodes_per_group|-> InodeCount \div N,          (* number of inodes per block group *)
        first_ino          |-> 11,          (* In revision 0, the first non-reserved inode is fixed to 11 *)
        inode_size         |-> 128,         (*  value indicating the size of the inode structure *)
        block_group_nr  |-> 1              (* ndicate the block group number hosting this superblock
structure *)
        ]
dir_entry == [
        inode |-> 0, (* inode number associated with the file*)
        rec_len |-> 0, (*length of the record*)
        name_len |-> 0, (*length of filename*)
        filename |-> "" (*name of the file*)
        ]

BGD_entry == [
        bg_block_bitmap |-> 0,(*the firts free data block in the block bitmap*)
        bg_inode_bitmap |-> 0, (*the first free inode in the inode bitmap*)
        bg_inode_table |-> 0 , (*the first free inode in the inode table*)
        bg_free_block_count |-> sb.blocks_per_group, (*the count of the free blocks*)
        bg_free_inode_count |-> sb.inodes_per_group, (*the count of the free inodes*)
```

```
        bg_used_dir_count |-> 1, (*the count of used directories *)
        bg_pad |-> 0, (*used for padding the structure on a 32bit boundary*)
        bg_reserved |-> 10 (*reserved for future purpose*)
        ]
BlockGroupDescriptorTable ==  [i \in 0..N-1 |-> BGD_entry]
(* Invariants *)
InodeCountInvariant == sb.inodes_count = InodeCount
BlockCountInvariant == sb.blocks_count = BlockCount
FreeBlockCountInvariant == sb.free_blocks_count <= BlockCount
FreeInodeCountInvariant == sb.free_inodes_count <= InodeCount
```

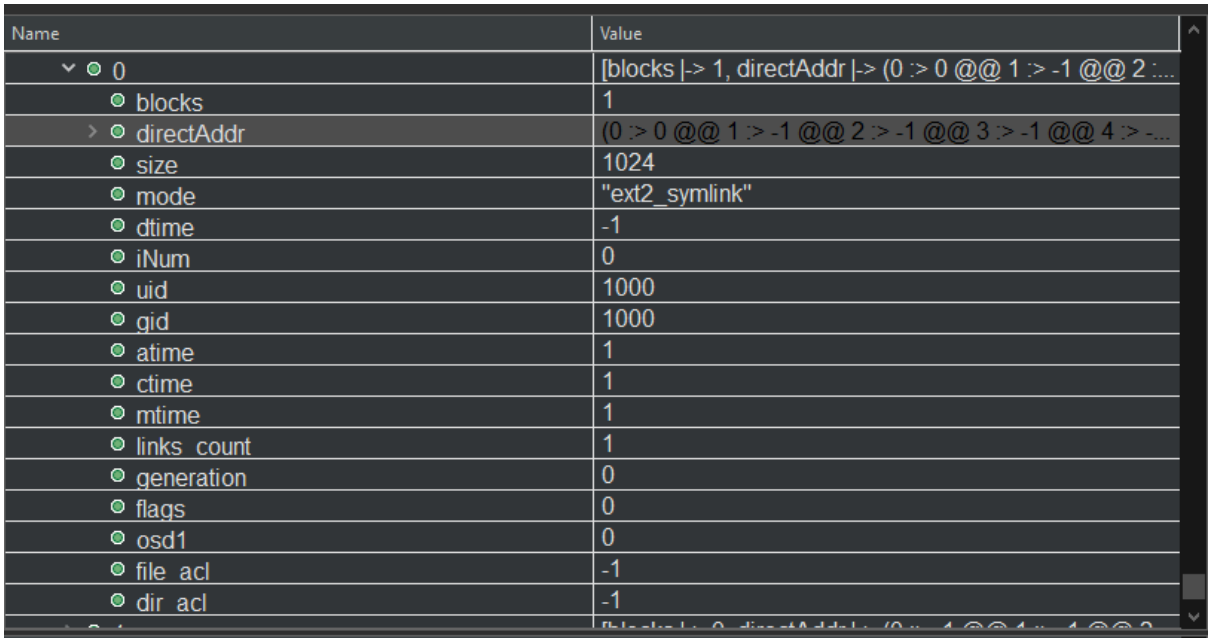
Παράρτημα ΙΙΙ



Εικόνα 10.1 : initial predicate

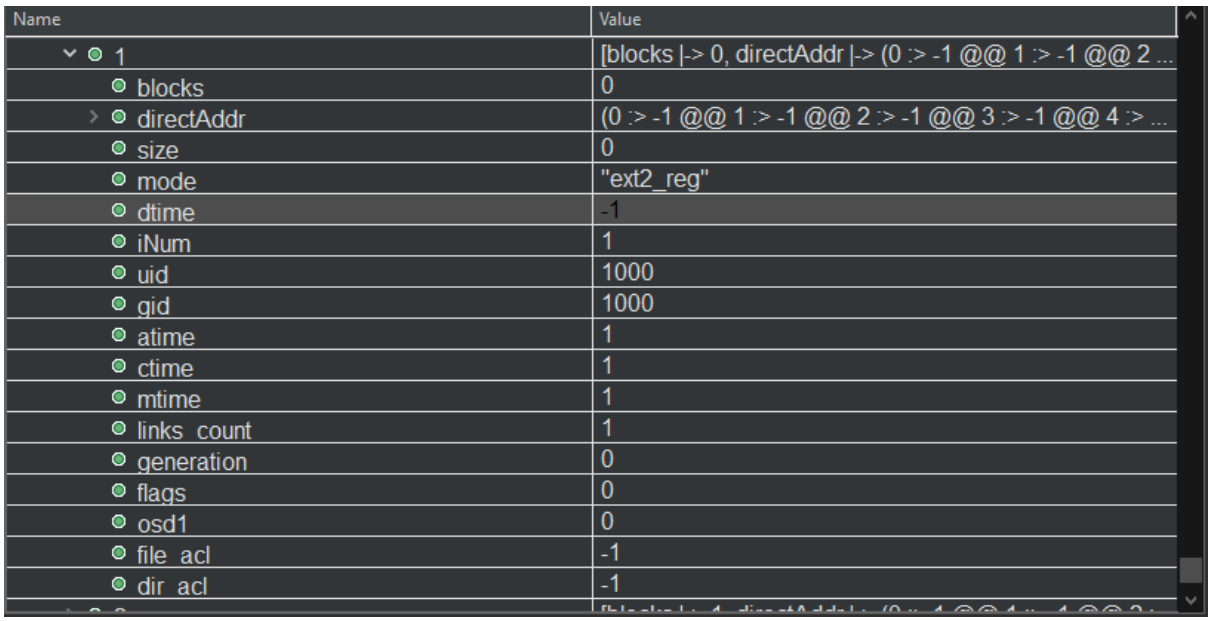


Εικόνα 10.2 : goback state



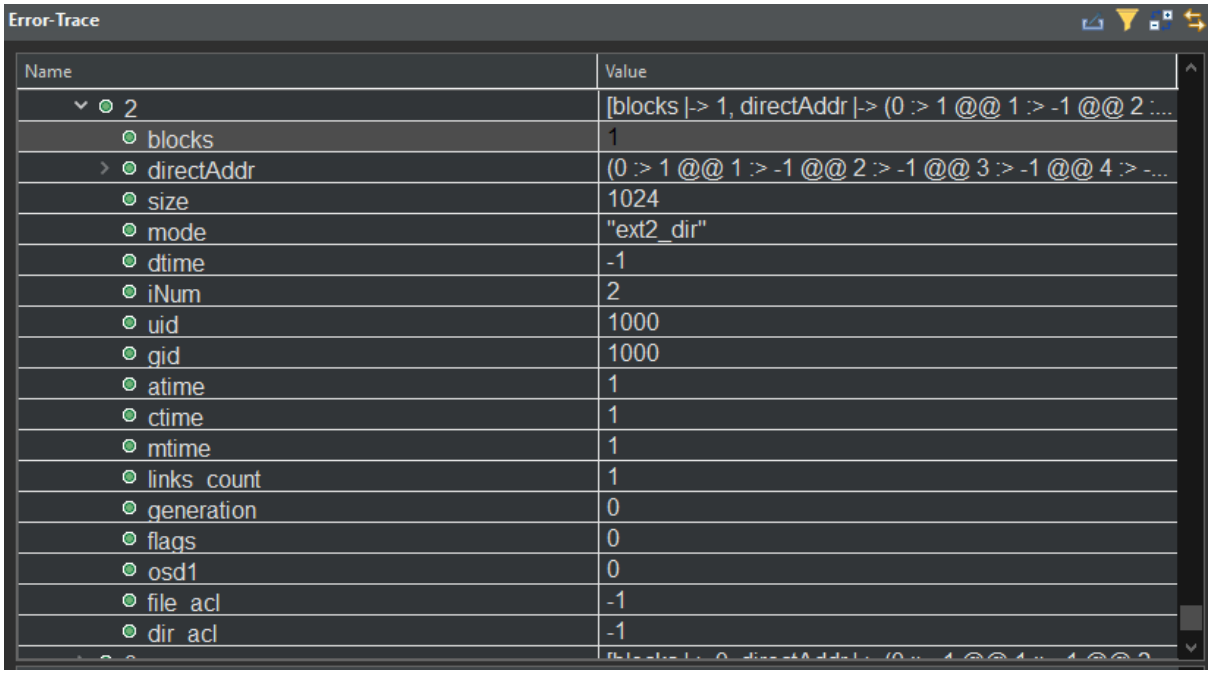
Name	Value
0	[blocks |-> 1, directAddr |-> (0 :> 0 @@ 1 :> -1 @@ 2 :...
blocks	1
directAddr	(0 :> 0 @@ 1 :> -1 @@ 2 :> -1 @@ 3 :> -1 @@ 4 :> -...
size	1024
mode	"ext2_symlink"
dtime	-1
iNum	0
uid	1000
gid	1000
atime	1
ctime	1
mtime	1
links_count	1
generation	0
flags	0
osd1	0
file_acl	-1
dir_acl	-1

Εικόνα 10.3 : Inode 0

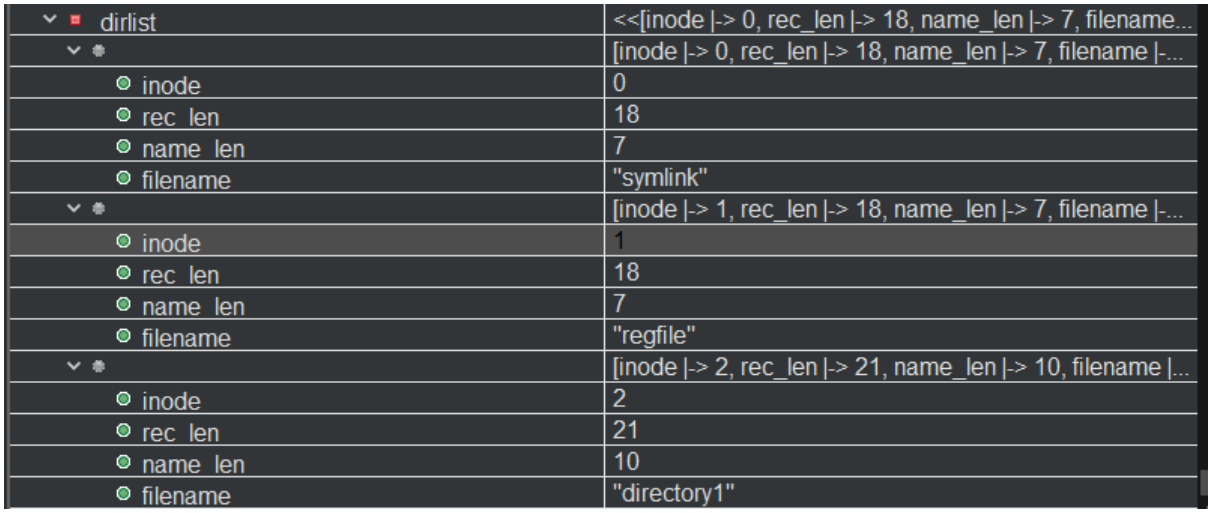


Name	Value
1	[blocks |-> 0, directAddr |-> (0 :> -1 @@ 1 :> -1 @@ 2 ...
blocks	0
directAddr	(0 :> -1 @@ 1 :> -1 @@ 2 :> -1 @@ 3 :> -1 @@ 4 :> ...
size	0
mode	"ext2_reg"
dtime	-1
iNum	1
uid	1000
gid	1000
atime	1
ctime	1
mtime	1
links_count	1
generation	0
flags	0
osd1	0
file_acl	-1
dir_acl	-1

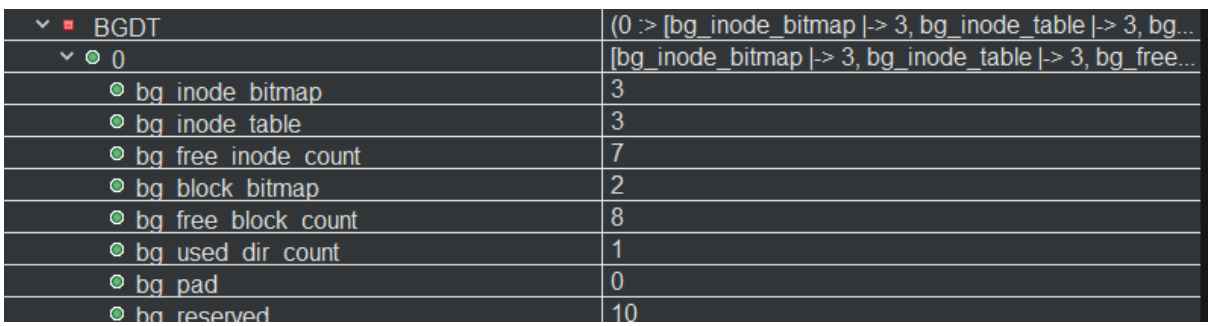
Εικόνα 10.4 : Inode 1



Εικόνα 10.5 : Inode 2



Εικόνα 10.6 : dirlist



Εικόνα 10.7 : Block Group Descriptor Table

sbins	[inodes_per_group |-> 10, blocks_per_group |-> 10, lo...
inodes_per_group	10
blocks_per_group	10
log_block_size	1024
free_inodes_count	47
free_blocks_count	48
inodes_count	50
blocks_count	50
r_blocks_count	0
first_data_block	0
log_frag_size	1024
first_ino	11
inode_size	128
block_group_nr	1

Εικόνα 10.8 : Superblock

Βιβλιογραφία

[1] L. Wirzenius, J. Oja, S. Stafford, και A. Weeks, *The Linux System Administrator's Guide*.

[2] M. J. Bach, *The Design of the UNIX Operating System*. People's Posts & Telecommunications Publishing House, 2003.

[3] A. S. Tanenbaum και A. S. Woodhull, *Operating systems: design and implementation*, τ. 2. Prentice Hall Englewood Cliffs, 1997.

[4] E. Nemeth, Επιμ., *UNIX and Linux system administration handbook*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2011.

[5] R. Card, 'Design and implementation of the second extended filesystem', στο *Procs. of the First Dutch International Symposium on Linux, 1994*, 1994.

[6] D. Poirier, 'The Second Extended File System - Internal Layout'.

[7] M. K. McKusick, W. N. Joy, S. J. Leffler, και R. S. Fabry, 'A fast file system for UNIX', *ACM Trans. Comput. Syst.*, τ. 2, τχ. 3, σσ. 181–197, Αυγούστου 1984, doi: 10.1145/989.990.

[8] 'Ext2 - OSDev Wiki'. https://wiki.osdev.org/Ext2 (ημερομηνία πρόσβασης 25 Μάρτιος 2023).

[9] R. S. Pressman, *Software engineering: a practitioner's approach*, 5th ed. στο McGraw-Hill series in computer science. Boston, Mass: McGraw Hill, 2000.

[10] 'Ch_27_Formal_spec.pdf'. Ημερομηνία πρόσβασης: 3 Απρίλιος 2023. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/WebChapters/PDF/Ch_27_Formal_spec.pdf

[11] J.-R. Abrial, 'Formal Methods: Theory Becoming Practice', *Formal Methods*.

[12] J. M. Wing, 'A specifier's introduction to formal methods', *Computer*, τ. 23, τχ. 9, σσ. 8–22, Σεπτεμβρίου 1990, doi: 10.1109/2.58215.

[13] M. V. Ruhela, 'Z Formal Specification Language - An Overview', *International Journal of Engineering Research*, τ. 1, τχ. 6, 2012.

[14] 'Formal methods'. https://dl.acm.org/doi/epdf/10.1145/242223.242257 (ημερομηνία πρόσβασης 6 Απρίλιος 2023).

[15] M. Batra, A. Malik, και D. M. Dave, 'FORMAL METHODS: BENEFITS, CHALLENGES AND FUTURE DIRECTION', 2010.

[16] 'The TLA+ Home Page'. https://lamport.azurewebsites.net/tla/tla.html (ημερομηνία πρόσβασης 25 Απρίλιος 2023).

[17] L. Lamport, *Specifying systems: the TLA+ language and tools for hardware and software engineers*. Boston: Addison-Wesley, 2003.

[18] 'Learn TLA+ — Learn TLA+'. https://learntla.com/index.html (ημερομηνία πρόσβασης 20 Απρίλιος 2023).

[19] 'c-manual.pdf'. Ημερομηνία πρόσβασης: 14 Απρίλιος 2023. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://lamport.azurewebsites.net/tla/c-manual.pdf

[20] 'PlusCal Tutorial - Introduction'. https://lamport.azurewebsites.net/tla/tutorial/intro.html (ημερομηνία πρόσβασης 1 Μάιος 2023).

[21] Y. Yu, P. Manolios, και L. Lamport, 'Model Checking TLA+ Specifications', στο *Correct Hardware Design and Verification Methods*, L. Pierre και T. Kropf, Επιμ., στο Lecture Notes in Computer Science, vol. 1703. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, σσ. 54–66. doi: 10.1007/3-540-48153-2_6.

[22] 'ext2 - fs/ext2 - Linux source code (v6.3.1) - Bootlin'. https://elixir.bootlin.com/linux/latest/source/fs/ext2 (ημερομηνία πρόσβασης 1 Μάιος 2023).