ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Αποδοτική και Αξιόπιστη Επικοινωνία με Κατανεμημένα Συστήματα Αποθήκευσης Δεδομένων μέσω Δικτυακής Συσκευής Αποθήκευσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Κ. Χατζηκωνσταντίνου

Επιβλέπων : Νεκτάριος Κοζύρης
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2008

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Αποδοτική και Αξιόπιστη Επικοινωνία με Κατανεμημένα Συστήματα Αποθήκευσης Δεδομένων μέσω Δικτυακής Συσκευής Αποθήκευσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Κ. Χατζηκωνσταντίνου

Επιβλέπων : Νεκτάριος Κοζύρης
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24η Μαρτίου 2008

…….....	……..	……
Νεκτάριος Κοζύρης	Κωνσταντίνος Σαγώνας	Νικόλαος Παπασπύρου
Επίκουρος Καθηγητής Ε.Μ.Π.	Αναπλ. Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2008

.....
Γεώργιος Κ. Χατζηκωνσταντίνου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Το vRAID είναι ένα κατανεμημένο αποθηκευτικό σύστημα που αναπτύσσεται από το εργαστήριο Υπολογιστικών Συστημάτων. Στην περίπτωση του vRAID η επικοινωνία ανάμεσα στους πελάτες και τα απομακρυσμένα αποθηκευτικά συστήματα δε γίνεται απευθείας. Αντίθετα, παρεμβάλλεται ένας κεντρικός ελεγκτής – controller, σκοπός του οποίου είναι να πραγματοποιεί τον απαραίτητο έλεγχο ανάμεσα στους πελάτες και τις αποθηκευτικές συσκευές. Όσο αφορά δε την επικοινωνία ανάμεσα στους πελάτες και τον controller, αυτή γίνεται με την χρήση του οδηγού NBD του Linux.

Σκοπός της διπλωματικής εργασίας είναι η επέκταση της παραπάνω αρχιτεκτονικής, ώστε να δίνεται στον πελάτη η δυνατότητα χρήσης περισσότερων του ενός ελεγκτή. Για να επιτευχθεί αυτό, ο οδηγός NBD τροποποιείται ώστε να μπορεί να διατηρεί και να ενημερώνει έναν πίνακα με τους διαθέσιμους controllers. Η διαχείριση αυτού του πίνακα γίνεται με την χρήση netlink sockets, οπότε και αναπτύσσεται το ανάλογο πρωτόκολλο για την επικοινωνία με τον πυρήνα. Τέλος, η επικοινωνία πελάτη-controller χωρίζεται σε δύο επίπεδα. Το πρώτο αφορά την ανταλλαγή μηνυμάτων για ανάγνωση και εγγραφή και το πρωτόκολλο που χρησιμοποιείται είναι αυτό του NBD. Το δεύτερο σχετίζεται με τα μηνύματα ελέγχου για την ανταλλαγή των οποίων δημιουργείται και το ανάλογο πρωτόκολλο.

Λέξεις Κλειδιά: Συστήματα Αποθήκευσης, Πλεονάζουσες Συστοιχίες Φθηνών Δίσκων, vRAID, Network Attached Storage, Storage Area Network, Συσκευή Block, Δικτυακή Συσκευή Block, Linux, Distributed Network Block Device

Abstract

vRAID is a storage area network that is under development by Computing Systems Laboratory. As regards vRAID, there is no direct communication between clients and the remote storage systems. Instead a controller is interposed between those two end points in order to carry out the necessary checks. Clients, on the other hand, communicate with the controller by using the Linux NBD (Network Block Device).

The purpose of this thesis is to extend the above architecture, so as to give client the ability to use more than one controller. In order to do that, NBD driver must maintain and update a list of available controllers. The management of this list is carried out by a user-space application which uses netlink sockets to communicate with the kernel. Finally, the client-controller communication is divided into two levels, data and control. The former includes read/write requests and the protocol used is that of NBD. The latter includes messages to check controllers' availability and handle client-controller connection and for that purpose an appropriate protocol is implemented.

Keywords: Storage Systems, RAID, vRAID, Network Attached Storage, Storage Area Network, Block Device, Network Block Device, Linux, Distributed Network Block Device

Πίνακας περιεχομένων

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

1 *Εισαγωγή*

Μία από τις κυριότερες λειτουργίες ενός υπολογιστικού συστήματος είναι η αποθήκευση και η ανάκληση πληροφοριών από τα διάφορα συστήματα αποθήκευσης. Ως σύστημα αποθήκευσης ορίζεται ο συνδυασμός των συσκευών μνήμης και των αλγορίθμων διαχείρισης και ελέγχου των αποθηκευμένων πληροφοριών. Ένα σύστημα αποθήκευσης θα πρέπει να προσφέρει μεγάλη ταχύτητα μεταφοράς των πληροφοριών και μηχανισμούς για την προστασία των πληροφοριών από σφάλματα και αστοχίες υλικού.

Καθώς το κόστος των δίσκων παραμένει μικρό, ήταν μονόδρομος η προσπάθεια δημιουργίας διατάξεων δίσκων με σκοπό την αύξηση της συνολικής προσφερόμενης ταχύτητας. Έτσι προέκυψαν τα συστήματα RAID, που συνδυάζουν αποτελεσματικά φθηνούς δίσκους, για τη δημιουργία γρηγορότερων και μεγαλύτερων αποθηκευτικών χώρων, προσφέροντας επίσης ασφάλεια στις σπάνιες, αλλά καταστροφικές αστοχίες υλικού.

Οι πλεονάζουσες συστοιχίες φθηνών δίσκων (**R**edundant **A**rrays of **I**nexpensive **D**isks) όπως ονομάστηκαν από τους ανθρώπους που τις σχεδίασαν, αποτελούν ομάδες δίσκων που χρησιμοποιούνται από κοινού, με σκοπό τη δημιουργία ενός αξιόπιστου αποθηκευτικού μέσου υψηλών επιδόσεων. Ο όρος RAID περιλαμβάνει όλα εκείνα τα αποθηκευτικά συστήματα που διαμοιράζουν τα δεδομένα και διατηρούν πολλαπλά αντίγραφα τους σε ένα σύνολο από δίσκους. Με τον τρόπο αυτό εξασφαλίζουν της ακεραιότητας των δεδομένων και βελτιώνουν την απόδοση των λειτουργιών ανάγνωσης και εγγραφής.

1.1 Σύστημα vRAID

Το vRAID είναι ένα κατανεμημένο σύστημα αποθήκευσης που αναπτύχθηκε από το Εργαστήριο Υπολογιστικών Συστημάτων. Ανήκει στην κατηγορία των SAN προσφέροντας ως επίπεδο επικοινωνίας αυτό του block. Σκοπός του είναι να μπορεί να αυξομειώνεται κατά

τις απαιτήσεις της εφαρμογής, ενώ παράλληλα θα προσφέρει αυξημένες δυνατότητες διαχείρισης επιτρέποντας την εύκολη δημιουργία αντιγράφων ασφαλείας.
Στην επικοινωνία ανάμεσα στους πελάτες και τα απομακρυσμένα αποθηκευτικά συστήματα παρεμβάλλεται ένας κεντρικός ελεγκτής – controller, σκοπός του οποίου είναι να πραγματοποιεί τον απαραίτητο έλεγχο ανάμεσα στους πελάτες και τις αποθηκευτικές συσκευές. Στη μέχρι τώρα υλοποίηση χρησιμοποιούνται για την επικοινωνία των πελατών με τον controller διεπαφές που υποστηρίζονται από τα υπάρχοντα λειτουργικά συστήματα, το Linux και το FreeBSD, όπως το NBD και το Ggate ή το AoE. Οι διεπαφές αυτές κάνουν χρήση της υπάρχουσας υποδομής δικτύωσης Ethernet, περιορίζοντας σημαντικά το κόστος υλοποίησης, αφού δεν απαιτούν την ύπαρξη εξειδικευμένης τεχνολογίας δικτύωσης.

1.2 Αντικείμενο Διπλωματικής

Στην παρούσα διπλωματική εξετάζεται η περίπτωση χρήσης της συσκευής NBD για την επικοινωνία πελάτη-controller. Ο μόνος περιορισμός που υπάρχει στην περίπτωση αυτή αφορά την αποκλειστική χρήση ενός μόνο controller από τον κάθε πελάτη. Έτσι η συσκευή NBD επεκτείνεται ώστε να παρέχει σε κάθε πελάτη τη δυνατότητα χρήσης περισσότερων του ενός controllers.
Η συσκευή που υλοποιείται στα πλαίσια της διπλωματικής ονομάζεται vNBD και θα πρέπει να μπορεί να διατηρεί και να ενημερώνει έναν πίνακα με τους διαθέσιμους controllers, στους οποίους θα προωθούνται οι αιτήσεις ανάγνωσης και εγγραφής δεδομένων. Επιπλέον, θα πρέπει να μπορεί να ελέγχει πότε κάποιος από τους controllers βρίσκεται εκτός λειτουργίας ώστε να σταματά να τον χρησιμοποιεί.

1.3 Οργάνωση Κειμένου

Εργασίες σχετικές με την υλοποίηση συσκευών που έχουν βασιστεί στη συσκευή NBD παρουσιάζονται στο κεφάλαιο 2. Στο κεφάλαιο 3 περιγράφονται ορισμένα γενικά χαρακτηριστικά του λειτουργικού συστήματος Linux και γίνεται μία εκτενής περιγραφή της διεπαφής που προσφέρεται από το συγκεκριμένο λειτουργικό για την ανάπτυξη οδηγών για συσκευές αποθήκευσης. Τα συστήματα RAID και οι λεπτομέρειες του τρόπου λειτουργίας του συστήματος vRAID δίνονται στο κεφάλαιο 4. Στο κεφάλαιο 5 αναφέρονται οι προδιαγραφές λειτουργίας της συσκευής vNBD και περιγράφονται οι περιπτώσεις χρήσης της, ενώ στο κεφάλαιο 6 παρουσιάζονται οι λεπτομέρειες υλοποίησης. Τέλος, στο κεφάλαιο 7 γίνεται η αξιολόγηση της επίδοσης της συσκευής και στο κεφάλαιο **8** γίνεται αναφορά στις βιβλιογραφικές πηγές.

2 *Σχετικές Εργασίες*

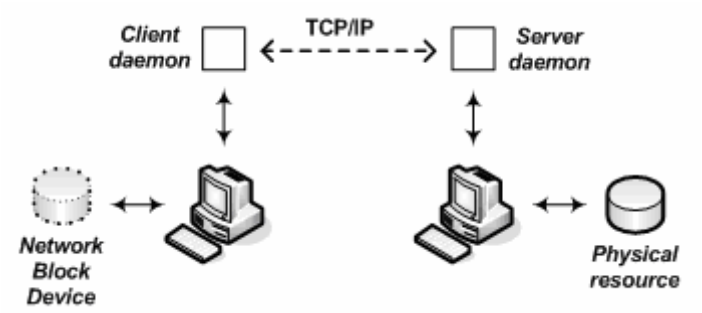
Η συσκευή NBD (Network Block Device) που υπάρχει στο Linux, έχει αποτελέσει τη βάση πολλών άλλων δικτυακών συσκευών block, κάθε μία από τις οποίες εισάγει μία σειρά από νέα χαρακτηριστικά, ανάλογα με την εφαρμογή για την οποία προορίζεται. Στο κεφάλαιο αυτό περιγράφεται αρχικά ο τρόπος λειτουργίας της συσκευής NBD, ενώ στη συνέχεια αναφέρονται κάποιες υλοποιήσεις οδηγών βασισμένων στον οδηγό της συσκευής NBD.

2.1 Network Block Device (NBD)

Ο πυρήνας του Linux περιλαμβάνει τον οδηγό της συσκευής NBD από την έκδοση 2.1.101. Ο οδηγός αυτός δίνει τη δυνατότητα χρήσης ενός απομακρυσμένου υπολογιστή, που παίζει τον ρόλο του εξυπηρετητή, σαν μία από τις συσκευές block του υπολογιστή-πελάτη στον οποίο βρίσκεται ο οδηγός. Έτσι, κάθε φορά που ο υπολογιστής-πελάτης διαβάζει ή γράφει στο ειδικό αρχείο της συσκευής, οι αιτήσεις εγγραφής και ανάγνωσης προωθούνται μέσω του οδηγού στον server.

Για τη μεταφορά των δεδομένων χρησιμοποιούνται TCP/IP sockets, ενώ ο χώρος αποθήκευσης στην πλευρά του server μπορεί να είναι ένα οποιοδήποτε αρχείο από το τοπικό σύστημα, ακόμα και ένα ειδικό αρχείο συσκευής block. Στο σχήμα 2.1 φαίνεται ο τρόπος λειτουργίας του NBD. Στην πλευρά του client υπάρχει μία διεργασία χώρου χρήστη για τη διαχείριση της συσκευής (client-daemon), ενώ στην πλευρά του server ο server-daemon είναι υπεύθυνος για την εξυπηρέτηση των αιτήσεων εγγραφής/ανάγνωσης.

Ο τρόπος λειτουργίας του NBD και οι δυνατότητες που προσφέρει στους υπολογιστές-πελάτες τον κάνουν ιδανικό για συστήματα με μειωμένο αποθηκευτικό χώρο αλλά ακόμα και για τερματικά που δε διαθέτουν δίσκο.



Σχήμα 2.1 – Λειτουργία της συσκευής *Network Block Device*

2.2 Enhanced Network Block Device (ENBD)

Η συσκευή ENBD υλοποιήθηκε από τον Peter T. Breuer με σκοπό να επεκτείνει τη λειτουργικότητα της συσκευής NBD, προσφέροντας επιπλέον στοιχεία που παρέχουν ασφάλεια στη μεταφορά δεδομένων και μεγαλύτερη αξιοπιστία στο σύστημα.

Ως πρωτόκολλο επικοινωνίας χρησιμοποιείται το Secure Sockets Layer (SSL). Πρόκειται για ένα πρωτόκολλο που λειτουργεί πριν το TCP/IP και μετά τις εφαρμογές υψηλού επιπέδου και χρησιμοποιεί μεθόδους κρυπτογράφησης των δεδομένων που ανταλλάσσονται, εγκαθιδρύοντας μία ασφαλή σύνδεση μέσω του διαδικτύου.

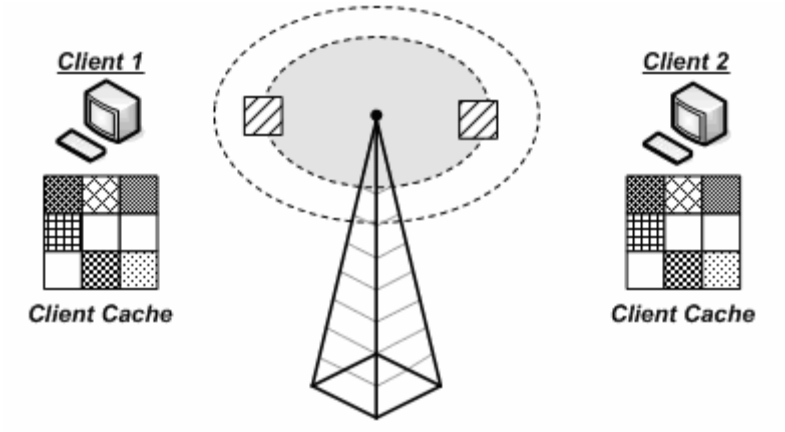
Επιπλέον, στην περίπτωση που για κάποιο λόγο η επικοινωνία client-server διακοπεί, η συσκευή αυτόματα ξεκινά τη διαδικασία δημιουργίας νέας σύνδεσης, ενώ δίνεται στον client η δυνατότητα χρήσεις απομακρυσμένων κλήσεων συστήματος ioctl.

2.3 Distributed Network Block Device (DNBD)

Η συσκευή DNBD [Tho06] είναι η μόνη που χρησιμοποιεί το πρωτόκολλο UDP για τη μεταφορά των δεδομένων και έχει σχεδιαστεί ειδικά για ασύρματα δίκτυα στα οποία τα τερματικά έχουν περιορισμένες δυνατότητες αποθήκευσης. Τα τερματικά μοιράζονται τις ίδιες συχνότητες, ενώ υπάρχει και ένα κεντρικό σημείο πρόσβασης με το οποίο μπορεί να επικοινωνεί ένας μόνο πελάτης τη φορά.

Για να βελτιωθεί η απόδοση του οδηγού δεν χρησιμοποιείται μηχανισμός κλειδωμάτων και κατά συνέπεια επιτρέπονται μόνο αιτήσεις ανάγνωσης. Κάθε φορά που κάποιο τερματικό-πελάτης ζητά ένα block για ανάγνωση, ο εξυπηρετητής το αποστέλλει σε όλα τα τερματικά (σχήμα 2.2), υποθέτοντας ότι είναι πολύ πιθανό οι υπόλοιποι πελάτες να ζητήσουν το ίδιο block δεδομένων. Επιπλέον, όλα τα τερματικά αποθηκεύουν τα δεδομένα σε κρυφές μνήμες (cashes).

Καθώς μπορεί να υπάρχουν περισσότερα από ένα σημεία πρόσβασης, ο οδηγός της συσκευής



Σχήμα 2.2 – Λειτουργία της συσκευής *Distributed Network Block Device.* Τα δεδομένα αποστέλλονται σε όλα τα τερματικά ανεξάρτητα από το ποιο έχει στείλει την αίτηση για ανάγνωση

παρέχει στα τερματικά τη δυνατότητα εντοπισμού και επιλογής του πιο αξιόπιστου σημείου πρόσβασης στα δεδομένα.

2.4 General Network Block Device (GeNBD)

Η συσκευή GeNBD [AKK04] αποτελείται από ανεξάρτητα μεταξύ τους στρώματα. Ο σχεδιασμός σε πολλαπλά στρώματα επιτρέπει την υποστήριξη διαφορετικών τύπων δικτύων και την εφαρμογή πολιτικής χρονοδρομολόγησης Ε/Ε ανάλογα με το δικτυακό περιβάλλον που χρησιμοποιείται.

Ο οδηγός της συσκευής είναι υπεύθυνος για ένα σύνολο από συσκευές block και παρέχει μια διεπαφή στα διάφορα δικτυακά στρώματα ώστε να καταχωρηθούν σε αυτόν και να συσχετιστούν με μία συγκεκριμένη συσκευή.

Πιο συγκεκριμένα ο οδηγός της συσκευής GeNBD υποστηρίζει την επικοινωνία με το απομακρυσμένο σύστημα αποθήκευσης για δίκτυα TCP/IP, για δίκτυα Myrinet και τέλος για δίκτυα SCI.

3 *Οδηγοί Συσκευών Αποθήκευσης στο Linux*

Στο κεφάλαιο αυτό παρουσιάζονται ορισμένα γενικά χαρακτηριστικά του λειτουργικού συστήματος Linux και γίνεται μία εκτενής περιγραφή της διεπαφής που προσφέρεται από το συγκεκριμένο λειτουργικό για την ανάπτυξη οδηγών για συσκευές αποθήκευσης (block drivers).

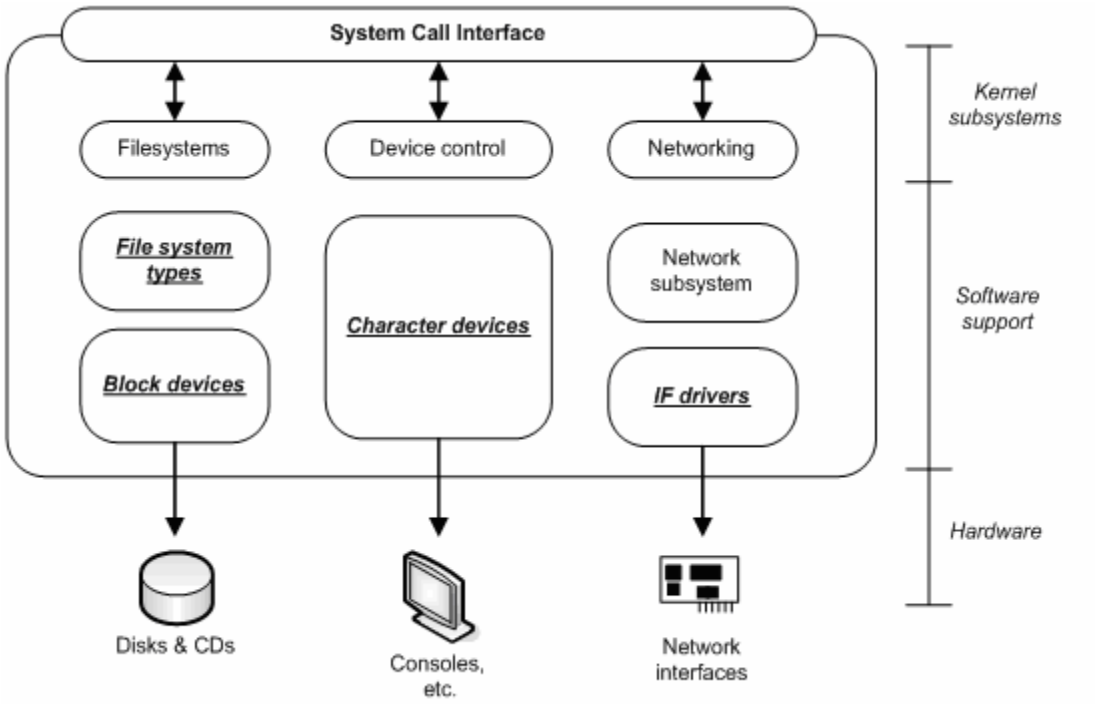
3.1 Το Λειτουργικό Σύστημα Linux

Το Linux είναι ένα σύγχρονο λειτουργικό σύστημα (Λ.Σ.) ανοικτού κώδικα, που βασίζεται στις βασικές σχεδιαστικές αρχές του Unix. Αναπτύχθηκε αρχικά από τον Φιλανδό Linus Torvalds το 1991 για τον επεξεργαστή i386 της Intel αλλά έχει μεταφερθεί μέχρι σήμερα σε πολλές διαφορετικές αρχιτεκτονικές.

3.1.1 Γενικά Χαρακτηριστικά του Linux

Όπως και τα περισσότερα λειτουργικά συστήματα που έχουν βασιστεί στο Unix, έτσι και το Linux είναι ένα μονολιθικό Λ.Σ. δηλαδή, οι περισσότερες λειτουργίες του υλοποιούνται σε ένα πρόγραμμα, τον πυρήνα, το οποίο εκτελείται ολόκληρο σε έναν ενιαίο χώρο διευθύνσεων. Ο πυρήνας του Linux αναλαμβάνει τη διαχείριση της κεντρικής μονάδας επεξεργασίας (CPU), την χρονοδρομολόγηση των διεργασιών, τη διαχείριση της μνήμης, τη διαχείριση των συσκευών εισόδου/εξόδου, τη διαχείριση του συστήματος αρχείων καθώς και τη διαδικτύωση. Επιπλέον στον πυρήνα βρίσκονται και οι οδηγοί συσκευών (device drivers), που επιτρέπουν τον έλεγχο των διαφόρων συσκευών του υλικού.

Ένα από τα μεγαλύτερα πλεονεκτήματα του Linux είναι ότι δίνει στον χρήστη τη δυνατότητα να επεκτείνει το σύνολο των λειτουργιών που προσφέρονται από τον πυρήνα χωρίς να απαιτείται επανεκκίνηση του συστήματος. Αυτό μπορεί να επιτευχθεί με τη δυναμική



Σχήμα 3.1 – Τα επίπεδα αφαίρεσης που προηγούνται της διεπαφής των κλήσεων συστήματος (System call interface). Τα στοιχεία που δίνονται υπογραμμισμένα μπορούν να προστεθούν ή να αφαιρεθούν δυναμικά ως modules.

εισαγωγή τμημάτων κώδικα στον πυρήνα τα οποία ονομάζονται modules και διακρίνονται σε κατηγορίες ή κλάσεις (classes) ανάλογα με το είδος της λειτουργίας τους. Μία από αυτές τις κατηγορίες είναι και οι οδηγοί συσκευών, γεγονός που σημαίνει ότι ένας οδηγός μπορεί να «φορτώνεται» δυναμικά όταν απαιτείται η χρήση της συσκευής στην οποία αντιστοιχεί και να «αφαιρείται» όταν πλέον δεν χρειάζεται. Χωρίς τα modules θα οδηγούμασταν σε μεγάλους σε μέγεθος πυρήνες, καθώς κάθε νέα λειτουργία θα έπρεπε να ενσωματώνεται στον μονολιθικό πυρήνα.

3.1.2 Οδηγοί Συσκευών

Ένα σημαντικό κομμάτι του πυρήνα είναι οι οδηγοί συσκευών οι οποίοι δίνουν στο λειτουργικό τη δυνατότητα να επικοινωνεί και να αλληλεπιδρά με το υλικό (hardware) του υπολογιστή. Καθώς κάθε συσκευή έχει κάποια ιδιαίτερα χαρακτηριστικά, οι οδηγοί λειτουργούν ως ένα ενδιάμεσο επίπεδο που συνδέει τη σαφώς ορισμένη διεπαφή που προσφέρεται από το Λ.Σ. με τις ιδιαίτερες για κάθε συσκευή μεθόδους. Το σχήμα 3.1 δείχνει την ενσωμάτωση διαφόρων συσκευών στο Λ.Σ. και τα διάφορα επίπεδα αφαίρεσης που προηγούνται της διεπαφής που προσφέρεται στον χρήση και η οποία του επιτρέπει την έμμεση πρόσβαση στις συσκευές μέσω των κλήσεων συστήματος (system calls).

Η ανάπτυξη κώδικα για οδηγούς συσκευών παρουσιάζει ορισμένα ιδιαίτερα χαρακτηριστικά που την διαφοροποιούν από την ανάπτυξη απλών εφαρμογών. Σε αντίθεση με κώδικα που εκτελείται ως μία διεργασία κάτω από το Λ.Σ., όπου πιθανή δυσλειτουργία οδηγεί στον τερματισμό της διεργασίας χωρίς να επηρεάζονται οι υπόλοιπες, εάν δυσλειτουργήσει κώδικας του πυρήνα, οι συνέπειες μπορεί να είναι από το να «κολλήσει» ο υπολογιστής έως το να υπάρξει σοβαρή βλάβη στο σύστημα αρχείων ή σε συσκευές υλικού. Κατά συνέπεια, απαιτείται γνώση του τρόπου οργάνωσης της μνήμης, της διαχείρισης των διεργασιών του πυρήνα, του μηχανισμού των κλειδωμάτων κ.τ.λ.. Επιπλέον, όταν κάποιος γράφει έναν οδηγό θα πρέπει να γνωρίζει σε βάθος τις λεπτομέρειες της αντίστοιχης συσκευής, έτσι ώστε να πετύχει τη μέγιστη απόδοση εκμεταλλευόμενος όλες τις δυνατότητες που αυτή παρέχει.

3.1.3 Αρχεία Συσκευών

Το Linux χρησιμοποιεί ένα ιεραρχικό σύστημα αρχείων (hierarchical file system) με δομή δέντρου με τα φύλλα να αντιστοιχούν στα αρχεία και τους υπόλοιπους κόμβους στους καταλόγους. Ολόκληρο το Λ.Σ. είναι δομημένο γύρω από αυτό το σύστημα αρχείων ενώ ακόμα και η προσπέλαση των περισσότερων συσκευών γίνεται μέσω ειδικών κόμβων που ονομάζονται «αρχεία συσκευών» (device files). Όπως και για τα συνήθη αρχεία, έτσι και για τα αρχεία συσκευών μπορούν να χρησιμοποιηθούν οι πρωτογενείς κλήσεις χειρισμού αρχείων (π.χ. open, close, read, write), με τη διαφορά ότι οι λειτουργίες αυτές μεταφράζονται σε ειδικές αιτήσεις και ο πυρήνας τις αντιστοιχεί σε κλήσεις στον κώδικα του οδηγού της συσκευής.

Συνήθως τα αρχεία συσκευών, λόγω της διαφορετικής τους ιδιότητας να είναι πύλες πρόσβασης σε συσκευές, βρίσκονται στον κατάλογο /dev χωρίς όμως αυτό να είναι δεσμευτικό. Επιπλέον, χαρακτηρίζονται από τον τύπο της συσκευής (συσκευή block ή συσκευή χαρακτήρων) ενώ διαθέτουν και ένα ζεύγος αριθμών οι οποίοι προσδιορίζουν μοναδικά την κάθε συσκευή. Πρόκειται για τους αριθμούς major και minor. Ο αριθμός major προσδιορίζει τον οδηγό που συνδέεται με τη συσκευή, ενώ ο αριθμός minor χρησιμοποιείται μόνο από τον οδηγό για να διακρίνει ανάμεσα στις συσκευές που χρησιμοποιούν τον ίδιο οδηγό.

3.2 Συσκευές Block

Μία συσκευή block είναι μία συσκευή (π.χ. ένας δίσκος) που μπορεί να φιλοξενήσει και να υποστηρίξει ένα σύστημα αρχείων. Στα περισσότερα Unix συστήματα μια τέτοια συσκευή μπορεί να διαχειριστεί λειτουργίες εισόδου/εξόδου (Ε/Ε) που αναφέρονται αποκλειστικά σε ένα ή περισσότερα blocks δεδομένων, τα οποία έχουν μέγεθος 512bytes (ή μία οποιαδήποτε

δύναμη του δύο). Το Linux παρόλα αυτά επιτρέπει την ανάγνωση και εγγραφή οποιουδήποτε μεγέθους δεδομένων κάθε φορά, αν και εσωτερικά διαχειρίζεται τα δεδομένα σε *τμήματα* (segments) που έχουν μέγεθος 512bytes. Όπως οι περισσότερες συσκευές έτσι και οι συσκευές block είναι προσβάσιμες μέσω ενός κόμβου στο σύστημα αρχείων και μέσω μίας διεπαφής, η οποία είναι αυστηρά ορισμένη από το Λ.Σ. .

Είναι σημαντικό να σημειωθεί ότι η απόδοση μίας συσκευής block επηρεάζει σε μεγάλο βαθμό την απόδοση ολόκληρου του συστήματος. Για τον λόγο αυτό το Λ.Σ. παρέχει μία σειρά από μηχανισμούς που διαχειρίζονται και βελτιώνουν σε μεγάλο βαθμό την πρόσβαση στις συσκευές block. Οι μηχανισμοί αυτοί περιλαμβάνουν μία σειρά από δομές δεδομένων, αλγορίθμους δρομολόγησης Ε/Ε και κρυφές μνήμες (cashes).

3.2.1 Ουρά Αιτήσεων Block

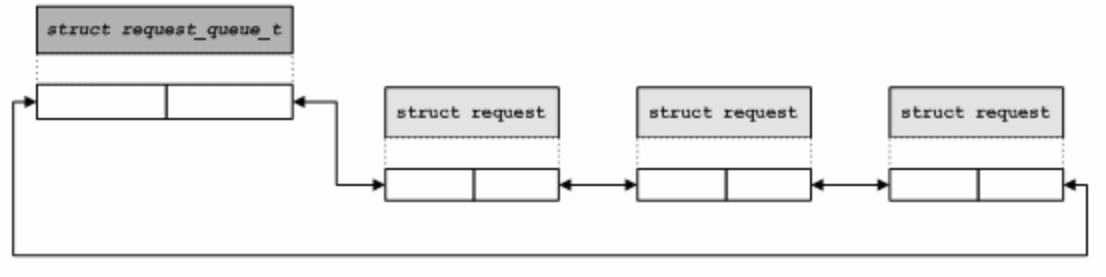
Η ουρά αιτήσεων (request queue) δεν είναι απλά μία ουρά που περιλαμβάνει αιτήσεις Ε/Ε block. Αποτελεί μία ιδιαίτερα πολύπλοκη δομή δεδομένων και είναι η κυριότερη δομή του στρώματος block.

Οι ουρές αιτήσεων περιέχουν διάφορες παραμέτρους που περιγράφουν το είδος των αιτήσεων που η συσκευή μπορεί να εξυπηρετήσει και όπως είναι φυσικό σε μία σωστά ορισμένη ουρά δε θα πρέπει να υπάρχουν αιτήσεις που η συσκευή δε μπορεί να διαχειριστεί. Επιπλέον, οι ουρές μπορούν να χρησιμοποιήσουν ένα σύνολο από δρομολογητές Ε/Ε που ονομάζονται elevators και αποτελούν ουσιαστικά τους αλγόριθμους βάσει των οποίων ορίζεται η σειρά επεξεργασίας των αιτήσεων.

Η δομή `request_queue_t` του πυρήνα, που χρησιμοποιείται για την αναπαράσταση της ουράς αιτήσεων αποτελείται από περισσότερα από 40 πεδία και ορίζεται στο `<linux/blkdev.h>`. Πρόκειται για μία διπλά συνδεδεμένη λίστα (σχήμα 3.2) που περιέχει δομές τύπου request όπως αυτές περιγράφονται στη συνέχεια ενώ η δομή περιλαμβάνει και δείκτες σε συναρτήσεις για τη δημιουργία, την ταξινόμηση και την συνένωση των αιτήσεων.

3.2.2 Αιτήσεις Block

Κάθε δομή `request` αναπαριστά μία αίτηση Ε/Ε block, παρά το γεγονός ότι μπορεί να έχει δημιουργηθεί μετά τη συνένωση ορισμένων ανεξάρτητων αιτήσεων. Αυτό γίνεται καθώς ο πυρήνας ομαδοποιεί αιτήσεις που αντιστοιχούν σε γειτονικούς τομείς στο δίσκο, χωρίς όμως ποτέ να συνδυάζει λειτουργίες εγγραφής και ανάγνωσης σε μία μοναδική δομή `request`. Επιπλέον, ο πυρήνας εξασφαλίζει ότι δε θα παραβιαστούν περιορισμοί που τίθενται από την



Σχήμα 3.2 – Οι ουρά αιτήσεων περιέχει ένα πεδίο τύπου list_head το οποίο αποτελεί την κεφαλή της διπλά συνδεδεμένης λίστας των αιτήσεων εγγραφής και ανάγνωσης.

ουρά αιτήσεων όπως το μέγιστο μέγεθος του τμήματος στο οποίο αντιστοιχεί μία αίτηση το οποίο δε θα πρέπει να ξεπερνά το μέγεθος μίας σελίδας (4096 bytes).
Εσωτερικά η δομή `request` υλοποιείται σαν μία συνδεδεμένη λίστα από δομές τύπου `bio` ενώ περιέχει μεταβλητές και πεδία σημαίες (flags) που σχετίζονται με τον δρομολογητή Ε/Ε.

3.2.3 Δομή bio

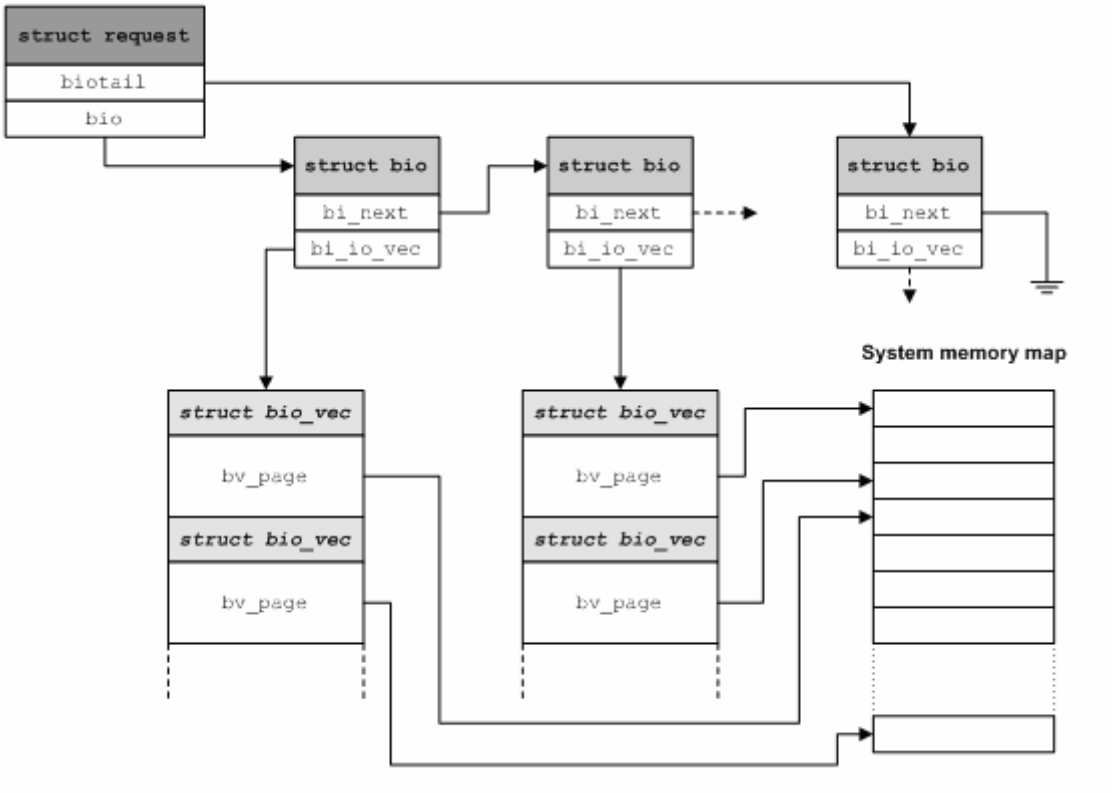
Η δομή `bio` είναι η μικρότερη δομική μονάδα μίας αίτησης. Όταν απαιτείται η μεταφορά ενός συνόλου τμημάτων δεδομένων από ή προς μία συσκευή block, ο πυρήνας δημιουργεί μία δομή αυτού του τύπου για να περιγράψει τη λειτουργία της μεταφοράς. Στη συνέχεια, η δομή `bio` είναι διαθέσιμη στο στρώμα block το οποίο την τοποθετεί είτε σε μία ήδη υπάρχουσα `request` είτε σε μία καινούργια. Μία δομή `bio` περιέχει όλες τις πληροφορίες που χρειάζεται ο οδηγός της συσκευής, ώστε να ολοκληρώσει την επεξεργασία της κάθε αίτησης, χωρίς να υπάρχει η ανάγκη για αναφορά στην διεργασία χώρου χρήστη που δημιούργησε αρχικά την αίτηση. Επιπλέον, η δομή αυτή, η οποία ορίζεται στο αρχείο `<linux/bio.h>`, περιέχει μία σειρά από πεδία τα οποία είναι ιδιαίτερα χρήσιμα :

- `sector_t bi_sector` : το πρώτο τμήμα (512 bytes) δεδομένων που θα πρέπει να μεταφερθεί για τη συγκεκριμένη δομή bio
- `unsigned int bi_size` : το μέγεθος των δεδομένων που πρόκειται να μεταφερθούν σε bytes. Συνήθως χρησιμοποιείται η μακροεντολή `bio_sectors(bio)` η οποία επιστρέφει το μέγεθος σε αριθμό τμημάτων των 512 bytes
- `unsigned long bi_flags` : ένα σύνολο από bits που περιγράφουν την κατάσταση της δομής

Παρόλ' αυτά το πιο σημαντικό πεδίο της δομής είναι ένας πίνακας που ονομάζεται `bi_io_vec`, και περιέχει στοιχεία του τύπου `bio_vec`. Τα πεδία του τύπου αυτού δίνονται στον πίνακα 3.1, ενώ στο σχήμα 3.3 φαίνεται ο τρόπος που συνδέονται μεταξύ τους οι δομές που αναφέρθηκαν. Όπως εύκολα μπορεί να δει κανείς από το σχήμα, όταν μία αίτηση

Τύπος Πεδίου	Όνομα Πεδίου	Περιγραφή
`struct page *`	`bv_page`	δείκτης στον περιγραφέα της σελίδας του τμήματος
`unsigned int`	`bv_len`	το μέγεθος του τμήματος σε bytes
`unsigned int`	`bv_offset`	το σημείο στο οποίο ξεκινά το τμήμα

Πίνακας 3.1 – Τα πεδία της δομής `bio_vec`



Σχήμα 3.3 – Η δομή `bio` και ο τρόπος που συνδέεται με τις υπόλοιπες δομές του στρώματος block

μετατρέπεται από το λειτουργικό σε μία δομή `bio`, αντιστοιχίζεται σε έναν πίνακα όπου για κάθε σελίδα που σχετίζεται με τη συγκεκριμένη αίτηση υπάρχει μία δομή τύπου bio_vec.

3.2.4 Δρομολογητής Ε/Ε

Παρά το γεγονός ότι οι οδηγοί συσκευών μπορούν να προσπελάσουν μόνο έναν τομέα του δίσκου την φορά, το στρώμα block δεν πραγματοποιεί μία λειτουργία Ε/Ε για τον κάθε τομέα. Κάτι τέτοιο θα οδηγούσε στη μικρή απόδοση του δίσκου, καθώς ο εντοπισμός της φυσικής θέσης του τομέα πάνω στην επιφάνεια του δίσκου απαιτεί αρκετό χρόνο. Αυτό που γίνεται στην πραγματικότητα είναι η ομαδοποίηση των λειτουργιών Ε/Ε που αναφέρονται σε γειτονικούς τομείς και η συνολική διαχείρισή τους, έτσι ώστε να μειωθεί ο μέσος αριθμός μετακινήσεων της κεφαλής του δίσκου.

Οι αναγνώσεις και οι εγγραφές δεδομένων μεταφράζονται εσωτερικά στον πυρήνα σε αιτήσεις συσκευής block, οι οποίες όμως δεν ικανοποιούνται από τον πυρήνα αμέσως μετά τη δημιουργία τους. Αντίθετα η κάθε αίτηση δρομολογείται έτσι ώστε να εκτελεστεί κάποια στιγμή από τον οδηγό, γεγονός που εισάγει μία χρονική καθυστέρηση στην εξυπηρέτηση της αίτησης. Αυτή ακριβώς η καθυστέρηση αποτελεί τον κεντρικό μηχανισμό βελτίωσης της απόδοσης των συσκευών block. Για κάθε αίτηση το στρώμα block καλεί τον δρομολογητή Ε/Ε για να προσδιοριστεί η ακριβής θέση της στην ουρά αιτήσεων. Οι αιτήσεις μάλιστα τοποθετούνται με τέτοιο τρόπο ώστε να είναι ταξινομημένες σύμφωνα με τον τομέα στον οποίο αναφέρονται. Έτσι η κεφαλή του δίσκου κινείται από τα εξωτερικά ίχνη (tracks) προς τα εσωτερικά ή αντίστροφα και όχι με τυχαίο τρόπο. Η κίνηση αυτή θυμίζει ανελκυστήρα γι΄αυτό και οι δρομολογητές Ε/Ε ονομάζονται διαφορετικά και elevators.

3.3 Διεπαφή Συσκευών Block

Στην παράγραφο που ακολουθεί περιγράφεται αναλυτικά η διεπαφή που προσφέρεται από το Linux για την ανάπτυξη οδηγών για συσκευές block.

3.3.1 Δήλωση της Συσκευής στον Πυρήνα

Το πρώτο πράγμα που πρέπει να κάνει κάθε συσκευή block είναι να συνδεθεί με τον πυρήνα. Η συνάρτηση που πρέπει να κληθεί για να επιτευχθεί αυτό είναι η `register_blkdev` :

```
int register_blkdev(unsigned int major, const char *name);
```

Τα ορίσματα που παίρνει είναι ο αριθμός major που θα χρησιμοποιείται από τη συσκευή και ένα αλφαριθμητικό που αντιστοιχεί στο όνομα της συσκευής το οποίο θα φαίνεται στο /proc/devices. Εάν ο αριθμός major είναι ίσος με μηδέν, ο πυρήνας δεσμεύει δυναμικά έναν ελεύθερο αριθμό major, τον οποίο και επιστρέφει. Ως συνήθως, σε περίπτωση σφάλματος επιστρέφεται μία αρνητική τιμή.

Η αντίστοιχη συνάρτηση για την αφαίρεση του οδηγού είναι η :

```
int unregister_blkdev(unsigned int major, const char *name);
```

Στην περίπτωση αυτή τα ορίσματα θα πρέπει να έχουν τις ίδιες ακριβώς τιμές με αυτές που είχαν κατά τη δήλωση της συσκευής, στην αντίθετη περίπτωση θα επιστραφεί μία αρνητική τιμή και η συσκευή δε θα αφαιρεθεί από τον πυρήνα.

Στον πυρήνα 2.6, η κλήση της `register_blkdev` είναι προαιρετική. Οι λειτουργίες που εκτελούνται από τη συνάρτηση αυτή περιορίστηκαν σταδιακά και πλέον οι μόνες ενέργειες που γίνονται κατά την κλήση της είναι η δυναμική δέσμευση ενός major αριθμού, αν κάτι τέτοιο απαιτείται, και η δημιουργία μίας καταχώρησης στο αρχείο /proc/devices.

Τύπος Πεδίου	Όνομα Πεδίου	Περιγραφή
`int`	`major`	ο αριθμός major της συσκευής
`struct block_device_operations *`	`fops`	συναρτήσεις που αντιστοιχούν στις λειτουργίες που αναλαμβάνει να επιτελέσει ο οδηγός συσκευής
`struct request_queue_t *`	`queue`	η ουρά αιτήσεων της συσκευής
`sector_t`	`capacity`	η χωρητικότητα της συσκευής σε τμήματα των 512 bytes
`void *`	`private_data`	ο δείκτης αυτός χρησιμοποιείται για να κρατούνται πληροφορίες για την κατάστασή της συσκευής ανάμεσα στις διάφορες κλήσεις συστήματος

Πίνακας 3.2 – Τα σημαντικότερα πεδία της δομής `gendisk`

3.3.2 Διεπαφή gendisk

Η δομή `gendisk` είναι η δομή που χρησιμοποιείται εσωτερικά από τον πυρήνα για να αναπαραστήσει μία συσκευή block. Τα σημαντικότερα πεδία της δομής δίνονται στον πίνακα 3.2. Ανάμεσα στα πεδία αυτά ξεχωρίζει το πεδίο fops το οποίο είναι ένας δείκτης σε μία δομή τύπου `block_device_operations` που περιέχει τις συναρτήσεις που αντιστοιχούν στις λειτουργίες που αναλαμβάνει να επιτελέσει ο οδηγός συσκευής (open, ioctl, release κτλ.). Οι συναρτήσεις αυτές ονομάζονται συχνά και «μέθοδοι» της συσκευής σύμφωνα με την ορολογία του αντικειμενοστρεφούς προγραμματισμού, θεωρώντας ότι ο πυρήνας ζητά από τον οδηγό συσκευής την εκτέλεση μίας λειτουργίας καλώντας την αντίστοιχη μέθοδο του «αντικειμένου» gendisk.

Ο οδηγός θα πρέπει να δεσμεύσει δυναμικά μία δομή gendisk όμως κάτι τέτοιο δε μπορεί να γίνει με απλή δέσμευση μνήμης, καθώς απαιτούνται ιδιαίτεροι μηχανισμοί αρχικοποίησης της δομής. Έτσι, χρησιμοποιείται η συνάρτηση :

```
struct gendisk *alloc_disk(int minors);
```

με το όρισμά της να αντιστοιχεί στο πλήθος των αριθμών minor που θα χρησιμοποιούνται από τη συσκευή.

Αντίθετα για την αποδέσμευσή της θα πρέπει να κληθεί η:

```
void del_gendisk(struct gendisk *gd);
```

Η δέσμευση όμως της δομής δε συνεπάγεται και ότι η συσκευή γίνεται διαθέσιμη στο σύστημα. Για να γίνει αυτό θα πρέπει μετά την αρχικοποίηση των πεδίων να κληθεί η συνάρτηση `add_disk` :

```
void add_disk(struct gendisk *gd);
```

Θα πρέπει εδώ να σημειωθεί ότι στην πραγματικότητα οι πρώτες αιτήσεις θα φτάσουν στη συσκευή πριν ακόμα η συνάρτηση `add_disk` επιστρέψει. Επομένως, ο οδηγός θα πρέπει να

έχει αρχικοποιηθεί και να είναι έτοιμος να επεξεργαστεί τις αιτήσεις πριν γίνει η κλήση της παραπάνω συνάρτησης.

3.3.3 Δημιουργία Ουράς Αιτήσεων

Η ουρά αιτήσεων αποτελεί μία ιδιαίτερα πολύπλοκη δομή η οποία θα πρέπει να δημιουργηθεί και να αρχικοποιηθεί από το στρώμα block. Η συνάρτηση που θα κληθεί από τον οδηγό για τη δέσμευση και την αρχικοποίηση της ουράς είναι η :

```
request_queue_t *blk_init_queue(request_fn_proc *req, spinlock_t *l);
```

Το πρώτο όρισμα είναι η συνάρτηση που θα διαχειρίζεται τις αιτήσεις. Η συνάρτηση αυτή καλείται από τον πυρήνα σε τακτά χρονικά διαστήματα και ουσιαστικά αποτελεί το τμήμα του κώδικα που είναι υπεύθυνο για την εξυπηρέτηση των αιτήσεων. Μία τέτοια συνάρτηση θα πρέπει να ορίζεται ως :

```
void req(request_queue_t *queue);
```

Το δεύτερο όρισμα της `blk_init_queue` είναι ένα κλειδί τύπου spinlock. Κάθε φορά που καλείται η μέθοδος `req` το κλειδί αυτό είναι δεσμευμένο από τον πυρήνα με αποτέλεσμα να μην επιτρέπεται η εισαγωγή νέων αιτήσεων στην ουρά.

Τέλος, το στρώμα block προσφέρει και μία συνάρτηση για την αποδέσμευση της ουράς αιτήσεων :

```
void blk_cleanup_queue(request_queue_t *queue);
```

3.3.4 Επεξεργασία Αιτήσεων

Το στρώμα block προσφέρει ένα μικρό σύνολο συναρτήσεων για την διαχείριση των αιτήσεων που βρίσκονται στην ουρά. Αρχικά θα πρέπει να κληθεί από τον οδηγό η `elv_next_request` :

```
struct request * elv_next_request(request_queue_t *queue);
```

η οποία επιστρέφει την επόμενη προς επεξεργασία αίτηση, όπως αυτή ορίζεται από τον δρομολογητή Ε/Ε, ενώ επιστρέφει NULL στην περίπτωση που η ουρά είναι άδεια. Αυτό που αξίζει να σημειωθεί είναι ότι η παραπάνω συνάρτηση δεν αφαιρεί την αίτηση από την ουρά, αλλά την σημειώνει ως ενεργή ώστε να αποτρέψει τον δρομολογητή από το να την συγχωνέψει με άλλες αιτήσεις. Η αφαίρεση της αίτησης από την ουρά γίνεται με κλήση στη συνάρτηση:

```
void blkdev_dequeue_request(struct request *req);
```

Αφού γίνει η επεξεργασία της αίτησης Ε/Ε, ο οδηγός θα πρέπει με κάποιο τρόπο να ενημέρωση το στρώμα block ότι η εξυπηρέτηση της αίτησης έχει ολοκληρωθεί. Πρόκειται για μία λειτουργία που γίνεται σε δύο βήματα.

Αρχικά καλείται η συνάρτηση :

`int end_that_request_first(struct request *req,int success,int count)`
η οποία ενημερώνει το στρώμα block ότι η αίτηση ολοκληρώθηκε με την μεταφορά `count` στον αριθμό τμημάτων. Η παράμετρος success παίρνει την τιμή 1 εάν η λειτουργία E/E ήταν επιτυχής και την τιμή 0 στην αντίθετη περίπτωση. Έαν δεν έχουν παρουσιαστεί σφάλματα, η προηγούμενη συνάρτηση επιστρέφει 0 και τώρα πλέον μπορεί να κληθεί η συνάρτηση για την αφαίρεση της αίτησης από την ουρά, εάν κάτι τέτοιο δεν έχει γίνει νωρίτερα.
Σε μία δεύτερη φάση ο οδηγός καλεί την :

```
void end_that_request_last(struct request *req);
```

με την οποία ενημερώνονται όσοι περιμένουν την ολοκλήρωση της συγκεκριμένης αίτησης ενώ παράλληλα η δομή `request` επιστρέφεται στο σύστημα.

3.3.5 Επεξεργασία Δομών bio

Όπως έχει περιγραφή και στην παράγραφο 3.2, κάθε δομή `request` περιέχει μία συνδεδεμένη λίστα δομών `bio`, οι οποίες με τη σειρά τους περιέχουν έναν πίνακα με στοιχεία τύπου `bio_vec`. Καθώς υπάρχουν περιπτώσεις κατά τις οποίες ένας οδηγός θα πρέπει να επεξεργαστεί την αίτηση μεταφέροντας το κάθε τμήμα δεδομένων χωριστά, ο πυρήνας παρέχει δύο μακροεντολές για την απλοποίηση της πρόσβασης στις επιμέρους δομές. Στη συνέχεια δίνεται ένα παράδειγμα στο οποίο φαίνεται η χρήση τον δύο αυτών μακροεντολών.

```
struct bio  *bio;
struct bio_vec    *bvec;
int          i;

rq_for_each_bio(bio,rq){
      bio_for_each_segment(bvec,bio,i){
      /* επεξεργασία του κάθε τμήματος χωριστά */
      }
}
```

4 *Συστήματα Αποθήκευσης*

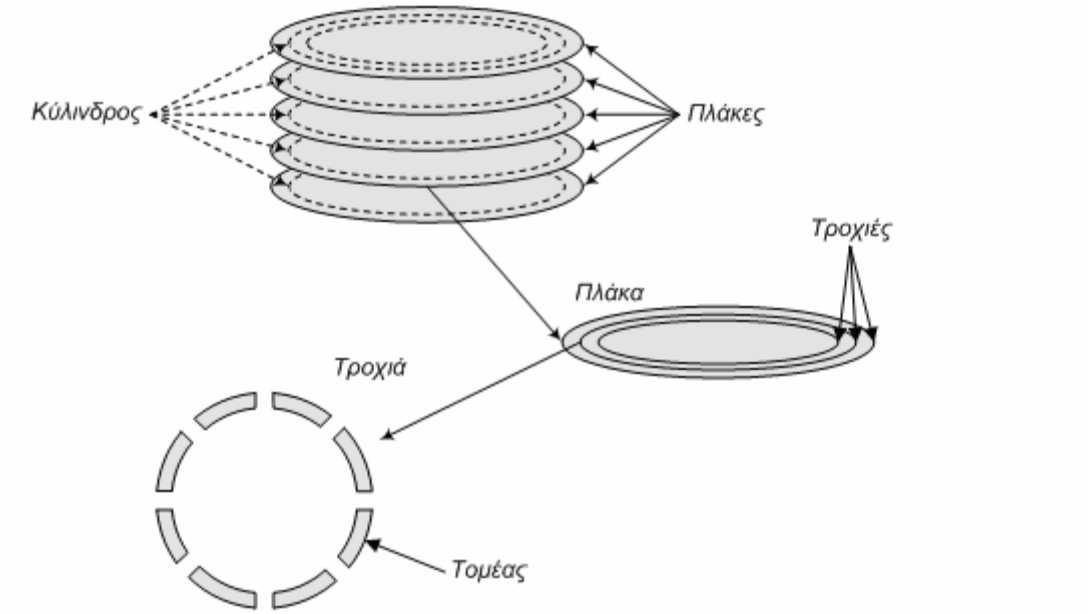
Ένα από τα βασικά υποσυστήματα ενός υπολογιστικού συστήματος είναι αυτό της αποθήκευσης. Κύριος εκπρόσωπός τους είναι ο σκληρός δίσκος, μια συσκευή που ήρθε να προσφέρει ασφαλή, στατική αποθήκευση δεδομένων. Με τα χρόνια όμως μετατράπηκε στον φτωχό συγγενή του επεξεργαστή, παραμένοντας αργός, χωρίς σημαντικές αλλαγές στην αρχιτεκτονική του και αποτελώντας στην ουσία τη σημαντικότερη αιτία καθυστέρησης στην επίδοση ενός υπολογιστικού συστήματος. Το μόνο που έχει επιτευχθεί είναι η συνεχής αύξηση του μεγέθους του. Έτσι είναι πλέον αδύνατο να τον αποχωριστούμε, αφού οι υπόλοιπες διαθέσιμες λύσεις αποθήκευσης υπολείπονται είτε σε μέγεθος, είτε σε ταχύτητα, είτε σε αξιοπιστία.

Ένα ακόμα δυνατό σημείο του σκληρού δίσκου, είναι το κόστος του, που παραμένει χαμηλό. Ήταν μονόδρομος λοιπόν η προσπάθεια δημιουργίας διατάξεων δίσκων με σκοπό την αύξηση της συνολικής προσφερόμενης ταχύτητας. Έτσι προέκυψαν τα συστήματα RAID, που συνδυάζουν αποτελεσματικά φθηνούς δίσκους, για τη δημιουργία γρηγορότερων και μεγαλύτερων αποθηκευτικών χώρων, προσφέροντας επίσης ασφάλεια στις σπάνιες, αλλά καταστροφικές αστοχίες υλικού.

Στη συνέχεια του κεφαλάιου γίνεται μία σύντομη περιγραφή των μαγνητικών δίσκων που αποτελούν και το κυριότερο μέσω αποθήκευσης ενώ παρουσιάζονται και οι διάφορες διατάξεις των συστημάτων RAID. Τέλος, περιγράφονται οι αρχιτεκτονικές δικτυακών συστημάτων αποθήκευσης και ο τρόπος λειτουργίας του συστήματος vRAID.

4.1 Μαγνητικοί Δίσκοι

Παρά τις συνεχόμενες επιθέσεις από τις νέες τεχνολογίες, οι μαγνητικοί δίσκοι έχουν κυριαρχήσει στη διατηρήσιμη μνήμη από το 1965. Οι μαγνητικοί δίσκοι παίζουν δύο ρόλους στα υπολογιστικά συστήματα:



Σχήμα 4.1 – Οι δίσκοι αποτελούνται από πλάκες, τροχιές και τομείς. Και οι δύο πλευρές των πλακών είναι επιστρωμένες έτσι ώστε η πληροφορία να μπορεί να αποθηκεύεται και στις δύο επιφάνειες. Ένας κύλινδρος αναφέρεται σε μία τροχιά στην ίδια θέση κάθε πλάκας.

- Μακροπρόθεσμη, διατηρήσιμη μνήμη αποθήκευσης αρχείων, ακόμη και όταν κανένα πρόγραμμα δεν εκτελείται,
- Ένα επίπεδο στην ιεραρχία μνήμης κάτω από την κύρια μνήμη που χρησιμοποιείται ως υποστηρικτικός χώρος για την εικονική μνήμη κατά την διάρκεια εκτέλεσης των προγραμμάτων.

Ένας μαγνητικός δίσκος αποτελείται από μία σειρά από πλάκες (platters), οι οποίες περιστρέφονται γύρω από έναν άξονα με ταχύτητα 3600 έως 15000 περιστροφές το λεπτό. Αυτές οι πλάκες είναι μεταλλικοί ή γυάλινοι δίσκοι και είναι καλυμμένοι με μαγνητικό υλικό εγγραφής και στις δύο πλευρές τους. Η επιφάνεια της κάθε πλάκας διαιρείται σε ομόκεντρους κύκλους, που ονομάζονται τροχιές (tracks). Τυπικά, υπάρχουν από 5000 μέχρι 30000 τροχιές σε κάθε μία από τις επιφάνειες. Κάθε τροχιά με τη σειρά της χωρίζεται σε τομείς (sectors), οι οποίοι περιέχουν την πληροφορία. Ο τομέας αποτελεί τη μικρότερη μονάδα που μπορεί να εγγραφεί ή να διαβαστεί και τα περισσότερα συστήματα ορίζουν το μέγεθος τυπικά στα 512bytes δεδομένων. Η ακολουθία που εγγράφεται στο μαγνητικό μέσο είναι ο αριθμός τομέα, ένα κενό, η πληροφορία αυτού του τομέα συμπεριλαμβανομένου κώδικα διόρθωσης σφαλμάτων, ένα κενό, ο αριθμός τομέα του επόμενου τομέα κοκ.

Προκειμένου να διαβάσουμε και να γράψουμε πληροφορίες μέσα σε έναν τομέα, ένας κινούμενος βραχίονας (arm) που περιέχει μια κεφαλή ανάγνωσης/εγγραφής (read/write head) βρίσκεται πάνω σε κάθε επιφάνεια. Οι βραχίονες όλων των επιφανειών συνδέονται μεταξύ τους και κινούνται συζευγμένα, έτσι ώστε όλοι οι βραχίονες να βρίσκονται πάνω στην ίδια

τροχιά όλων των επιφανειών. Ο όρος κύλινδρος (cylinder) χρησιμοποιείται για να αναφερόμαστε σε όλες τις τροχιές κάτω από τους βραχίονες όλων των επιφανειών σε μια δεδομένη χρονική στιγμή.

Για να διαβαστεί ή να εγγραφεί ένας τομέας, ο ελεγκτής δίσκου στέλνει μια εντολή στο βραχίονα προκειμένου να μεταβεί στην κατάλληλη τροχιά. Αυτή η ενέργεια ονομάζεται αναζήτηση (seek), και ο χρόνος που απαιτείται για να μετακινηθεί ο βραχίονας στην επιθυμητή τροχιά καλείται χρόνος αναζήτησης (seek time).

4.2 Πλεονάζουσες Συστοιχίες Φθηνών Δίσκων

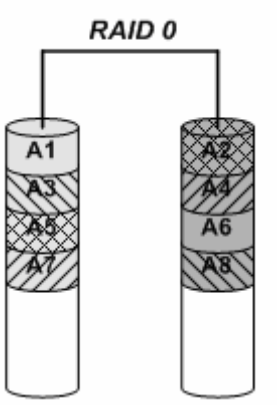
Οι πλεονάζουσες συστοιχίες φθηνών δίσκων (**R**edundant **A**rrays of **I**nexpensive **D**isks) όπως ονομάστηκαν από τους ανθρώπους που τις σχεδίασαν, αποτελούν ομάδες δίσκων που χρησιμοποιούνται από κοινού, με σκοπό τη δημιουργία ενός αξιόπιστου αποθηκευτικού μέσου υψηλών επιδόσεων. Ο όρος RAID περιλαμβάνει όλα εκείνα τα αποθηκευτικά συστήματα που διαμοιράζουν τα δεδομένα και διατηρούν πολλαπλά αντίγραφα τους σε ένα σύνολο από δίσκους. Με τον τρόπο αυτό εξασφαλίζουν της ακεραιότητας των δεδομένων και βελτιώνουν την απόδοση των λειτουργιών ανάγνωσης και εγγραφής.

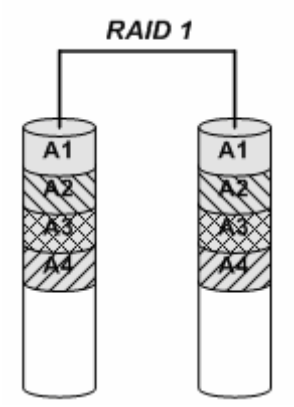
Τα συστήματα RAID ομαδοποιούν τους δίσκους σε μία ενιαία μονάδα Ε/Ε χρησιμοποιώντας ειδικό υλικό ή λογισμικό, οπότε έχουμε το λεγόμενο hardware ή software RAID αντίστοιχα. Στην περίπτωση του υλικού, το σύστημα έχει σχεδιαστεί έτσι ώστε να παρουσιάζεται στο λειτουργικό ως μία μονάδα Ε/Ε, ενώ το λειτουργικό αγνοεί τις λεπτομέρειες υλοποίησης. Αντίθετα, στην περίπτωση του λογισμικού το σύστημα RAID συνήθως υλοποιείται από το ίδιο το λειτουργικό και φαίνεται από της εφαρμογές χώρου χρήση ως μία ενιαία μονάδα Ε/Ε.

Στα συστήματα RAID χρησιμοποιούνται κυρίως οι εξής τεχνικές :

- ο κατοπτρισμός δεδομένων (mirroring), δηλαδή η αντιγραφή των δεδομένων σε περισσότερους του ενός δίσκου,
- ο επιμερισμός δεδομένων (striping) και
- η διόρθωση σφαλμάτων (error correction) που επιτυγχάνεται με την χρήση της πλεονάζουσας (redundant) πληροφορίας που υπάρχει στο σύστημα και επιτρέπει τον εντοπισμό και την αποκατάσταση σφαλμάτων

Τα διάφορα συστήματα RAID που υπάρχουν χρησιμοποιούν ορισμένες ή και όλες αυτές τις τεχνικές μαζί ανάλογα με τις απαιτήσεις που υπάρχουν για την κάθε εφαρμογή. Οι παραλλαγές της υλοποίησης είναι πολλές και συνήθως αναφέρονται ως επίπεδα RAID. Στη συνέχεια αυτής της παραγράφου περιγράφονται τα κυριότερα επίπεδα RAID.





Σχήμα 4.2 – Χωρίς Πλεονασμό RAID 0

Σχήμα 4.3 – Κατοπτρισμός RAID 1

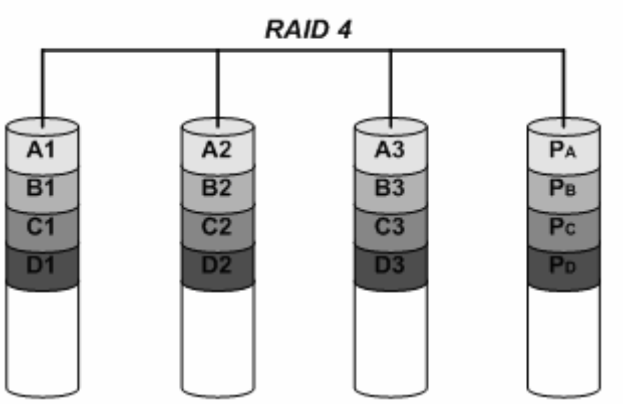
4.2.1 RAID 0

Αυτός ο συμβολισμός αναφέρεται σε μια συστοιχία δίσκων, στην οποία τα δεδομένα επιμερίζονται, αλλά δεν υπάρχει πλεονασμός για να ανεχθεί αποτυχία δίσκου. Ο επιμερισμός σε ένα σύνολο δίσκων κάνει τη συλλογή να εμφανίζεται στο λογισμικό ως ένας μοναδικός μεγάλος δίσκος, ο οποίος απλοποιεί τη διαχείριση αποθήκευσης. Επίσης, βελτιώνει την απόδοση για μεγάλες προσπελάσεις, αφού πολλοί δίσκοι μπορούν να λειτουργούν ταυτόχρονα. Τα συστήματα επεξεργασίας βίντεο, για παράδειγμα , συχνά επιμερίζουν τα δεδομένα τους.

Το RAID 0 είναι κάπως εσφαλμένα χαρακτηρισμένο καθώς δεν υπάρχει πλεονασμός, δεν βρίσκεται στην αρχική ταξινόμηση των RAID και ο επιμερισμός προηγείται των RAID. Παρά ταύτα , τα επίπεδα RAID συχνά αφήνονται να καθοριστούν από τον χειριστή κατά τη δημιουργία ενός συστήματος αποθήκευσης και το RAID 0 συχνά αναφέρεται ως μια από τις επιλογές. Έτσι, ο όρος RAID 0 χρησιμοποιείται ευρέως.

4.2.2 RAID 1

Το παραδοσιακό μοντέλο για την ανοχή αποτυχιών, που ονομάζεται κατοπτρισμός η σκίαση, χρησιμοποιεί διπλάσιους δίσκους από ότι το RAID 0. Οποτεδήποτε εγγράφονται δεδομένα σε έναν δίσκο, αυτά τα δεδομένα εγγράφονται επίσης και στον πλεονάζοντα δίσκο, ώστε να υπάρχουν πάντα δύο αντίγραφα της πληροφορίας. Εάν ένας δίσκος αποτύχει, το σύστημα απλά στρέφεται στον «κατοπτρικό» του για να πάρει την επιθυμητή πληροφορία. Η διάταξη αυτή χρησιμοποιείται κυρίως σε συστήματα, όπως οι βάσεις δεδομένων, όπου η διαθεσιμότητα του συστήματος και ο ρυθμός απάντησης των αιτήσεων είναι πιο σημαντικός από την αποτελεσματική αξιοποίηση του διαθέσιμου αποθηκευτικού χώρου. Τέλος, πρέπει να σημειωθεί ότι ο κατοπτρισμός είναι η πιο ακριβή λύση RAID, καθώς απαιτεί τους περισσότερους δίσκους.



Σχήμα 4.4 – RAID 4. Η ισοτιμία υπολογίζεται ανά block και όχι ανά byte όπως στην περίπτωση του RAID 3

4.2.3 RAID 2

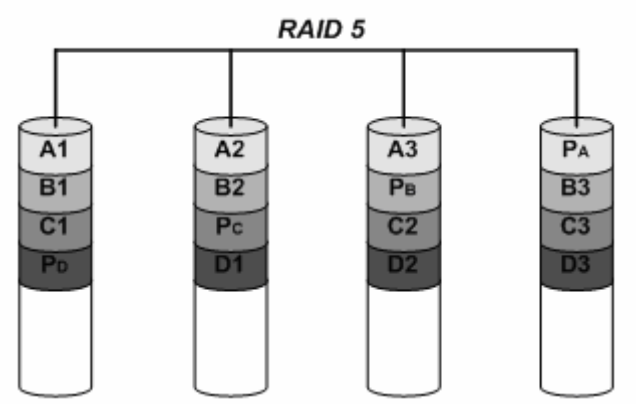
Αυτή η διάταξη πλέον δεν βρίσκει εφαρμογή σε εμπορικά συστήματα και συνήθως δεν αναφέρεται πλέον, γιατί παρουσιάζει μεγάλο κόστος υλοποίησης, αφού απαιτεί πάρα πολλούς δίσκους, και δεν έχει και τις αναμενόμενες επιδόσεις. Για την διατήρηση της ακεραιότητας του συστήματος κάνει χρήση κωδικών επιδιόρθωσης σφαλμάτων Hamming, ενώ ο αριθμός των πλεοναζόντων δίσκων πρέπει να είναι ο λογάριθμος του συνολικού αριθμού δίσκων του συστήματος.

4.2.4 RAID 3 και RAID 4

Αντί να έχουμε ένα ολοκληρωμένο αντίγραφο των πρωτότυπων δεδομένων για κάθε δίσκο, χρειάζεται μόνο να προσθέσουμε αρκετή πλεονάζουσα πληροφορία για να ανακτήσουμε τη χαμένη πληροφορία στην περίπτωση αποτυχίας. Αναγνώσεις ή εγγραφές γίνονται σε όλους τους δίσκους της ομάδας, με έναν πρόσθετο δίσκο να διατηρεί την πληροφορία ελέγχου σε περίπτωση που υπάρχει κάποια αποτυχία. Το RAID 3 είναι δημοφιλές σε εφαρμογές με μεγάλα σύνολα δεδομένων, όπως είναι τα πολυμέσα και μερικοί επιστημονικοί κώδικες.

Όταν ένας δίσκος αποτύχει, τότε αφαιρώντας όλα τα δεδομένα των καλών δίσκων από τον δίσκο ισοτιμίας, η εναπομένουσα πληροφορία είναι η χαμένη πληροφορία. Η ισοτιμία είναι απλά το άθροισμα modulo δύο. Η υπόθεση πίσω από αυτή την τεχνική είναι ότι οι αποτυχίες είναι τόσο σπάνιες, ώστε το να καταναλώνουμε περισσότερο χρόνο για να επανέλθουμε από μια αποτυχία μειώνοντας τον πλεονάζοντα χώρο αποθήκευσης αποτελεί μια καλή επιλογή.

Η διαφορά που υπάρχει ανάμεσα στα επίπεδα 3 και 4 είναι ότι στο RAID 3 τα δεδομένα κατανέμονται ανά byte, ενώ στο RAID 4 ανά block. Κατά συνέπεια στο RAID 3 κάθε εγγραφή ή ανάγνωση απαιτεί τη συμμετοχή όλων των σκληρών δίσκων, γεγονός που δημιουργεί πρόβλημα στην ταχύτητα απόκρισης του συστήματος αποθήκευσης. Αντίθετα, το RAID 4 που διανέμει τα δεδομένα ανά block, δίνει τη δυνατότητα στο σύστημα να



Σχήμα 4.5 – RAID 5. Κατανέμοντας τα μπλοκ ισοτιμίας σε όλους τους δίσκους, ορισμένες μικρές εγγραφές μπορούν να πραγματοποιηθούν παράλληλα.

επεξεργαστεί και να αποκριθεί παράλληλα σε περισσότερες από μία αιτήσεις τη φορά, προσφέροντας έτσι καλύτερη απόδοση.

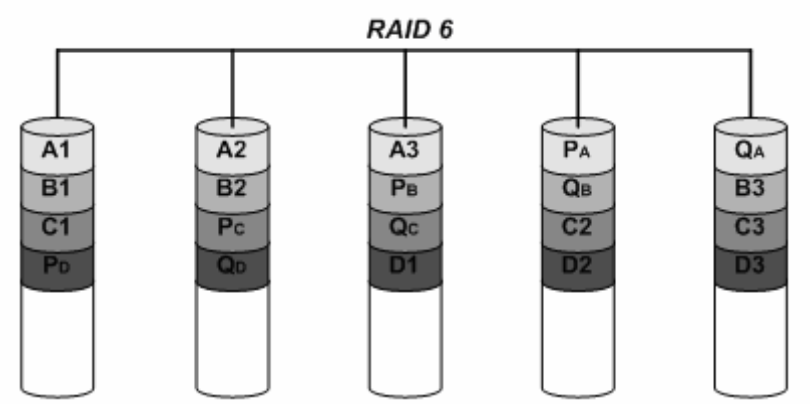
4.2.5 RAID 5

Το RAID 4 υποστηρίζει αποτελεσματικά ένα μίγμα μεγάλων αναγνώσεων, μεγάλων εγγραφών, μικρών αναγνώσεων και μικρών εγγραφών. Ένα μειονέκτημα του συστήματος είναι ότι ο δίσκος ισοτιμίας πρέπει να ενημερώνεται σε κάθε εγγραφή, οπότε είναι σημείο συμφόρησης για συνεχείς εγγραφές. Προκειμένου να επιδιορθωθεί η συμφόρηση εγγραφής ισοτιμίας, η πληροφορία ισοτιμίας μπορεί να απλωθεί σε όλους τους δίσκους , ώστε να μην υπάρχει μοναδικό σημείο συμφόρησης για εγγραφές. Η κατανεμημένη οργάνωση της ισοτιμίας αποτελεί το RAID 5.

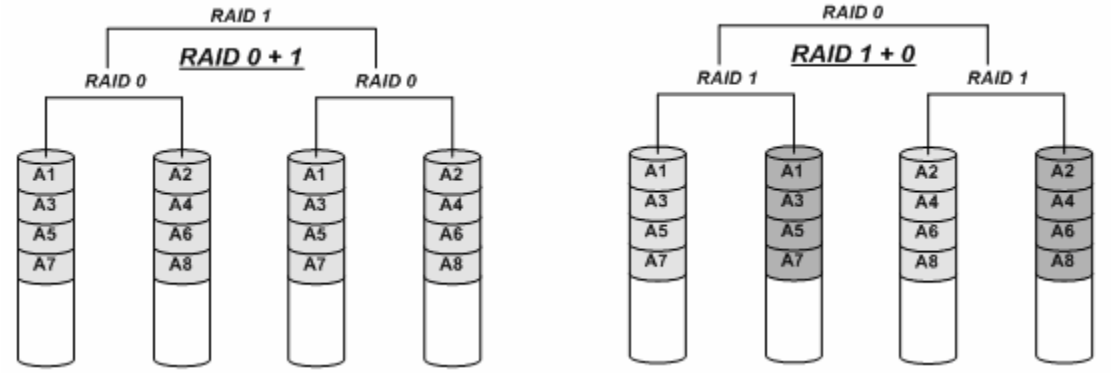
Όπως φαίνεται και στο σχήμα 4.5, στο RAID 5 η ισοτιμία που σχετίζεται με κάθε γραμμή των μπλοκ δεδομένων δεν περιορίζεται σε έναν δίσκο πλέον. Αυτή η οργάνωση επιτρέπει σε πολλαπλές εγγραφές να λαμβάνουν χώρα ταυτόχρονα, εφόσον οι μονάδες διαμοιρασμού δεν εντοπίζονται μέσα στους ίδιους δίσκους.

4.2.6 RAID 6

Τα σχήματα που βασίζονται στην ισοτιμία προστατεύουν ενάντια σε μια μοναδική αυτό-αναγνωριζόμενη αποτυχία. Όταν η διόρθωση μιας μοναδικής αποτυχίας δεν επαρκεί, η ισοτιμία μπορεί να γενικευθεί προκειμένου να έχουμε έναν δεύτερο υπολογισμό πάνω στα δεδομένα και έναν άλλο δίσκο ελέγχου πληροφορίας. Αυτό το δεύτερο μπλοκ ελέγχου επιτρέπει την επαναφορά από μια δεύτερη αποτυχία. Έτσι , η συνολική επιβάρυνση της αποθήκευσης είναι διπλάσια από αυτή του RAID5.



Σχήμα 4.6 – RAID 6, πλεονασμός P + Q



Σχήμα 4.7– RAID 01 και RAID 10

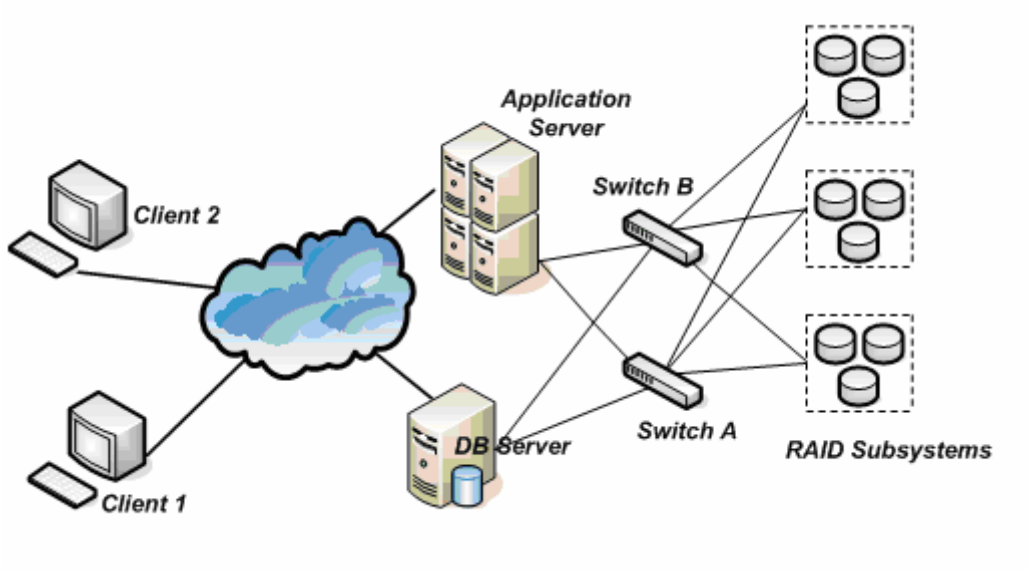
Ο αποκαλούμενος πλεονασμός P+Q, κάνει χρήση κωδικών εντοπισμού σφαλμάτων Reed-Solomon, που επιτρέπουν τον εντοπισμό οποιουδήποτε αριθμού αποτυχιών δίσκου προσθέτοντας στο σύστημα τους αντίστοιχους πλεονάζοντες δίσκους. Η δομή της συστοιχίας μοιάζει πάρα πολύ με αυτήν του RAID 5, κατανέμοντας την πλεονάζουσα πληροφορία σε όλους τους δίσκους.

4.2.7 Υβριδικά επίπεδα RAID

Τα επίπεδα RAID που περιγράφονται παραπάνω συχνά συνδυάζονται, προκειμένου να επιτευχθεί καλύτερη απόδοση. Στις περισσότερες περιπτώσεις ένα επίπεδο RAID που χρησιμοποιεί πλεονασμό πληροφορίας, συνδυάζεται με το RAID 0. Στο σχήμα 4.7 παρουσιάζονται οι διατάξεις για τα RAID 0+1 και RAID 1+0.

4.2.8 Επίδοση Συστημάτων RAID

Συμπερασματικά μπορούμε να διαπιστώσουμε εύκολα ότι οι διατάξεις RAID 0 και 1 απευθύνονται σε λύσεις όπου δίνεται μεγάλη έμφαση στις επιδόσεις του συστήματος, θυσιάζοντας όμως την αξιοπιστία του και το συνολικό κόστος. Αντίθετα τα επίπεδα 5 και 6



Σχήμα 4.8 – Τυπική διάταξη SAN

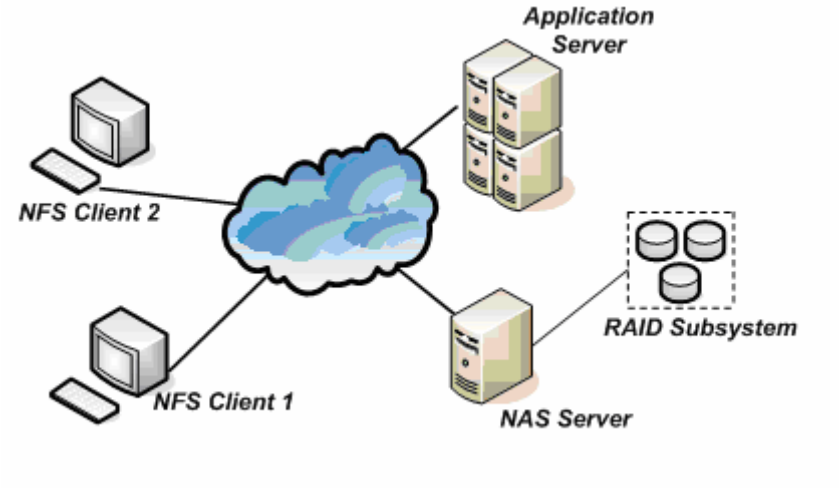
προσφέρουν αυξημένη ασφάλεια και οι επιδόσεις τους κυμαίνονται πολύ κοντά σε αυτές των επιπέδων 0 και 1. Βέβαια η ιδανική λύση σε ένα μεγάλο σύστημα δεν είναι ποτέ μονομερής. Ο συνδυασμός των επιπέδων RAID ανάλογα με τις απαιτήσεις της εφαρμογής μπορεί να βελτιώσει σε σημαντικό βαθμό την συνολική απόδοση του συστήματος.

4.3 Αρχιτεκτονικές Δικτυακών Συστημάτων Αποθήκευσης

Παραπάνω περιγράψαμε τις διάφορες διατάξεις που μπορούν να έχουν τα τοπικά συστήματα αποθήκευσης. Τα μειονεκτήματά τους είναι ότι ο αποθηκευτικός τους χώρος παραμένει περιορισμένος αλλά το κυριότερο ότι δεν μπορεί να είναι προσπελάσιμος από άλλους υπολογιστές. Λύση στο παραπάνω πρόβλημα δίνουν τα συστήματα δικτυακής αποθήκευσης που επιτρέπουν σε απομακρυσμένους υπολογιστές να συνδέονται με αυτά και να ανταλλάσσουν δεδομένα. Οι λύσεις που προσφέρονται, εκτείνονται σε μεγάλο εύρος με έμφαση πλέον να δίνεται στην έρευνα για κατανεμημένα αποθηκευτικά συστήματα τα οποία θα μπορούν να διαχειριστούν μεγάλα ποσά πληροφορίας. Οι κυριότερες τεχνολογίες που χρησιμοποιούνται για τη δικτυακή αποθήκευση δεδομένων είναι η τεχνολογία NAS και η τεχνολογία SAN.

4.3.1 Τεχνολογία SAN

Η τεχνολογία SAN (Storage Area Network) συνιστά ένα ξεχωριστό και αυτόνομο δίκτυο που διασυνδέει τα αποθηκευτικά μέσα εξυπηρετητών, αλλά και αποθηκευτικά μέσα που είναι άμεσα συνδεδεμένα με αυτούς, όπως δίσκους και οπτικά μέσα σε ένα ξεχωριστό και αυτόνομο δίκτυο. Συνεπώς στο SAN κινούνται μόνο δεδομένα κι έτσι δεν παρατηρούνται συμφορήσεις όπως σε ένα κοινό δίκτυο. Μερικά από τα βασικά πλεονεκτήματα της



Σχήμα 4.9 – Τυπική διάταξη NAS

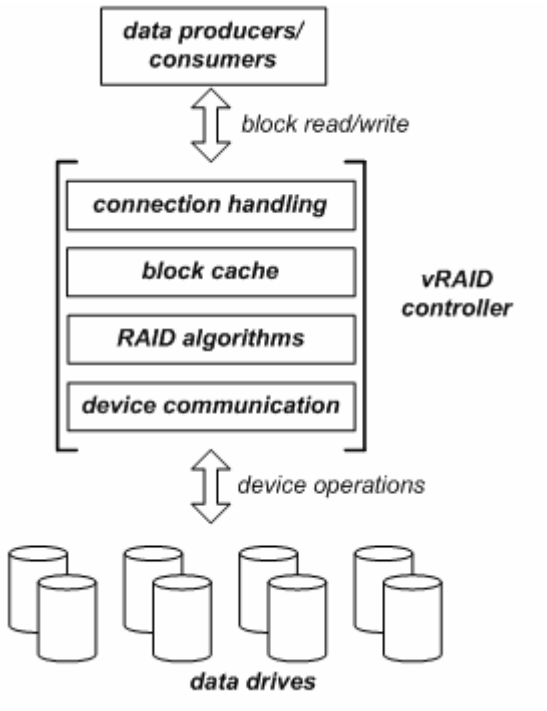
τεχνολογίας είναι ο πολύ μικρός χρόνος μεταφοράς δεδομένων, η εύκολη επέκταση, η εύκολη κεντρική διαχείριση, οι γρήγορες και αξιόπιστες διαδικασίες λήψης αντιγράφων ασφαλείας και αποκατάστασης και η μεγάλη αξιοπιστία και ανοχή σε καταστροφές.

4.3.2 Τεχνολογία NAS

Η τεχνολογία NAS (Network Attached Storage) είναι ένα σύστημα αποτελούμενο από συσκευές κατάλληλες για διαμοιρασμό αρχείων στις οποίες η πρόσβαση επιτυγχάνεται μέσω ενός δικτυακού πρωτοκόλλου, όπως το TCP/IP, και εφαρμογών, όπως Network File System (NFS) και Common Internet File System (CIFS). Ένα σύστημα NAS συχνά αποτελείται από ένα σύστημα RAID συνδεδεμένο με έναν εξυπηρετητή. Τα βασικά πλεονεκτήματα της τεχνολογίας είναι η συμβατότητα με τα περισσότερα λειτουργικά συστήματα, η μεγάλη χωρητικότητα, ο εύκολος διαμοιρασμός αρχείων, η εύκολη εγκατάσταση και συντήρηση. Στα μειονεκτήματά του περιλαμβάνονται η καθυστέρηση εξαιτίας των επιβαρύνσεων από το δικτυακό πρωτόκολλο, η μείωση του εύρους ζώνης του LAN και το γεγονός ότι δεν μπορεί να υπάρξει κλιμάκωση, χωρίς να μειωθεί η απόδοση.

4.4 vRAID

Το vRAID είναι ένα κατανεμημένο σύστημα αποθήκευσης που αναπτύχθηκε από το εργαστήριο. Ανήκει στην κατηγορία των SAN προσφέροντας ως επίπεδο επικοινωνίας αυτό του block. Σκοπός του είναι να μπορεί να αυξομειώνεται κατά τις απαιτήσεις της εφαρμογής. Παράλληλα θα προσφέρει αυξημένες δυνατότητες διαχείρισης επιτρέποντας την εύκολη δημιουργία αντιγράφων ασφαλείας. Στο επίπεδο του hardware προσπαθεί να εκμεταλλευτεί ήδη δημοφιλείς λύσεις, όπως το Ethernet στο επίπεδο δικτύωσης, ενσωματώνοντας όμως και πιο εξελιγμένες λύσεις hardware στα επιμέρους τμήματά του που επιτρέπουν και την αύξηση των επιδόσεών του. Έτσι θα είναι εύκολη η εγκατάστασή του στις υπάρχουσες



Σχήμα 4.10 – Αρχιτεκτονική του vRAID

υποδομές χωρίς αύξηση του κόστους, ενώ παράλληλα θα είναι δυνατή και η μετάβαση σε μελλοντικές λύσεις.

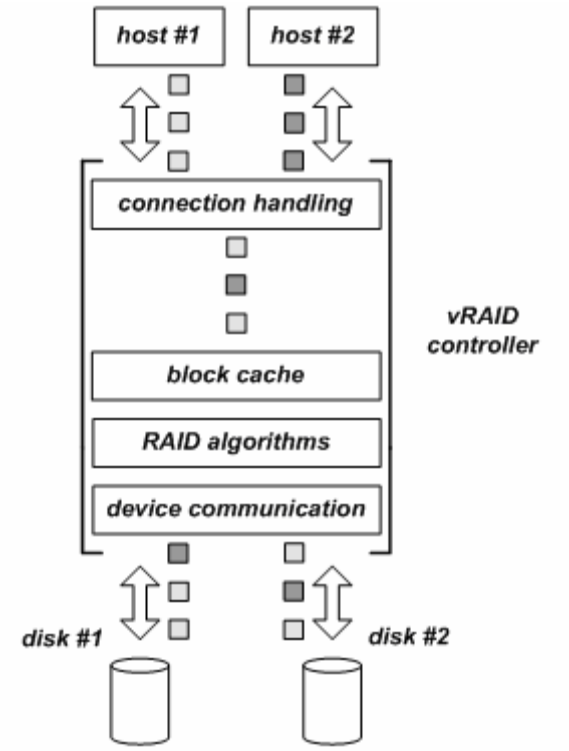
4.4.1 Αρχιτεκτονική του vRAID

Όπως φαίνεται και από το σχήμα 4.10, η αρχιτεκτονική του vRAID μπορεί να χωριστεί σε τρία βασικά κομμάτια :

- τους πελάτες του συστήματος που στέλνουν αιτήσεις ανάγνωσης και εγγραφής,
- τους ελεγκτές – controllers και
- τα απομακρυσμένα αποθηκευτικά συστήματα.

Οι ελεγκτές αποτελούν το σημαντικότερο κομμάτι της αρχιτεκτονικής και είναι υπεύθυνοι για τη διαχείριση και τον έλεγχο της επικοινωνίας μεταξύ των αποθηκευτικών συστημάτων και των εφαρμογών που δημιουργούν τις αιτήσεις ανάγνωσης/εγγραφής. Συγκεκριμένα, λαμβάνουν τις αιτήσεις E/E από τους πελάτες οι οποίες αναφέρονται στον ενιαίο «εικονικό» χώρο block που παρέχει το σύστημα και τις μεταφράζουν σε αιτήσεις E/E στις επιμέρους αποθηκευτικές συσκευές. Η εσωτερική δομή τους αποτελείται από τέσσερα επίπεδα :

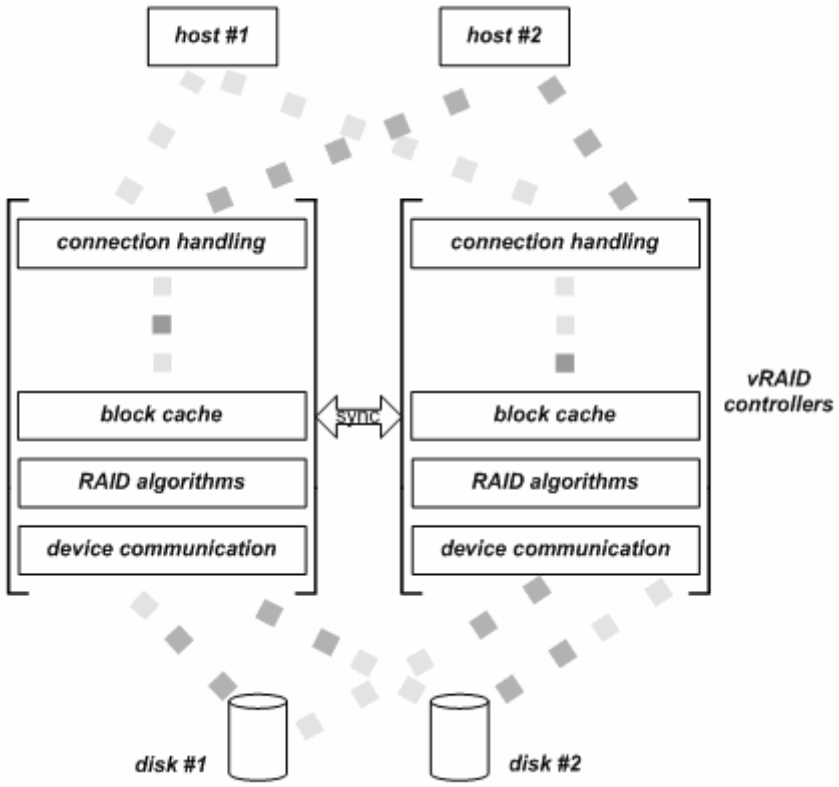
- *Διαχείριση Συνδέσεων* : Τοποθετώντας τη διαχείριση συνδέσεων σε ένα ξεχωριστό επίπεδο οι vRAID controllers είναι σε θέση να διαχειριστούν ταυτόχρονα έναν μεγάλο αριθμό από πελάτες. Το επίπεδο αυτό είναι υπεύθυνο για την υλοποίηση του πρωτοκόλλου επικοινωνίας ανάμεσα στους πελάτες και τον controller, εγκαθιστώντας τις ανάλογες συνδέσεις και αποκρύπτοντας από τα χαμηλότερα



Σχήμα 4.11 – Επίπεδο διαχείρισης συνδέσεων. Ο τρόπος που δρομολογούνται οι απαντήσεις από τα χαμηλότερα επίπεδα του controller αποκρύπτετε από τους πελάτες.

επίπεδα του vRAID την πηγή της κάθε αίτησης. Τα χαμηλότερα επίπεδα γνωρίζουν μόνο τα χαρακτηριστικά της αίτησης και δεν ενδιαφέρονται για την πηγή της αίτησης. Όταν μία αίτηση φτάνει στο επίπεδο διαχείρισης συνδέσεων, σημαδεύεται με έναν αριθμό μοναδικό για τον κάθε πελάτη. Με τον τρόπο αυτό οι απαντήσεις μπορούν να δρομολογηθούν πάλι πίσω στον αιτούντα. Ο αριθμός αυτός συνοδεύει την αίτηση σε όλο τον κύκλο ζωής της μέσα στον controller και ανάλογα με την υλοποίηση μπορεί να περιλαμβάνεται και στις ανταλλασσόμενες πληροφορίες με τους απομακρυσμένους δίσκους.

- *Block cache:* Η διαδικασία της ενδιάμεσης αποθήκευσης χρησιμοποιείται ως μία τεχνική για την επιτάχυνση εξυπηρέτησης των αιτήσεων που απευθύνονται στις ίδιες διευθύνσεις block. Οι vRAID controllers αποθηκεύουν blocks στην cache στο επίπεδο του ενιαίου αποθηκευτικού χώρου διευθύνσεων, με σκοπό τα δεδομένα να μη φτάσουν στο επίπεδο του RAID. Η καθυστέρηση που αφορά την εγγραφή και την ανάγνωση στο επίπεδο RAID εξαρτάται από τον τύπο της αίτησης, αλλά γενικά απαιτεί την παραγωγή και την εκτέλεση πολυάριθμων αιτήσεων προς τους δίσκους που είναι αρκετά χρονοβόρο. Επίσης η ύπαρξη της cache μπορεί να βοηθήσει την επίδοση κατανεμημένων εφαρμογών στους πελάτες οι οποίες απαιτούν την απενεργοποίηση των τοπικών caches για λόγους ακεραιότητας ανάμεσα στους πελάτες. Βέβαια το ίδιο πρόβλημα προκύπτει και για το σύστημα vRAID στην



Σχήμα 4.12 – Στην περίπτωση λειτουργίας περισσότερων του ενός controller θα πρέπει να εφαρμοστούν μηχανισμοί συγχρονισμού των περιεχομένων των caches.

περίπτωση που ενεργοποιηθούν περισσότεροι από έναν controllers (σχήμα 4.12) και επιβάλλεται η χρησιμοποίηση κάποιου πρωτοκόλλου διατήρησης της ακεραιότητας των δεδομένων.

- *Αλγόριθμοι RAID:* Σκοπός του συστήματος είναι να υλοποιήσει τους υπάρχοντες αλγόριθμους που χρησιμοποιούνται ως επί το πλείστον από τα περισσότερα αποθηκευτικά συστήματα. Μέχρι στιγμής προσφέρονται υλοποιήσεις των RAID 0 και RAID 1, με σκοπό στο μέλλον να προστεθούν και υλοποιήσεις των RAID 5 και RAID 6.
- *Επικοινωνία με τις απομακρυσμένες συσκευές:* Αφορά την επικοινωνία των ελεγκτών με τα αποθηκευτικά συστήματα και το πρωτόκολλο που χρησιμοποιείται είναι το TCP/IP.

Όπως φαίνεται και από τα παραπάνω αλλά και από το σχήμα 4.10 που παρουσιάζει την αρχιτεκτονική του συστήματος vRAID, διακρίνονται δύο επίπεδα επικοινωνίας. Το πρώτο αφορά την επικοινωνία των ελεγκτών με τα αποθηκευτικά συστήματα. Το δεύτερο επίπεδο είναι αυτό της επικοινωνίας ανάμεσα στον ελεγκτή και τους πελάτες και είναι αυτό το οποίο πραγματεύεται η παρούσα διπλωματική. Στη μέχρι τώρα υλοποίηση χρησιμοποιούνται διεπαφές επικοινωνίας που υποστηρίζονται από τα υπάρχοντα λειτουργικά συστήματα, το Linux και το FreeBSD, όπως το NBD και το Ggate ή το AoE. Οι διεπαφές αυτές κάνουν

χρήση της υπάρχουσας υποδομής δικτύωσης Ethernet, περιορίζοντας σημαντικά το κόστος υλοποίησης, αφού δεν απαιτούν την ύπαρξη εξειδικευμένης τεχνολογίας δικτύωσης.

Στην παρούσα διπλωματική εξετάζεται η περίπτωση χρήσης της συσκευής NBD για την επικοινωνία πελάτη-controller. Ο μόνος περιορισμός που υπάρχει στην περίπτωση αυτή αφορά την αποκλειστική χρήση ενός μόνο controller από τον κάθε πελάτη. Έτσι η συσκευή NBD επεκτείνεται ώστε να παρέχει σε κάθε πελάτη τη δυνατότητα χρήσης περισσότερων του ενός controllers.

5 *Σχεδιασμός της συσκευής*

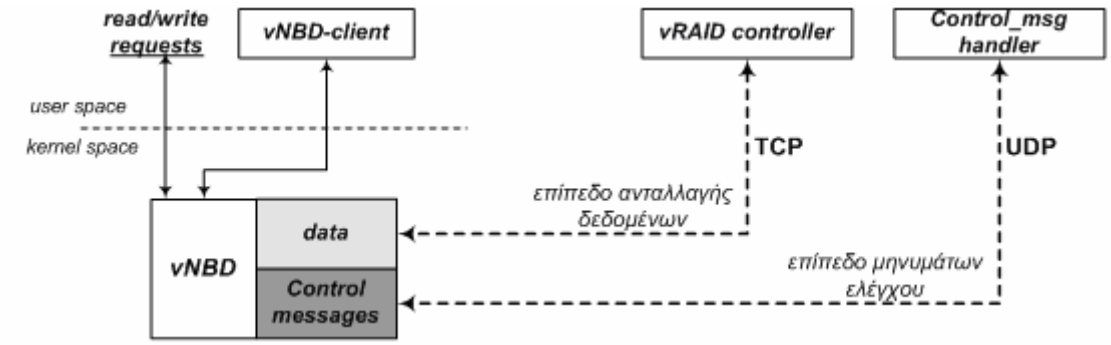
Στο κεφάλαιο αυτό παρουσιάζονται τα χαρακτηριστικά και η δομή της δικτυακής συσκευής block που υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής. Ουσιαστικά αποτελεί μία επέκταση της συσκευής NBD του λειτουργικού συστήματος Linux και καθώς δημιουργήθηκε για να δίνει τη δυνατότητα χρήσης του κατανεμημένου συστήματος αποθήκευσης vRAID, ονομάστηκε vNBD.

5.1 Προδιαγραφές λειτουργίας της συσκευής

Όπως αναφέρθηκε και προηγουμένως, στην περίπτωση του vRAID η επικοινωνία ανάμεσα στους πελάτες και τα απομακρυσμένα αποθηκευτικά συστήματα δε γίνεται απευθείας. Αντίθετα, παρεμβάλλεται ένας ελεγκτής (controller) που πραγματοποιεί τον έλεγχο ανάμεσα στους πελάτες και τις αποθηκευτικές συσκευές. Η επικοινωνία ανάμεσα στους πελάτες και τον controller γίνεται με την χρήση της συσκευής NBD, η οποία δημιουργεί τοπικά στον πελάτη μία εικονική συσκευή block, που προωθεί τις αιτήσεις ανάγνωσης και εγγραφής στον controller.

Από την άλλη πλευρά, η συσκευή vNBD καλείται να δίνει στον πελάτη τη δυνατότητα χρήσης περισσότερων του ενός ελεγκτή. Κατά συνέπεια, θα πρέπει να μπορεί να διατηρεί και να ενημερώνει έναν πίνακα με τους διαθέσιμους controllers, στους οποίους θα προωθούνται οι αιτήσεις. Επιπλέον, αν κάποιος από τους ελεγκτές σταματήσει για κάποιο λόγο να λειτουργεί, η συσκευή θα πρέπει να πάψει να τον χρησιμοποιεί.

Βάσει των παραπάνω, η επικοινωνία πελάτη-controller θα χωρίζεται σε δύο επίπεδα. Το πρώτο αφορά την ανταλλαγή αιτήσεων για ανάγνωση και εγγραφή δεδομένων και θα αναφέρετε ως *επίπεδο ανταλλαγής δεδομένων*, ενώ το δεύτερο σχετίζεται με τα μηνύματα ελέγχου για τη δημιουργία και τον έλεγχο της σύνδεσης και θα αναφέρετε ως *επίπεδο μηνυμάτων ελέγχου*.



Σχήμα 5.1 – Λειτουργία της συσκευής vNBD.

5.2 Επίπεδα Επικοινωνίας Πελάτη - Controller

Πριν την περιγραφή των δύο επιπέδων επικοινωνίας, όπως αυτά ορίστηκαν παραπάνω, θα πρέπει να αναλυθεί η δομή και τα συστατικά μέρη τόσο του πελάτη όσο και του controller. Στην πλευρά του πελάτη, εκτός από τον οδηγό της συσκευής θα υπάρχει και μία εφαρμογή η οποία θα χρησιμοποιεί τον οδηγό και θα διαχειρίζεται την συσκευή. Αντίστοιχα, στην πλευρά του controller θα προστεθεί μία διεργασία επιφορτισμένη με τη διαχείριση των μηνυμάτων του επιπέδου ελέγχου. Στο σχήμα 5.1 φαίνεται η δομή του πελάτη και του controller και ο τρόπος που αυτά τα δύο συστήματα αλληλεπιδρούν.

5.2.1 Επίπεδο Ανταλλαγής Δεδομένων

Για τη μεταφορά των δεδομένων χρησιμοποιείται το πρωτόκολλο NBD, το οποίο βρίσκεται πάνω από το πρωτόκολλο TCP. Για τον κάθε controller που υπάρχει στον πίνακα της συσκευής, θα υπάρχει και ένα TCP socket για την ανταλλαγή μηνυμάτων δεδομένων. Το socket αυτό θα δημιουργείται και θα αρχικοποιείται στον χώρο χρήστη μέσω της εφαρμογής vNBD-client, η οποία στη συνέχεια θα το περνά στον πυρήνα με την κατάλληλη κλήση ioctl, ενώ ταυτόχρονα θα δημιουργείται και μία διεργασία υπεύθυνη να λαμβάνει τα μηνύματα δεδομένων που φτάνουν από τον controller (λεπτομέρειες για τον τρόπο με τον οποίο γίνεται αυτό δίνονται στη συνέχεια του κεφαλαίου).

Σε κάθε δεδομένη χρονική στιγμή, η συσκευή θα διαθέτει έναν αριθμό ενεργών controllers στους οποίους θα μπορεί να προωθήσει τις αιτήσεις ανάγνωσης ή εγγραφής. Έτσι, όταν ο πυρήνας θα καλεί την μέθοδο που είναι υπεύθυνη για την εξυπηρέτηση των αιτήσεων, θα πρέπει για κάθε δομή request της ουράς της συσκευής να επιλέξει σε ποιόν controller θα την στείλει. Στη συνέχεια, η αίτηση τοποθετείται στην ουρά που αντιστοιχεί στον συγκεκριμένο controller και παραμένει εκεί μέχρι να ληφθεί η απάντηση, οπότε και τερματίζεται.

Το σύνολο των ενεργών controllers υλοποιείται ως μία διπλά συνδεδεμένη λίστα, ενώ για τον κάθε ένα ορίζεται και μία τιμή βάρους, η οποία αντιστοιχεί στον αριθμό των αιτήσεων που θα στείλει η συσκευή πριν αρχίσει να χρησιμοποιεί τον επόμενο στη λίστα controller.

Client -> Controller	Controller -> Client
REGISTER	REGISTRATION_DENIED REGISTRATION_ACCEPTED
PING	PING_REPLY
GET_STAT	STAT

Πίνακας 5.1 – Μηνύματα επιπέδου ελέγχου

5.2.2 Επίπεδο Μηνυμάτων Ελέγχου

Τα μηνύματα ελέγχου ακολουθούν το πρωτόκολλο vNBD το οποίο δημιουργήθηκε ειδικά για τη συγκεκριμένη συσκευή και βρίσκεται πάνω από το πρωτόκολλο UDP. Όπως και στην περίπτωση του επιπέδου ανταλλαγής δεδομένων, έτσι και εδώ θα υπάρχει ένα UDP socket για κάθε controller που θα βρίσκεται στον πίνακα της συσκευής. Σε αντίθεση όμως με τα TCP sockets, τα UDP socket θα δημιουργούνται μέσα στον πυρήνα κάθε φορά που ένας νέος controller εισάγεται στον πίνακα.

Επιπλέον, θα πρέπει να υπάρχει και ένα UDP socket στο οποίο θα λαμβάνονται τα μηνύματα που στέλνουν οι controllers προς τη συσκευή. Το socket αυτό δημιουργείται στον χώρο χρήστη από την εφαρμογή vNBD-client , η οποία στη συνέχεια το περνάει στον πυρήνα κατά την αρχικοποίηση της συσκευής.

Τα μηνύματα ελέγχου διακρίνονται σε δύο κατηγορίες, ανάλογα με το ποιος είναι ο αποστολέας. Στον πίνακα 5.1 αναφέρονται οι διάφοροι τύποι μηνυμάτων του επιπέδου ελέγχου ανά κατηγορία. Συγκεκριμένα διακρίνονται οι παρακάτω τύποι μηνυμάτων :

- REGISTER : Ο client στέλνει αίτηση σύνδεσης στον controller
- PING : Τα μηνύματα αυτά τα στέλνει η συσκευή στους controllers που χρησιμοποιούνται για την ανταλλαγή δεδομένων, για να ελέγχει αν αυτοί είναι ακόμα σε λειτουργία
- GET_STAT : Ο client ζητά από τον controller να του στείλει στατιστικά στοιχεία σχετικά με τη λειτουργία του
- REGISTRATION_DENIED : Ο controller αποδέχεται τη σύνδεση με τον client
- REGISTRATION_ACCEPTED : Ο controller δεν αποδέχεται τη σύνδεση με τον client
- PING_REPLY : Ο controller απαντά με ένα τέτοιο μήνυμα κάθε φορά που λαμβάνει PING από τη συσκευή
- STAT : Αποστολή στατιστικών στοιχείων για τη λειτουργία του controller

Όμως, εκτός από την επικοινωνία του client με τον controller, θα πρέπει να οριστεί και ο τρόπος επικοινωνίας ανάμεσα στη συσκευή και τον χρήστη, δηλαδή ουσιαστικά ο τρόπος με τον οποίο θα ελέγχεται η συσκευή. Αυτό ακριβώς διαπραγματεύεται η επόμενη παράγραφος.

Κλήσεις `ioctl`	
SET_INST_SOCKET	VNBD_SET_BLKSIZE
SET_DATA_SOCK	VNBD_SET_SIZE
READ_INST_SOCK	VNBD_SET_SIZE_BLOCKS
GET_RECV_INST_PID	VNBD_DO_IT
PRINT_CONTROLLERS	VNBD_SET_BLKSIZE

Πίνακας 5.2 – Κλήσεις ioctl της συσκευής vNBD

5.3 Διαχείριση Συσκευής

Η διαχείριση της συσκευής vNBD περιλαμβάνει την αρχικοποίηση της, τον τερματισμό της λειτουργίας της και κυρίως τη διαχείριση του πίνακα με τους controllers που διατηρεί. Όλες αυτές οι λειτουργίες γίνονται μέσω της εφαρμογής vNBD-client, η οποία χρησιμοποιεί κλήσεις ioctl αλλά και netlink sockets για να επικοινωνεί με τη συσκευή. Στην παράγραφο αυτή δίνονται συνοπτικά οι κλήσεις ioctl και τα μηνύματα του netlink πρωτοκόλλου που υποστηρίζονται από τη συσκευή. Η λεπτομερής περιγραφή τους και ο τρόπος χρήσης τους αναβάλλονται για την παράγραφο 5.4 οπού παρουσιάζονται αναλυτικά τα σενάρια χρήσης της συσκευής.

5.3.1 Κλήσεις ioctl

Η πλειοψηφία των συσκευών που υπάρχουν στο Linux, παρέχουν στον χρήστη κλήσεις ioctl για την διαχείριση τους. Η συσκευή vNBD δεν αποτελεί εξαίρεση και παρέχει στον χρήστη τις εξής κλήσεις ioctl :

- SET_INST_SOCKET: Αρχικοποίηση του UDP socket στο οποίο η συσκευή θα λαμβάνει τα μηνύματα ελέγχου από τους controllers. Το socket δημιουργείται στον χώρο χρήστη και στη συνέχεια με την κλήση αυτή, η διεργασία το περνά στον πυρήνα.
- SET_DATA_SOCK: Για κάθε controller θα πρέπει η διεργασία χώρου χρήστη να δημιουργήσει ένα TCP socket και στη συνέχεια να το περάσει στον πυρήνα με αυτή την κλήση.
- READ_INST_SOCK: Θα πρέπει να κληθεί ώστε να δημιουργηθεί μία διεργασία υπεύθυνη να λαμβάνει τα μηνύματα ελέγχου που φτάνουν από τους controllers.
- GET_RECV_INST_PID: Επιστρέφει το id της διεργασίας που λαμβάνει τα μηνύματα ελέγχου.
- PRINT_CONTROLLERS: Τυπώνει τον πίνακα των controllers που διατηρεί η συσκευή.
- VNBD_SET_BLKSIZE: Θέτει το μέγεθος των block που θα χρησιμοποιεί η συσκευή
- VNBD_SET_SIZE: Θέτει το μέγεθος της συσκευής σε bytes.

- VNBD_SET_SIZE_BLOCKS: Θέτει το μέγεθος της συσκευής σε blocks.
- VNBD_DO_IT: Θα πρέπει να κληθεί για τον κάθε controller ώστε να δημιουργηθεί μία διεργασία υπεύθυνη να λαμβάνει τα μηνύματα δεδομένων που φτάνουν από τον συγκεκριμένο controller.

5.3.2 Netlink sockets

Τα netlink sockets παρέχουν μία διεπαφή για την επικοινωνία ανάμεσα στον πυρήνα και τις διεργασίες χώρου χρήστη. Η επικοινωνία μπορεί να είναι αμφίδρομη, ενώ τουλάχιστον σε ότι αφορά τον χώρο χρήστη, δεν υπάρχει διαφορά από τα υπόλοιπα sockets. Αρχικά θα γίνει μία σύντομη περιγραφή των netlink sockets και στη συνέχεια θα δοθούν οι λεπτομέρειες του netlink πρωτοκόλλου που υποστηρίζονται από τη συσκευή.

5.3.2.1 Γενικά για τα Netlink sockets

Τα netlink sockets χρησιμοποιούν την οικογένεια διευθύνσεων AF_NETLINK, σε αντίθεση για παράδειγμα με τα TCP/IP sockets που χρησιμοποιούν την AF_INET, ενώ θα πρέπει να έχει οριστεί και ο τύπος του πρωτοκόλλου στο αρχείο `<linux/netlink.h>`.
Τα netlink sockets παρουσιάζουν μία σειρά από πλεονεκτήματα έναντι του ioctl που παραδοσιακά χρησιμοποιείται για την διαχείρηση συσκευών. Το πιο σημαντικό από αυτά είναι ότι προσφέρει τη δυνατότητα ασύγχρονης επικοινωνίας με τον πυρήνα. Όπως σε όλα τα socket APIs έτσι και στα netlink sockets τα μηνύματα τοποθετούνται σε ουρές και η διαχείρισή τους μπορεί να αναβληθεί για κάποια επόμενη χρονική στιγμή. Αντίθετα, οι κλήσεις συστήματος δεν είναι ασύγχρονές. Κατά συνέπεια, αν χρησιμοποιηθούν κλήσεις συστήματος για να στέλνονται τα μηνύματα στον πυρήνα, μπορεί εύκολα να επηρεαστεί η απόδοση της συσκευής, εάν ο χρόνος επεξεργασίας είναι μεγάλος.
Ακόμα, οι κλήσεις συστήματος και το ioctl δεν επιτρέπουν στον πυρήνα να ξεκινήσει την επικοινωνία με μία διεργασία στον χώρο χρήστη. Έτσι, αν ο πυρήνας θέλει να ενημερώσει μία διεργασία για κάποια αλλαγή στη συσκευή θα πρέπει η ίδια η διεργασία να ελέγχει περιοδικά τη συσκευή προκειμένου να ενημερωθεί για την αλλαγή.

5.3.2.2 Netlink Πρωτόκολλο της Συσκευής

Η συσκευή υποστηρίζει δυο ειδών μηνύματα, τα μηνύματα *stat* τα οποία ζητούν από τη συσκευή να στείλει ένα μήνυμα GET_STAT σε κάποιον controller και τα μηνύματα *action* τα οποία χρησιμοποιούνται αποκλειστικά για τη διαχείριση του πίνακα των controllers.
Τα τελευταία δίνουν στην εφαρμογή vNBD-client τη δυνατότητα να εκτελέσει μία από τις παρακάτω ενέργειες για μία εγγραφή του πίνακα:

- Εισαγωγή ενός controller στον πίνακα
- Διαγραφή ενός controller από τον πίνακα
- Αποστολή αίτησης σύνδεσης προς έναν controller
- Κατάργηση της σύνδεσης με έναν controller
- Κατάργηση της σύνδεσης με έναν controller και διαγραφή του από τον πίνακα

Κατά συνέπεια ορίζονται πέντε διαφορετικά μηνύματα action, ένα για την κάθε λειτουργία:

- ACTION_INSERT
- ACTION_DELETE
- ACTION_REGISTER
- ACTION_DEREGISTER
- ACTION_DEREGDELETE

5.4 Παραδείγματα Χρήσης/Λειτουργίας της Συσκευής

5.4.1 Έναρξη Λειτουργίας της Συσκευής

Αρχικά δημιουργείται και αρχικοποιείται στον χώρο χρήστη ένα UDP socket το οποίο ακούει σε μία προκαθορισμένη θύρα. Στη συνέχεια με κλήση της ioctl- SET_INST_SOCKET το socket περνάει στον πυρήνα όπου πλέον το διαχειρίζεται η συσκευή. Στο socket αυτό, το οποίο θα ονομάσουμε *inst_socket,* θα φτάνουν τα μηνύματα του επιπέδου ελέγχου από τους controllers. Η διαδικασία ολοκληρώνεται με τη δημιουργία μίας διεργασίας χώρου χρήστη η οποία καλεί την ioctl- READ_INST_SOCK και είναι υπεύθυνη για την ανάγνωση των μηνυμάτων του επιπέδου ελέγχου.

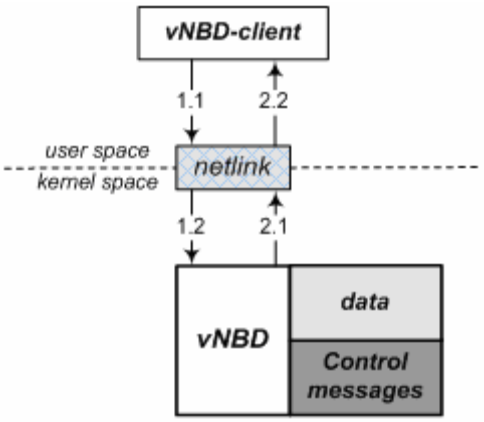
5.4.2 Τερματισμός Λειτουργίας της Συσκευής

Κατά τον τερματισμό λειτουργίας της συσκευής, η διεργασία που είχε δημιουργηθεί για να λαμβάνει τα μηνύματα του επιπέδου ελέγχου τερματίζεται και το *inst_socket* κλείνει. Το pid της διεργασίας λαμβάνεται με την κλήση ioctl- GET_RECV_INST_PID.

5.4.3 Εισαγωγή Controller

Για να εισάγει ο χρήστης έναν νέο controller στον πίνακα της συσκευής, θα πρέπει να δώσει κατά την εισαγωγή τρία στοιχεία:

1. την διεύθυνση IP του υπολογιστή στον οποίο βρίσκεται ο controller
2. τον αριθμό θύρας στον οποίο «ακούει» τα μηνύματα ελέγχου ο controller (δηλ. η διεργασία *control_msg_handler*)
3. το βάρος για τον συγκεκριμένο controller



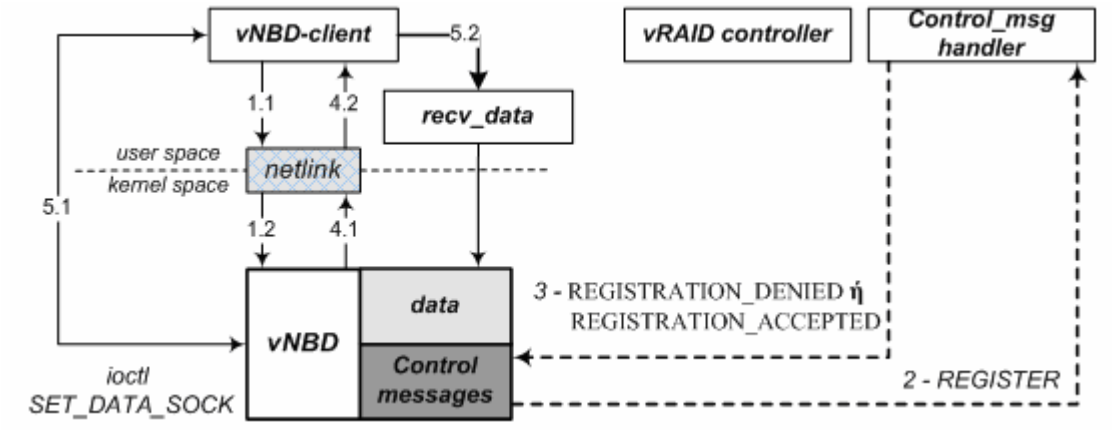
Σχήμα 5.2 – Εισαγωγή/Διαγραφή controller. Και στις δύο αυτές περιπτώσεις το μήνυμα *action* στέλνεται στη συσκευή η οποία το επεξεργάζεται και απαντά με ένα ίδιο μήνυμα. Στην περίπτωση της εισαγωγής το μήνυμα που επιστρέφεται περιέχει το id του controller.

Στη συνέχεια η εφαρμογή vNBD-client στέλνει στη συσκευή ένα μήνυμα ACTION_INSERT και η συσκευή αναλαμβάνει να δημιουργήσει ένα UDP socket και να αρχικοποιήσει μία δομή `sockaddr_in`. Αν δεν παρουσιαστούν σφάλματα κατά τις αρχικοποιήσεις, ένα μήνυμα ACTION_INSERT στέλνεται πίσω στην εφαρμογή χώρου χρήστη και ενημερώνει τον χρήστη για την επιτυχία εισαγωγής νέου controller, γνωστοποιώντας του ταυτόχρονα τη θέση του στο πίνακα (id του contoller). Αν έχουν παρουσιαστεί σφάλματα, το μήνυμα περιέχει τον κωδικό σφάλματος. Συγκεκριμένα, επιστρέφεται *ENOFID* στην περίπτωση που δεν υπάρχει θέσει στον πίνακα, καθώς η συσκευή υποστηρίζει έναν μέγιστο αριθμό από controllers. Αντίθετα, εάν η συσκευή δεν μπορέσει να δημιουργήσει το UDP socket επιστρέφεται ο κωδικός σφάλματος *ECINSTS*.

5.4.4 Διαγραφή Controller

Στην περίπτωση αυτή η εφαρμογή στέλνει στη συσκευή ένα μήνυμα ACTION_DELETE το οποίο περιέχει το id του controller που πρέπει να διαγραφεί. Η συσκευή λαμβάνοντας το μήνυμα, κλείνει το UDP socket εφόσον κάτι τέτοιο μπορεί να συμβεί. Θα πρέπει να σημειωθεί ότι δεν επιτρέπεται η διαγραφή ενός controller, όταν αυτός χρησιμοποιείται από τη συσκευή. Στην περίπτωση αυτή θα πρέπει να χρησιμοποιηθεί ένα από τα μηνύματα ACTION_DEREGISTER ή ACTION_DEREGDELETE που περιγράφονται παρακάτω, έτσι ώστε ο controller να έχει πρώτα αποσυνδεθεί.

Στην περίπτωση που ο χρήστης προσπαθήσει να διαγράψει έναν controller που χρησιμοποιείται από τη συσκευή επιστρέφεται ο κωδικός σφάλματος *EILLACT* (*E*rror *ILL*egal *A*ction). Επιπλέον, εάν το id έχει τιμή εκτός των ορίων του πίνακα controllers της συσκευής επιστρέφεται ο κωδικός σφάλματος *EINVID* (*E*rror *INV*alid *ID*).



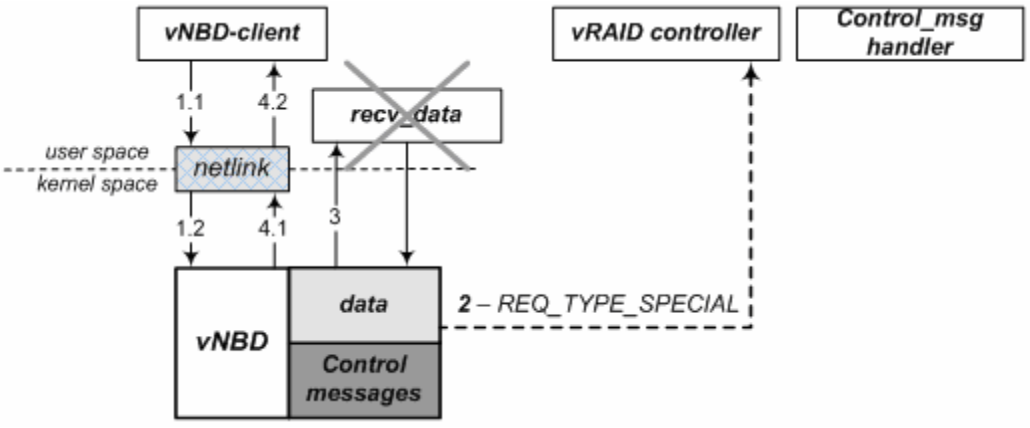
Σχήμα 5.3 – Αποστολή αίτησης σύνδεσης. Το βήμα 5 (5.1 και 5.2) δεν εκτελείται στην περίπτωση που ο controller απαντήσει με REGISTRATION_DENIED ή δεν απαντήσει καθόλου.

5.4.5 Αποστολή Αίτησης Σύνδεσης

Η σύνδεση με έναν controller γίνεται με τα παρακάτω βήματα:

1. Η εφαρμογή vNBD-client στέλνει ένα μήνυμα ACTION_REGISTER μέσω του netlink socket στη συσκευή, με παράμετρο το id του controller με τον οποίο θέλει να συνδεθεί.
2. Η συσκευή στέλνει ένα μήνυμα ελέγχου REGISTER προς τον controller μέσω του UDP socket που είχε δημιουργήσει κατά την εισαγωγή του στον πίνακα.
3. Ο controller απαντά με ένα μήνυμα REGISTRATION_DENIED ή REGISTRATION_ACCEPTED. Στην δεύτερη περίπτωση το μήνυμα περιέχει και τον αριθμό της θύρας στην οποίο ο controller δέχεται τα μηνύματα δεδομένων.
4. Η συσκευή ενημερώνει την εφαρμογή vNBD-client σχετικά με την απάντηση του controller. Αν ο controller αποδέχεται τη σύνδεση, η συσκευή στέλνει στην εφαρμογή τον αριθμό της θύρας δεδομένων και την διεύθυνση IP του controller.
5. Στην περίπτωση που ο controller αποδέχεται τη σύνδεση, η εφαρμογή χρήστη δημιουργεί ένα TCP socket το οποίο περνά στον πυρήνα με κλήση της ioctl – SET_DATA_SOCK. Αν η κλήση ioctl δεν επιστρέψει κάποιο σφάλμα, δημιουργείται και μία νέα διεργασία η οποία καλεί την ioctl - VNBD_DO_IT και είναι υπεύθυνη για να λαμβάνει τα μηνύματα του επιπέδου ανταλλαγής δεδομένων.

Θα πρέπει να σημειωθεί ότι αν στο βήμα 3 ο controller δεν απαντήσει μέσα σε χρόνο T η συσκευή στέλνει ένα νέο μήνυμα REGISTER. Αν και μετά από 3 προσπάθειες δε λάβει απάντηση η εφαρμογή χρήστη ενημερώνεται ότι ο controller δεν απάντησε καθώς η συσκευή επιστρέφει *ETIMEOUT*. Στην περίπτωση που το id του controller δεν έχει έγκυρη τιμή επιστρέφεται *EINVID* , ενώ εάν ο controller είναι ήδη συνδεδεμένος και ο χρήστης επιχειρήσει να εκτελέσει τη συγκεκριμένη λειτουργία η συσκευή επιστρέφει *EILLACT* .



Σχήμα 5.4 – Κατάργηση της σύνδεσης με έναν controller.

5.4.6 Κατάργηση Σύνδεσης

Η κατάργηση της σύνδεσης με έναν controller γίνεται με τα παρακάτω βήματα:

1. Η εφαρμογή vNBD-client στέλνει ένα μήνυμα ACTION_DEREGISTER μέσω του netlink socket στη συσκευή, με παράμετρο το id του controller τον οποίο θέλει να σταματήσει να χρησιμοποιεί.
2. Μία ειδική αίτηση REQ_TYPE_SPECIAL αποστέλλεται στον controller και η επικοινωνία για την ανταλλαγή δεδομένων διακόπτεται
3. Η διεργασία *recv_data* τερματίζει καθώς επιστρέφει η κλήση ioctl - VNBD_DO_IT
4. Όλες οι αιτήσεις ανάγνωσης και εγγραφής που έχουν σταλεί στον συγκεκριμένο controller τερματίζονται από τη συσκευή χωρίς φυσικά να έχουν ολοκληρωθεί
5. Η συσκευή ενημερώνει την εφαρμογή χρήστη ότι η σύνδεση έχει διακοπεί ή επιστρέφει έναν κωδικό λάθους αν έχει παρουσιαστεί κάποιο σφάλμα στην διαδικασία αποσύνδεσης. Συγκεκριμένα επιστρέφεται *EINVID* στην περίπτωση που το id του controller δεν έχει έγκυρη τιμή, *EILLACT* εάν ο controller δεν είναι συνδεδεμένος και ο χρήστης επιχειρήσει να εκτελέσει τη συγκεκριμένη λειτουργία και *ESENDD* εάν τη δεδομένη χρονική στιγμή η συσκευή χρησιμοποιεί τον controller για την αποστολή δεδομένων.

5.4.7 Κατάργηση Σύνδεσης και Διαγραφή Controller

Στην περίπτωση αυτή η εφαρμογή στέλνει στη συσκευή ένα μήνυμα ACTION_DEREGDELETE το οποίο περιέχει το id του controller που πρέπει να αποσυνδεθεί και να διαγραφεί. Οι λειτουργίες που γίνονται και τα βήματα που ακολουθούνται είναι αυτά που περιγράφονται στα σενάρια 5.4.6 (κατάργηση σύνδεσης) και 5.4.4 (διαγραφή controller). Οι κωδικοί σφάλματος που μπορεί να επιστρέψει η συσκευή είναι ίδιοι με αυτούς των προηγούμενων σεναρίων, ενώ και πάλι για να μπορεί η συγκεκριμένη λειτουργία να

εκτελεστεί για των controller id, θα πρέπει ο controller να είναι ήδη συνδεδεμένος και να μη χρησιμοποιείται εκείνη τη χρονική στιγμή από τη συσκευή για την αποστολή δεδομένων.

5.4.8 Εκτύπωση Πίνακα Ελεγκτών

Η εκτύπωση του πίνακα ελεγκτών γίνεται με κλήση στην ioctl - PRINT_CONTROLLERS. Η κλήση αυτή επιστρέφει μέσω της παραμέτρου arg έναν πίνακα, τον οποίο στη συνέχεια η εφαρμογή vNBD-client τυπώνει στην οθόνη. Συγκεκριμένα, για τον κάθε controller τυπώνονται τα παρακάτω στοιχεία:

- η διεύθυνση IP του controller,
- η θύρα στην οποία ο controller ακούει για μηνύματα ελέγχου,
- η θύρα στην οποία ο controller ακούει για μηνύματα δεδομένων (αιτήσεις Ε/Ε), εάν έχει γίνει η σύνδεση με τον controller,
- το βάρος που έχει οριστεί για τον controller,
- το id του controller και
- η κατάσταση στην οποίο βρίσκεται ο controller και μπορεί να έχει μία από τις τιμές του πίνακα 5.3.

Κατάσταση	***Περιγραφή***
REGISTERED	Έχει γίνει η σύνδεση με τον controller, ο οποίος τώρα μπορεί να χρησιμοποιείται από τη συσκευή για την αποστολή αιτήσεων Ε/Ε.
NOT_REGISTERED	Δεν έχει γίνει ακόμα σύνδεση με τον controller.
IN_REGISTRATION	Δεν έχει γίνει ακόμα σύνδεση με τον controller αλλά η συσκευή έχει στείλει αίτηση σύνδεσης.

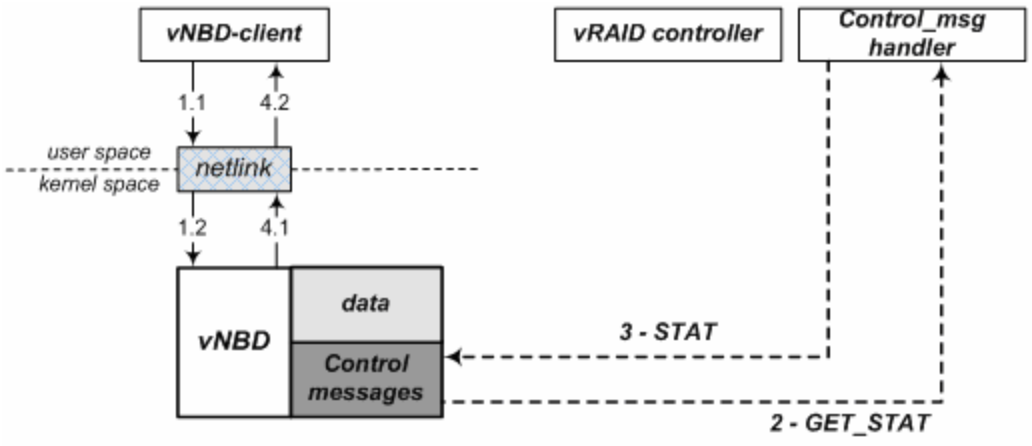
Πίνακας 5.3 – Οι τιμές που μπορεί να πάρει το πεδίο κατάστασης ενός controller

5.4.9 Έλεγχος Λειτουργίας Controller

Σύμφωνα με τις προδιαγραφές που δόθηκαν στην αρχή του κεφαλαίου, αν κάποιος από τους ελεγκτές (controllers) σταματήσει για κάποιο λόγο να λειτουργεί, η συσκευή θα πρέπει να πάψει να τον χρησιμοποιεί. Για το λόγο αυτό η συσκευή στέλνει σε τακτά χρονικά διαστήματα μηνύματα PING προς τους ελεγκτές, ενώ οι τελευταίοι απαντούν με ένα μήνυμα PING_REPLY. Εάν η συσκευή δε λάβει απάντηση σε τρία συνεχόμενα μηνύματα PING, η σύνδεση με τον controller διακόπτεται και η συσκευή σταματά να τον χρησιμοποιεί.

5.4.10 Λήψη Στατιστικών Controller

Όταν ο χρήστης θέλει να λάβει στατιστικά στοιχεία για έναν controller, θα πρέπει να στείλει μέσω της εφαρμογής vNBD-client ένα μήνυμα *stat* στη συσκευή. Στη συνέχεια, η συσκευή



Σχήμα 5.5 – Λήψη στατιστικών από έναν controller.

στέλνει ένα μήνυμα GET_STAT στον controller, ο οποίος απαντά με την αποστολή των στοιχείων. Συγκεκριμένα στέλνει 3 τιμές :

1. το χρόνο που ο controller βρίσκεται σε λειτουργία,
2. τον αριθμό των clients που χρησιμοποιούν τον συγκεκριμένο controllet και
3. τον αριθμό των μηνυμάτων δεδομένων που λαμβάνει στη μονάδα του χρόνου.

Στην παρούσα υλοποίηση τα μηνύματα *stat* δε χρησιμοποιούνται και ο controller απαντά στέλνοντας κάποιες προκαθορισμένες τιμές. Παρόλ' αυτά έχουν συμπεριληφθεί στην υλοποίηση για την πληρότητα της.

5.4.11 Εγγραφή και Ανάγνωση Δεδομένων

Όπως έχει ήδη αναφερθεί, κάθε συσκευή block διαθέτει μία ουρά στην οποία τοποθετούνται οι αιτήσεις για ανάγνωση και εγγραφή. Οι εξυπηρέτηση αυτών των αιτήσεων γίνεται μέσω μίας προκαθορισμένης συνάρτησης του οδηγού, της do_request, η οποία καλείται σε τακτά χρονικά διαστήματα από τον πυρήνα.

Για κάθε αίτηση της ουράς, καθορίζεται από τη συσκευή ο controller που θα αναλάβει την εξυπηρέτησή της και η συσκευή στέλνει ένα TCP μήνυμα προς τον controller. Στη συνέχεια, η αίτηση τοποθετείται στην ουρά που αντιστοιχεί στον συγκεκριμένο controller και παραμένει εκεί μέχρι να ληφθεί η απάντηση, οπότε και τερματίζεται.

6 *Υλοποίηση της συσκευής*

Στο κεφάλαιο αυτό περιγράφονται λεπτομέρειες που αφορούν στην υλοποίηση της συσκευής. Συγκεκριμένα, παρουσιάζονται αρχικά οι κύριες δομές δεδομένων που χρησιμοποιεί η συσκευή ενώ στη συνέχεια δίνονται οι λεπτομέρειες της υλοποίησης του επιπέδου ανταλλαγής δεδομένων και του επιπέδου μηνυμάτων ελέγχου. Τέλος, περιγράφεται και ο τρόπος διαχείρισης της συσκευής από την εφαρμογή vNBD-client.

6.1 Κύριες Δομές της Συσκευής

Δύο είναι οι κυριότερες δομές της συσκευής vNBD. Η πρώτη ονομάζεται `controller_struct` και χρησιμοποιείται για την αποθήκευση όλων των πληροφοριών που αφορούν έναν συγκεκριμένο controller ενώ η δεύτερη ονομάζεται `vnbd_dev` και αναπαριστά την ίδια τη συσκευή.

6.1.1 Η Δομή controller_struct

Όπως φαίνεται και στον πίνακα 6.1, σε αυτήν τη δομή αποθηκεύονται η διεύθυνση ip του controller, το βάρος για τον συγκεκριμένο controller καθώς και οι θύρες που χρησιμοποιεί για την ανταλλαγή δεδομένων και μηνυμάτων ελέγχου, ενώ υπάρχουν και δύο socket ένα για το κάθε επίπεδο επικοινωνίας πελάτη-controller.
Επιπλέον, η δομή περιέχει μία ουρά στην οποία η συσκευή τοποθετεί τις αιτήσεις που έχει στείλει προς το συγκεκριμένο controller. Όταν θα ληφθεί η απάντηση, η συσκευή θα βρει την αντίστοιχη αίτηση μέσα στην ουρά και αφού την εξάγει θα την τερματίσει κατάλληλα.
Η συσκευή διατηρεί εσωτερικά έναν πίνακα από δομές `controller_struct`. Η θέση του κάθε controller μέσα στον πίνακα, τον προσδιορίζει μοναδικά και αποτελεί το αναγνωριστικό του (id). Παράλληλα, τα πεδία next και prev δίνουν τη δυνατότητα στη συσκευή να

Τύπος Πεδίου	*Όνομα Πεδίου*	*Περιγραφή*
`unsigned long int`	`ip`	η διεύθυνση ip του controller
`int`	`data_port`	το μέγεθος του τμήματος σε bytes
`int`	`inst_port`	το σημείο στο οποίο ξεκινά το τμήμα
`struct socket *`	`data_sock`	TCP socket για την ανταλλαγή δεδομένων
`struct file *`	`file`	δείκτης στη δομή file του TCP socket
`struct socket *`	`inst_sock`	UDP socket για την αποστολή μηνυμάτων ελέγχου
`struct sockaddr_in`	`inst_addr`	η δομή sockaddr_in που αντιστοιχεί στο UDP socket
`spinlock_t`	`tx_lock`	κλειδί για τη διαχείριση της δομής
`atomic_t`	`status`	η κατάσταση της δομής
`atomic_t`	`sending_data`	έχει την τιμή 1 για όσο χρονικό διάστημα η συσκευή στέλνει δεδομένα στον controller
`atomic_t`	`ping_err`	ο αριθμός των μηνυμάτων ping στα οποία ο contoller δεν απάντησε
`spinlock_t`	`queue_lock`	κλειδί για τη διαχείριση της ουράς των αιτήσεων
`struct list_head`	`queue_head`	η ουρά των αιτήσεων που έχουν σταλεί στον controller
`struct request *`	`active_req`	η αίτηση που επεξεργάζεται η συσκευή και αφορά τον συγκεκριμένο controller
`int`	`weight`	το βάρος για τον συγκεκριμένο controller
`int`	`times_used`	ο αριθμός των διαδοχικών αιτήσεων που έχουν σταλεί από τη συσκευή στον controller
`int`	`next`	η συσκευή χρησιμοποιεί τα πεδία αυτά για να υλοποιεί λίστες
`int`	`prev`	

Πίνακας 6.1 – Τα πεδία της δομής `controller_struct`

δημιουργεί και να διατηρεί κάποιες λίστες οι οποίες όμως ουσιαστικά υλοποιούνται από τον πίνακα. Ο μοναδικός περιορισμός που υπάρχει είναι ότι σε κάθε χρονική στιγμή μία δομή θα μπορεί να ανήκει σε μία μόνο λίστα από αυτές που διατηρεί η συσκευή.

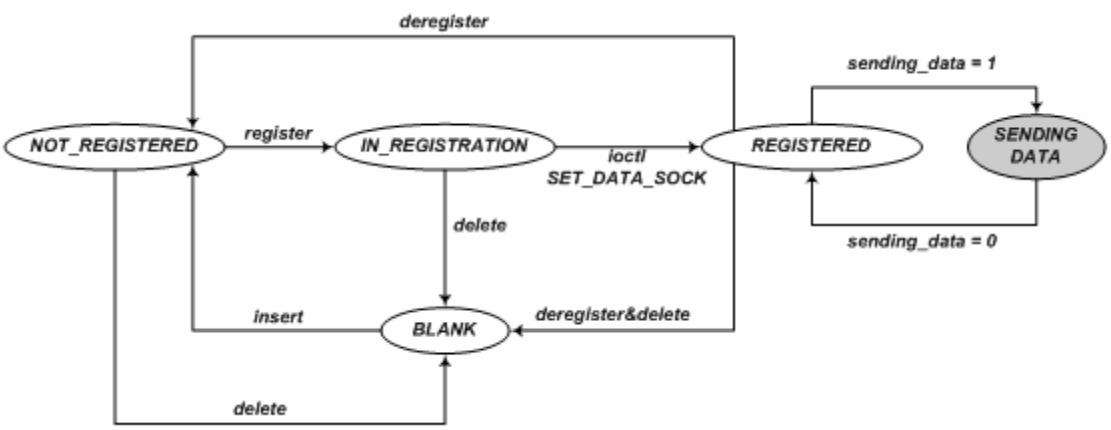
Αξίζει να σημειωθεί ότι σε κάθε δεδομένη χρονική στιγμή μία δομή αυτού του τύπου μπορεί να βρίσκεται σε μία κατάσταση η οποία περιγράφεται από τα πεδία status και sending_data. Η κατάσταση στην οποία βρίσκεται μία δομή, δηλαδή μία εγγραφή του πίνακα, ορίζει και ποιες ενέργειες μπορούν να εκτελεστούν σε αυτήν. Όπως έχει ήδη αναφερθεί, οι ενέργειες που μπορούν να εκτελεστούν είναι οι εξής:

- Εισαγωγή ενός controller στον πίνακα (insert)
- Διαγραφή ενός controller από τον πίνακα (delete)
- Αποστολή αίτησης σύνδεσης προς έναν controller (register)
- Κατάργηση της σύνδεσης με έναν controller (deregister)
- Κατάργηση σύνδεσης και διαγραφή του controller (deregister&delete)

Στο σύνολο αυτών των λειτουργιών θα πρέπει να προστεθεί και η κλήση ioctl – SET_DATA_SOCK με την οποία αρχικοποιείται το TCP socket της δομής. Στον πίνακα 6.2

Κατάσταση	*Περιγραφή*
NOT_AVAILABLE	Η συσκευή εκτελεί κάποια ενέργεια πάνω στη δομή και η ενέργεια αυτή δεν έχει ακόμα ολοκληρωθεί
BLANK	Η δομή δε χρησιμοποιείται από τη συσκευή, επομένως μπορεί να χρησιμοποιηθεί για την εισαγωγή ενός νέου controller
REGISTERED	Έχει γίνει η σύνδεση με τον controller, ο οποίος τώρα μπορεί να χρησιμοποιείται από τη συσκευή για την αποστολή αιτήσεων E/E.
NOT_REGISTERED	Δεν έχει γίνει ακόμα σύνδεση με τον controller.
IN_REGISTRATION	Δεν έχει γίνει ακόμα σύνδεση με τον controller αλλά η συσκευή έχει στείλει αίτηση σύνδεσης.

Πίνακας 6.2– Οι τιμές που μπορεί να πάρει το πεδίο status της δομής controller_struct



Σχήμα 6.1 – Διάγραμμα κατάστασης της δομής controller_struct. Η κατάσταση NOT_AVAILABLE δε φαίνεται στο διάγραμμα για απλοποίηση. Η δομή περνά από την κατάσταση αυτή πριν από κάθε αλλαγή κατάστασης.

δίνονται οι τιμές που μπορεί να πάρει το πεδίο status, ενώ στο σχήμα 6.1 φαίνεται το διάγραμμα κατάστασης για μία δομή `controller_struct`.

Κάθε φορά που καλείται μία από τις συναρτήσεις που είναι υπεύθυνη για τη μετάβαση από μία αρχική κατάσταση σε κάποια άλλη, θα πρέπει να ελεγχθεί αν η αρχική κατάσταση είναι τέτοια που να επιτρέπει την συγκεκριμένη ενέργεια. Ακόμα, η δομή θα πρέπει να κλειδωθεί ώστε να αποτρέψει την αλλαγή πεδίων της από άλλες κλήσεις που ενδεχομένως να κάνει ο χρήστης.

Όλα τα παραπάνω εξασφαλίζονται με τη χρήση ενός μηχανισμού κλειδώματος της κάθε δομής. Ο μηχανισμός αυτός χρησιμοποιεί το κλειδί της δομής (πεδίο tx_lock) καθώς και τα πεδία status και sending_data, τα οποία ενημερώνονται ατομικά. Το ιδιαίτερο χαρακτηριστικό του μηχανισμού αυτού είναι ότι δεν κρατά το κλειδί της δομής μέχρι να ολοκληρωθεί η κάθε ενέργεια, επιτρέποντας έτσι σε κάποιες λειτουργίες να γίνονται παράλληλα.

```
πάρε_το_κλειδί_της_δομής;
IF status == NOT_AVAILABLE Ή status == BLANK THEN
      ελευθέρωσε_το_κλειδί_της_δομής;
      RETURN –EILLACT;
ELSE
      IF status ∈ A THEN
            status = NOT_AVAILABLE;
            ελευθέρωσε_το_κλειδί_της_δομής;
            { εκτέλεση του κώδικα που αφορά τη συγκεκριμένη περίπτωση }
            status = ΤΕΛΙΚΗ_ΚΑΤΑΣΤΑΣΗ; /*atomic variable*/
      ELSE
            ελευθέρωσε_το_κλειδί_της_δομής;
            RETURN –EILLACT;
      ENDIF
ENDIF
```

Σχήμα 6.2 – Ψευδοκώδικας του μηχανισμού κλειδωμάτων για τη δομή controller_struct. Το σύνολο Α είναι το σύνολο των επιτρεπτών αρχικών καταστάσεων για την κάθε λειτουργία και δίνεται στον πίνακα 6.3

Λειτουργία	***Σύνολο Αρχικών Καταστάσεων Α***	***Τελική Κατάσταση***
delete	{NOT_REGISTERED,IN_REGISTRATION}	BLANK
register	{ NOT_REGISTERED}	IN_REGISTRATION
deregister	{ REGISTERED }	NOT_REGISTERED
deregister&delete	{ REGISTERED }	BLANK
SET_DATA_SOCK	{ IN_REGISTRATION }	REGISTERED

Πίνακας 6.3 – Σύνολα αρχικών καταστάσεων και τελικές καταστάσεις για την κάθε λειτουργία.

Με εξαίρεση την λειτουργία insert, όλες οι υπόλοιπες λειτουργίες που αναφέρθηκαν παραπάνω ακολουθούν τον μηχανισμό κλειδωμάτων που φαίνεται στο σχήμα 6.2. Για τη λειτουργία insert δεν απαιτείται ο μηχανισμός αυτός, καθώς κάθε φορά που ένας νέος controller εισάγεται στον πίνακα, η συσκευή παίρνει ένα ελεύθερο id, δηλαδή μία ελεύθερη θέση στον πίνακα, από την λίστα ελεύθερων id. Οι λεπτομέρειες για τον τρόπο με τον οποίο γίνεται αυτό θα δοθούν στην παράγραφο 6.1.2.

6.1.2 Η Δομή vnbd_dev

Τα πεδία της δομής vnbd_dev δίνονται στον πίνακα 6.4. Εκτός από τον πίνακα τον controllers, η συσκευή διαθέτει και δύο λίστες. Η πρώτη είναι μία απλά συνδεδεμένη λίστα που περιέχει τις ελεύθερες θέσεις μέσα στον πίνακα, με το πεδίο freeIds να αντιστοιχεί στην κεφαλή της. Η δεύτερη είναι μία διπλά συνδεδεμένη λίστα με το πεδίο c_list να «δείχνει» στον πρώτο συνδεδεμένο controller και το πεδίο curr_c στον controller που χρησιμοποιείται

Τύπος Πεδίου	*Όνομα Πεδίου*	*Περιγραφή*
`struct gendisk *`	`disk`	δείκτης στη δομή τύπου gendisk που χρησιμοποιείται από τη συσκευή
`int`	`blksize`	χωρητικότητα της συσκευής σε blocks
`u64`	`bytesize`	χωρητικότητα της συσκευής σε bytes
`int`	`status`	η κατάσταση της συσκευής. Έχει την τιμή ACTIVE για όσο χρόνο η συσκευή βρίσκεται σε λειτουργία
`struct controller_struct`	`c[]`	ο πίνακας των controllers. Μπορεί να περιέχει MAX_CONTROLLERS το πολύ ελεγκτές
`int`	`freeIds`	η πρώτη ελεύθερη θέση στον πίνακα των controllers, αποτελεί την κεφαλή της λίστας ελεύθερων ids
`spinlock_t`	`f_lock`	κλειδί για τη διαχείριση της λίστας ελεύθερων ids
`struct socket *`	`in_sock`	το UDP socket που χρησιμοποιεί το επίπεδο μηνυμάτων ελέγχου
`struct file *`	`in_file`	δείκτης στη δομή file του UDP socket
`pid_t`	`recv_inst_pid`	το pid της διεργασίας χρήστη που ελέγχει την λήψη μηνυμάτων ανταλλαγής δεδομένων
`struct rw_semaphore`	`c_lock`	κλειδί για τη διαχείριση της λίστας των συνδεδεμένων controllers
`int`	`c_list`	η κεφαλή της λίστας συνδεδεμένων controllers
`int`	`curr_c`	ο controller που χρησιμοποιείται τη δεδομένη χρονική στιγμή από τη συσκευή
`struct sock *`	`nls`	το netlink socket
`struct task_struct *`	`nl_task`	το thread πυρήνα που διαχειρίζεται τα netlink μηνύματα που προορίζονται για τη συσκευή.
`struct task_struct *`	`ping_task`	το thread πυρήνα που ελέγχει εάν όλοι οι controllers με τους οποίους έχει γίνει σύνδεση βρίσκονται σε λειτουργία
`struct event_table`	`ev`	πίνακας γεγονότων. Ως γεγονός αναφέρεται η λήψη ενός μηνύματος του επιπέδου ελέγχου
`wait_queue_head_t`	`thread_wq`	ουρά αναμονής για τα threads του πυρήνα

Πίνακας 6.4 – Τα πεδία της δομής `vnbd_dev`

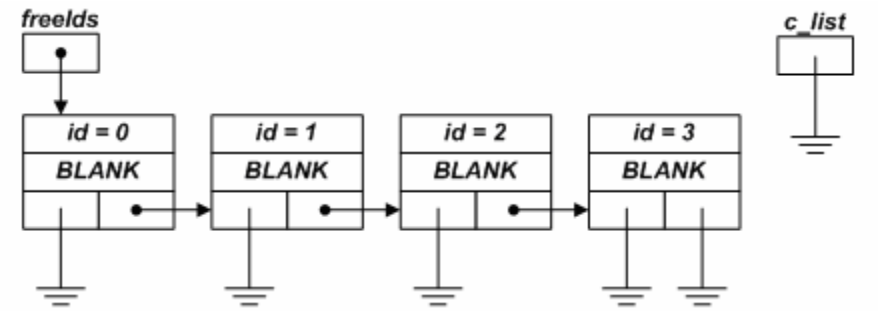
από τη συσκευή . Όπως έχει ήδη αναφερθεί η λίστες αυτές υλοποιούνται με τη βοήθεια του πίνακα των controllers και όχι με δείκτες.

Όταν η συσκευή αρχικοποιείται όλες οι θέσεις του πίνακα είναι ελεύθερες ενώ η λίστα των συνδεδεμένων controllers είναι κενή. Στο σχήμα 6.3 φαίνονται οι τιμές των πεδίων c_list και freeIds καθώς και ορισμένων πεδίων του πίνακα ελεγκτών μετά την αρχικοποίηση της συσκευής, ενώ δίνεται και μία αναπαράσταση με λίστες.

	status	next	prev
0	BLANK	1	-1
1	BLANK	2	-1
2	BLANK	3	-1
3	BLANK	-1	-1

freeIds	c_list
0	-1

Σχήμα 6.3A – Ο πίνακας των controllers και τα πεδία freeIds και c_list της συσκευής μετά την αρχικοποίηση της για MAX_CONTROLLERS=4

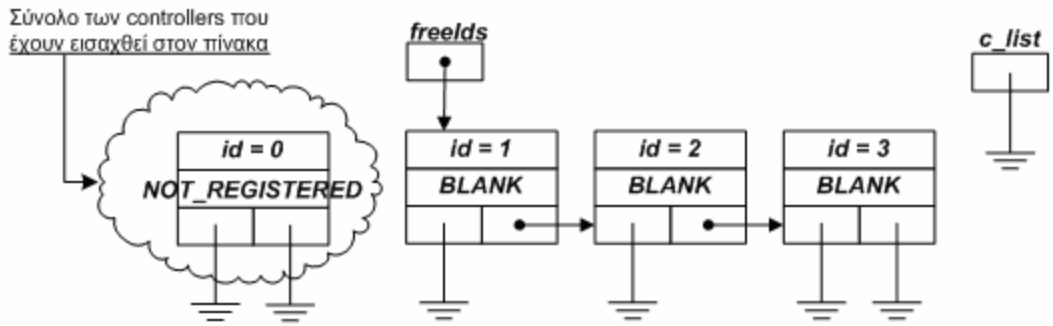


Σχήμα 6.3B – Αναπαράσταση του πίνακα και των πεδίων με λίστες

	status	next	prev
0	NOT_REGISTERED	-1	-1
1	BLANK	2	-1
2	BLANK	3	-1
3	BLANK	-1	-1

freeIds	c_list
1	-1

Σχήμα 6.4A – Ο πίνακας των controllers και τα πεδία freeIds και c_list της συσκευής μετά την εισαγωγή του πρώτου controller.



Σχήμα 6.4B – Αναπαράσταση του πίνακα και των πεδίων με λίστες

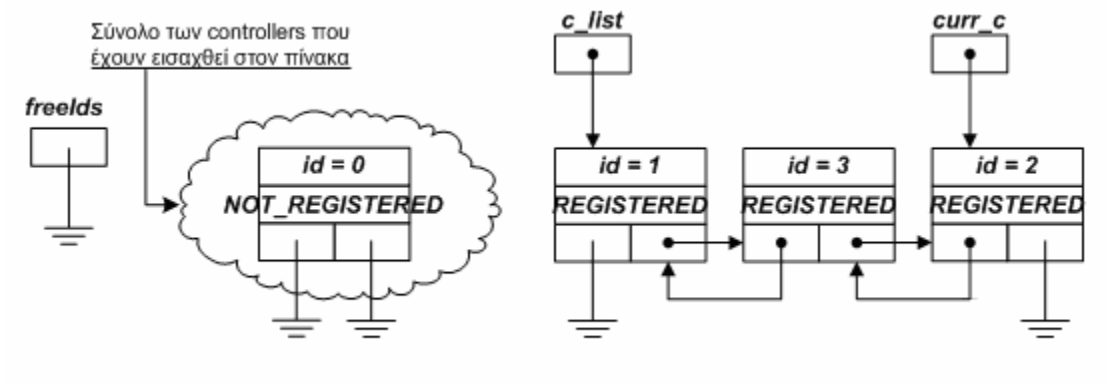
Όταν θα πρέπει να γίνει εισαγωγή ενός controller, η συσκευή παίρνει το πρώτο ελεύθερο id από την λίστα freeIds και αφού αρχικοποιήσει τη δομή θέτει την κατάστασή της στην τιμή NOT_REGISTERED. Στο σχήμα 6.4 φαίνονται οι τιμές των πεδίων μετά την εισαγωγή ενός controller.

Μετά την εισαγωγή στον πίνακα, η συσκευή μπορεί να στείλει μία αίτηση σύνδεσης στον controller και εάν δεν παρουσιαστούν σφάλματα η αντίστοιχη δομή εισάγεται στη λίστα c_list, η οποία περιέχει τους ελεγκτές που η συσκευή μπορεί να χρησιμοποιήσει για την

	status	next	prev
0	NOT_REGISTERED	-1	-1
1	REGISTERED	3	-1
2	REGISTERED	-1	3
3	REGISTERED	2	1

freeIds	c_list	curr_c
-1	1	2

Σχήμα 6.5A – Ο πίνακας των controllers και τα πεδία freeIds, c_list και curr_c της συσκευής σε μία τυχαία χρονική στιγμή.



Σχήμα 6.5B – Αναπαράσταση του πίνακα και των πεδίων με λίστες

αποστολή και λήψη δεδομένων. Στο σχήμα 6.5 παρουσιάζονται οι τιμές των διαφόρων πεδίων στην περίπτωση που έχει γίνει σύνδεση με 3 ελεγκτές.

Πέρα όμως από τον πίνακα και τις λίστες ελεγκτών η δομή vnbd_dev περιέχει έναν πίνακα γεγονότων και δύο πεδία τύπου task_struct τα οποία αντιστοιχούν σε διεργασίες πυρήνα που η κάθε μία είναι υπεύθυνη για κάποια συγκεκριμένη λειτουργία της συσκευής.

Η πρώτη διεργασία είναι υπεύθυνη για τη διαχείριση και την εξυπηρέτηση των μηνυμάτων του επιπέδου netlink, ενώ η δεύτερη αναλαμβάνει να ελέγχει σε τακτά χρονικά διαστήματα εάν οι ελεγκτές που χρησιμοποιούνται από τη συσκευή, βρίσκονται ακόμα σε λειτουργία.

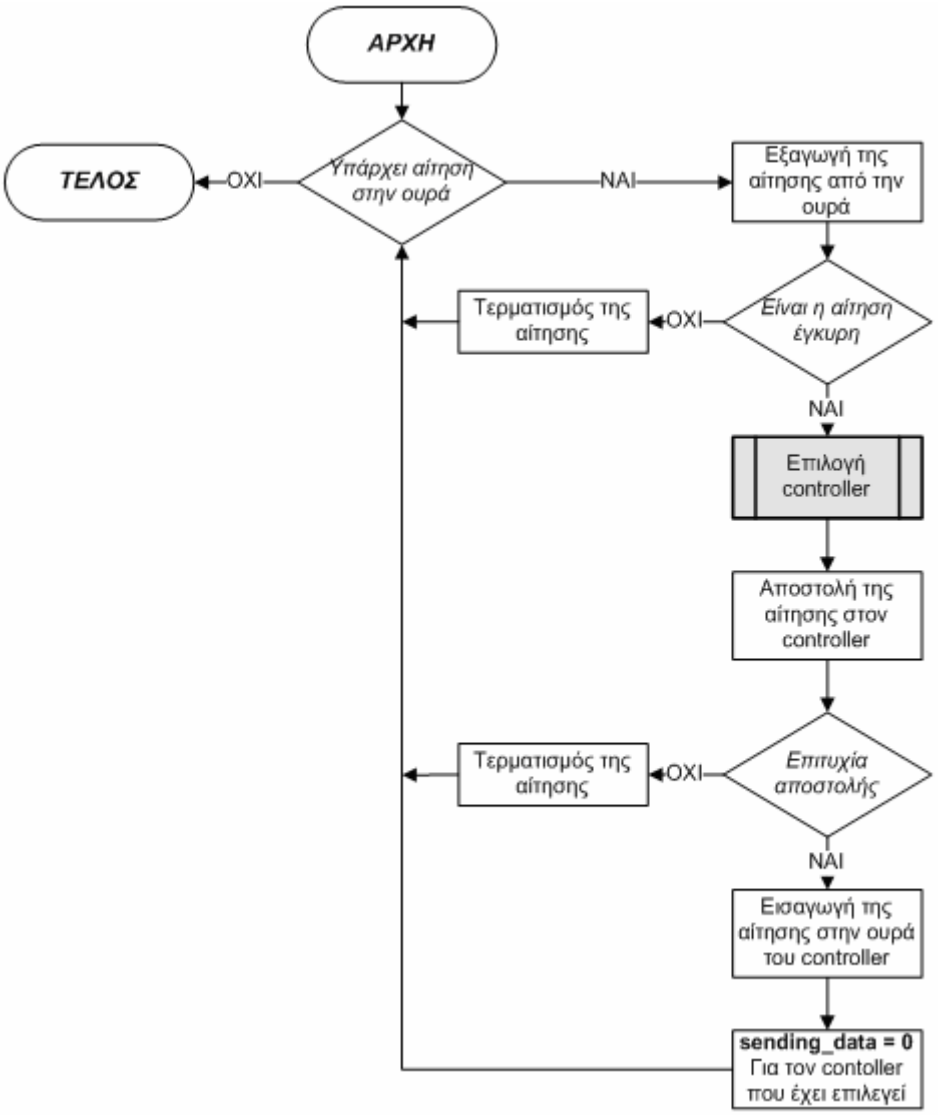
Η περιγραφή του τρόπου λειτουργίας της διεργασίας που διαχειρίζεται τα μηνύματα του επιπέδου netlink γίνεται στην παράγραφο 6.4, ενώ ο πίνακας γεγονότων και η διεργασία για τον έλεγχο των ελεγκτών περιγράφονται στην 6.3.

6.2 Επίπεδο Ανταλλαγής Δεδομένων

Το επίπεδο ανταλλαγής δεδομένων περιλαμβάνει τη συνάρτηση για την εξυπηρέτηση των αιτήσεων, το πρωτόκολλο επικοινωνίας με τους ελεγκτές και τις διεργασίες που είναι υπεύθυνες να λαμβάνουν τα μηνύματα δεδομένων στην πλευρά της συσκευής.

6.2.1 Συνάρτηση Εξυπηρέτησης Αιτήσεων

Οι αιτήσεις εγγραφής και ανάγνωσης που δημιουργούνται από τις εφαρμογές τοποθετούνται στην ουρά της συσκευής και όταν καλείται η συνάρτηση do_vnbd_request από τον πυρήνα η

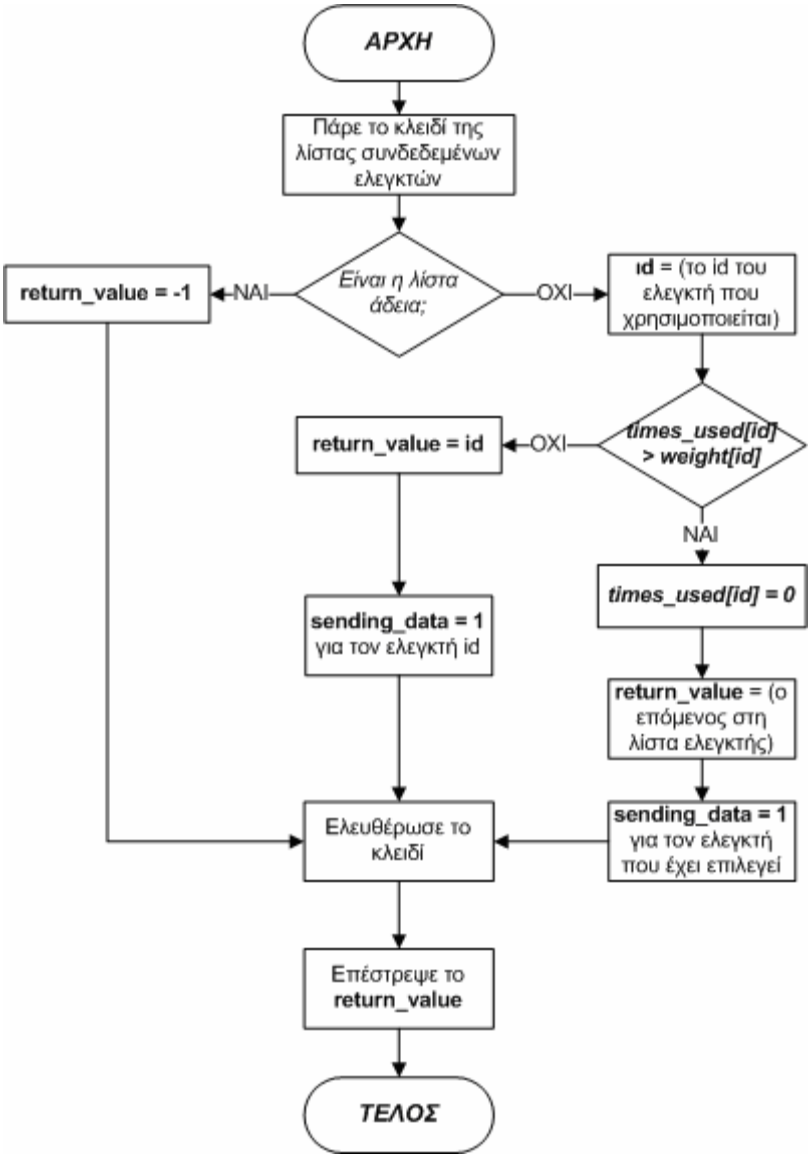


Σχήμα 6.6 – Διάγραμμα ροής της συνάρτησης do_vnbd_request

συσκευή αναλαμβάνει να τις εξυπηρετήσει. Αρχικά επιλέγει σε ποιόν controller θα σταλεί η κάθε αίτηση και στη συνέχεια την τοποθετεί στην ουρά που αντιστοιχεί στον controller. Στο σχήμα 6.6 δίνεται το διάγραμμα ροής της συνάρτησης do_vnbd_request.

Η συνάρτηση για την εξυπηρέτηση των αιτήσεων είναι ίδια με την αντίστοιχη της συσκευής NBD. Η μόνη διαφορά βρίσκεται στον τρόπο επιλογής του controller στον οποίο θα σταλεί η αίτηση.

Η συσκευή εξετάζει την ουρά των συνδεδεμένων ελεγκτών και συγκεκριμένα τον controller στον οποίο δείχνει το πεδίο curr_c. Εάν ο αριθμός των διαδοχικών αιτήσεων που έχει στείλει στον ελεγκτή είναι ίσος ή μεγαλύτερος από την τιμή του βάρους που έχει οριστεί κατά την εισαγωγή του, τότε χρησιμοποιεί τον επόμενο στη λίστα ελεγκτή. Εάν η λίστα είναι άδεια η αίτηση τερματίζεται. Στο σχήμα 6.7 δίνεται το διάγραμμα ροής της συνάρτησης που είναι υπεύθυνη για την επιλογή του controller πριν την αποστολή της κάθε αίτησης.



Σχήμα 6.7 – Διάγραμμα ροής της συνάρτησης που είναι υπεύθυνη για την επιλογή του controller πριν την αποστολή της κάθε αίτησης.

6.2.2 Πρωτόκολλο Επικοινωνίας

Το επίπεδο ανταλλαγής δεδομένων χρησιμοποιεί το πρωτόκολλο NBD για την επικοινωνία με τους ελεγκτές. Η συσκευή στέλνει μηνύματα nbd_request ενώ οι ελεγκτές απαντούν με nbd_reply. Οι δομές που χρησιμοποιούνται για τα μηνύματα είναι οι εξής:

```
struct nbd_request {
      __be32 magic;
      __be32 type;    /* == READ || == WRITE    */
      char handle[8];
      __be64 from;
      __be32 len;
}
```

```
struct nbd_reply {
      __be32 magic;
      __be32 error;           /* 0 = ok, else error   */
      char handle[8];         /* handle you got from request      */
};
```

Το πεδίο handle και στα δύο μηνύματα περιέχει τη διεύθυνση μνήμης της δομής request της ουράς αιτήσεων για να είναι δυνατός ο εντοπισμός της αίτησης μετά την λήψη της απάντησης.

6.2.3 Λήψη Μηνυμάτων Δεδομένων

Στην πλευρά της συσκευής δημιουργείται μία διεργασία χώρου χρήστη για τον κάθε συνδεδεμένο ελεγκτή, η οποία είναι υπεύθυνη να λαμβάνει τα μηνύματα δεδομένων. Αφού δημιουργηθεί το TCP socket και περαστεί στον πυρήνα, μία διεργασία χώρου χρήστη καλεί την ioctl - VNBD_DO_IT η οποία περιέχει έναν βρόχο στον οποίο γίνεται ανάγνωση από το socket και επεξεργασία του μηνύματος όταν αυτό ληφθεί. Η κλήση ioctl επιστρέφει όταν κλείσει το socket, οπότε τερματίζεται και η διεργασία.

6.3 Επίπεδο Μηνυμάτων Ελέγχου

Το επίπεδο μηνυμάτων ελέγχου περιλαμβάνει το πρωτόκολλο επικοινωνίας, τη διεργασία που είναι υπεύθυνη να λαμβάνει τα μηνύματα ελέγχου στην πλευρά της συσκευής, τη διεργασία που λαμβάνει και επεξεργάζεται τα μηνύματα στην πλευρά του ελεγκτή και τέλος την διεργασία για τον έλεγχο των controllers.

6.3.1 Πρωτόκολλο Επικοινωνίας

Η συσκευή στέλνει μηνύματα inst_msg_request ενώ οι ελεγκτές απαντούν με inst_msg _reply. Οι δομές που χρησιμοποιούνται για τα μηνύματα είναι οι εξής:

```
struct inst_msg_request{
        __be32 code;
        __be32 id;
        __be32 event_id;
};

struct inst_msg_reply{
        __be32 code;
        __be32 id;
        __be32 event_id;
        __be32 arg; //data port number OR number of clients(for STAT)
        __be32 uptime;
        __be32 msg_to_time;
};
```

inst_msg_request	inst_msg_reply
REGISTER	REGISTRATION_DENIED
	REGISTRATION_ACCEPTED
PING	PING_REPLY
GET_STAT	STAT

Πίνακας 6.5 – Τιμές του πεδίου code για τα μηνύματα επιπέδου ελέγχου

	Λεκτικό Πεδίου		
Τύπος Μηνύματος	**arg**	**uptime**	**msg_to_time**
REGISTRATION _DENIED	-	-	-
REGISTRATION _ACCEPTED	αρ. θύρας που ο ελεγκτής ακούει για μην. δεδομένων	-	-
PING_REPLY	-	-	-
STAT	αρ. πελατών που εξυπηρετεί ο ελεγκτής	χρόνος λειτουργίας του ελεγκτή	αρ. μηνυμάτων δεδομένων που δέχεται ο ελεγκτής στη μονάδα του χρόνου

Πίνακας 6.6 – Τα πεδία παραμέτρων που χρησιμοποιεί ο κάθε τύπος μηνύματος ελέγχου

Το πεδίο code είναι ο κωδικός που αντιστοιχεί στο μήνυμα που στέλνεται ή λαμβάνεται. Οι κωδικοί των μηνυμάτων δίνονται στον πίνακα 6.5 και η λειτουργία τους έχει περιγραφεί στην παράγραφο 5.2.2. Ανάλογα με το μήνυμα που επιστρέφεται από τον ελεγκτή, χρησιμοποιούνται ορισμένα, κανένα ή όλα τα πεδία παραμέτρων του μηνύματος inst_msg _reply. Στον πίνακα 6.6 φαίνονται τα πεδία που χρησιμοποιεί ο κάθε τύπος μηνύματος και η περιγραφή για το κάθε πεδίο.

Στο πεδίο id των μηνυμάτων τοποθετείται η τιμή του αναγνωριστικού που χρησιμοποιεί η συσκευή για τον ελεγκτή, ενώ στο πεδίο event_id τοποθετείται ο αριθμός γεγονότος που αντιστοιχεί στο μήνυμα που αποστέλλεται στον ελεγκτή.

6.3.1.1 Πίνακας Γεγονότων

Το πεδίο ev της δομής vnbd_dev αντιστοιχεί σε έναν πίνακα γεγονότων. Ως γεγονός αναφέρεται η λήψη ενός μηνύματος του επιπέδου ελέγχου και η δομή που το περιγράφει δίνεται στον πίνακα 6.7.

Το πεδίο ev είναι του τύπου struct event_table ο οποίος ορίζεται ως εξής:

```
struct event_table{
      struct event e[MAX_CONTROLLERS]; /* Πίνακας γεγονότων */
      atomic_t curr_event_id;
      wait_queue_head_t wq;
};
```

Τύπος Πεδίου	*Όνομα Πεδίου*	*Περιγραφή*
`unsigned long int`	`request_event_id`	ο αύξων αριθμός γεγονότος που αντιστοιχεί στο μήνυμα που στέλνεται
`unsigned long int`	`reply_event_id`	ο αύξων αριθμός γεγονότος στο μήνυμα που λαμβάνει η συσκευή
`atomic_t`	`flag`	παίρνει την τιμή WAIT για όσο χρόνο αναμένεται απάντηση και την τιμή DONE όταν φτάσει η απάντηση
`int`	`code`	ο κωδικός του μηνύματος που λαμβάνεται
`int`	`arg`	οι παράμετροι που επιστρέφονται μέσα στο μήνυμα που λαμβάνεται από τη συσκευή.
`unsigned long int`	`uptime`	
`int`	`msg_to_time`	

Πίνακας 6.7 – Τα πεδία της δομής `event`

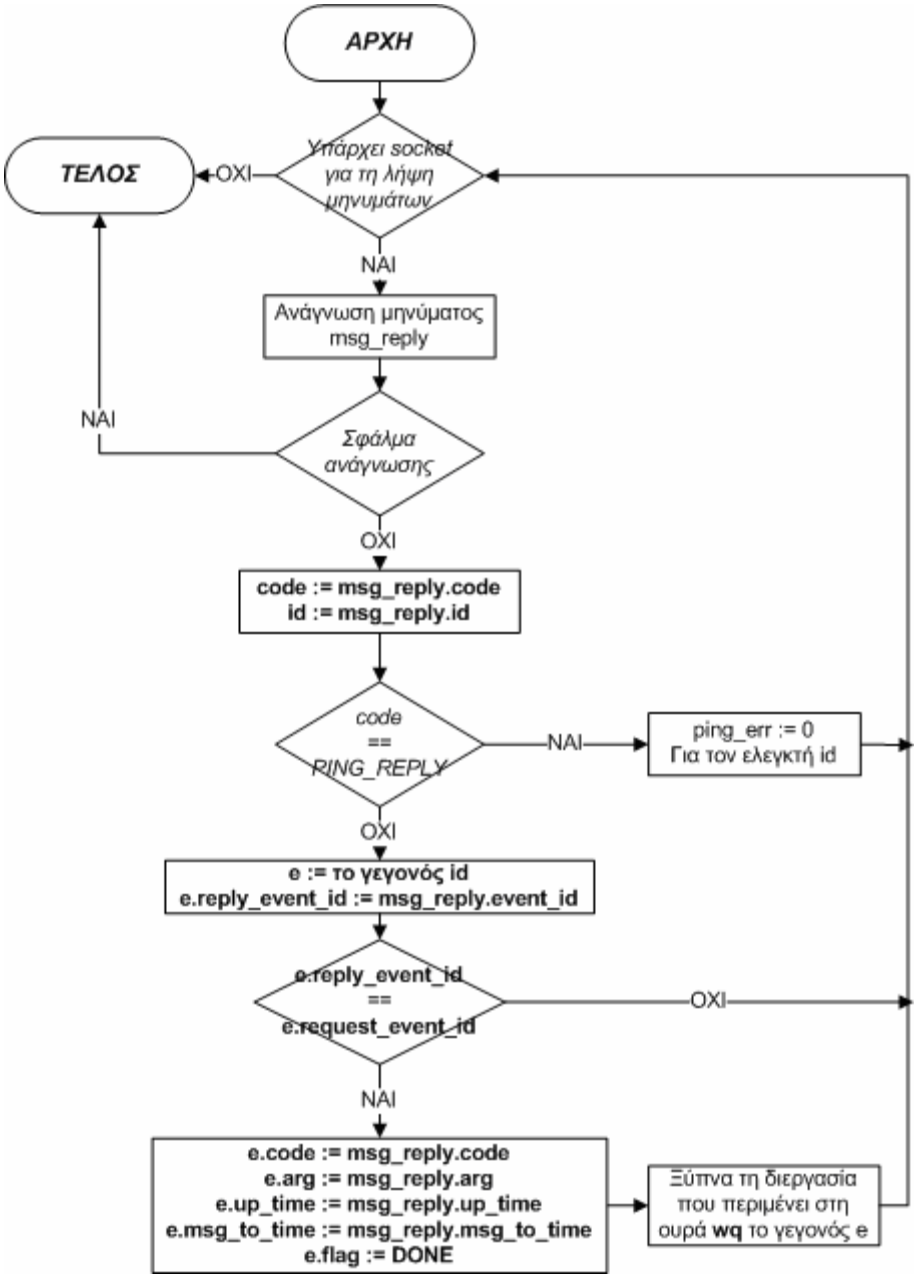
Ο πίνακας γεγονότων χρησιμοποιείται μόνο από τα μηνύματα REGISTER και GET_STAT και η διαδικασία που ακολουθείται κατά την αποστολή ενός μηνύματος ελέγχου προς τον ελεγκτή με αναγνωριστικό id περιλαμβάνει τα εξής βήματα:

1. Αρχικοποίηση του γεγονότος-id, δηλαδή η συσκευή θέτει την τιμή του πεδίου `request_event_id` ίση με την τιμή του `curr_event_id` και την τιμή του πεδίου `flag` ίση με WAIT.
2. Το `curr_event_id` αυξάνεται κατά ένα.
3. Δημιουργία του μηνύματος που θα σταλεί και συμπλήρωση των πεδίων του με τις ανάλογες τιμές.
4. Αν το πεδίο `flag` έχει τιμή WAIT συνέχισε στο βήμα 5
5. Αποστολή του μηνύματος στον ελεγκτή id
6. Αναμονή στην ουρά `wq` για χρόνο T ή μέχρι η τιμή του `flag` να γίνει ίση με DONE, δηλαδή μέχρι να ληφθεί η απάντηση.
7. Αν δεν έχει γίνει αποστολή του μηνύματος πάνω από 3 φορές επέστρεψε στο βήμα 4.

Ο μόνος περιορισμός που υπάρχει βάσει των παραπάνω είναι ότι δεν μπορεί να γίνει για τον ίδιο ελεγκτή αποστολή ενός μηνύματος REGISTER όσο εκκρεμεί ένα μήνυμα GET_STAT ή το αντίστροφο. Κάτι τέτοιο όμως δεν μπορεί να συμβεί καθώς τα μηνύματα REGISTER μπορούν να σταλούν μόνο σε ελεγκτές που η συσκευή έχει καταχωρήσει ως NOT_REGISTERED, ενώ αντίθετα τα μηνύματα GET_STAT σε αυτούς που είναι ήδη συνδεδεμένοι (κατάσταση REGISTERED). Αφού λοιπόν ένας ελεγκτής μπορεί να βρίσκεται σε μία μόνο κατάσταση, δεν υπάρχει περίπτωση να γίνει αποστολή των δύο τύπων μηνυμάτων μέσω του ίδιου γεγονότος.

6.3.2 Διεργασία Λήψης Μηνυμάτων Ελέγχου

Αφού η διεργασία χρήστη που είναι υπεύθυνη για τη διαχείριση της συσκευής αρχικοποιήσει ένα UDP socket και το περάσει στον πυρήνα με κλήση της ioctl-SET_INST_SOCKET,



Σχήμα 6.8 – Διάγραμμα ροής της συνάρτησης που αντιστοιχεί στην κλήση ioctl-READ_INST_SOCK .

δημιουργείται μία διεργασία στο χώρο χρήστη η οποία καλεί με τη σειρά της την ioctl-READ_INST_SOCK. Με την κλήση αυτή εκτελείται εσωτερικά στη συσκευή ένας βρόχος στον οποίο γίνεται ανάγνωση από το socket και επεξεργασία του μηνύματος όταν αυτό ληφθεί. Η κλήση ioctl επιστρέφει όταν κλείσει το socket, οπότε τερματίζεται και η διεργασία. Στο σχήμα 6.8 δίνεται το διάγραμμα ροής της συνάρτησης που αντιστοιχεί στην κλήση ioctl-READ_INST_SOCK.

Όπως φαίνεται και από το διάγραμμα η συνάρτηση σταματά εάν συμβεί κάποιο σφάλμα κατά την ανάγνωση, γεγονός που σημαίνει είτε ότι το socket δεν είναι πλέον ανοιχτό είτε ότι η

διεργασία χρήστη που έχει καλέσει την ioctl- READ_INST_SOCK έχει τερματιστεί με κάποιο μήνυμα SIGKILL.

6.3.3 Επεξεργασία Μηνυμάτων Ελέγχου στον Ελεγκτή

Στην πλευρά του κάθε ελεγκτή υπάρχει μία διεργασία υπεύθυνη να λαμβάνει τα μηνύματα του επιπέδου ελέγχου και να απαντά με την αποστολή του σωστού μηνύματος. Ανάλογα με τον κωδικό του μηνύματος αποστέλλονται και οι παράμετροι που απαιτούνται όπως περιγράφονται στον πίνακα 6.6.

6.3.4 Διεργασία Ελέγχου Λειτουργίας των Controllers

Κατά την αρχικοποίηση της συσκευής δημιουργείται μία διεργασία στον χώρο του πυρήνα, η οποία είναι υπεύθυνη να ελέγχει εάν οι συνδεδεμένοι controllers συνεχίζουν να βρίσκονται σε λειτουργία. Η συσκευή διατρέχει τον πίνακα των ελεγκτών κάθε T δευτερόλεπτα και για κάθε εγγραφή που βρίσκεται στην κατάσταση REGISTERED στέλνει ένα μήνυμα PING στον ανάλογο controller και αυξάνει κατά ένα το πεδίο ping_err της δομής controller_struct. Εάν πριν την αποστολή του μηνύματος το πεδίο αυτό είναι ίσο με 3, γεγονός που σημαίνει ότι δεν έχουν έρθει απαντήσεις για 3 συνεχόμενα μηνύματα PING, καλείται η μέθοδος deregister με την οποία η δομή controller_struct τίθεται σε κατάσταση NOT_REGISTERED και κατά συνέπεια σταματά να χρησιμοποιείται από τη συσκευή.
Όπως φαίνεται και από το διάγραμμα ροής στο σχήμα 6.8, κάθε φορά που λαμβάνεται ένα μήνυμα PING_REPLY το πεδίο ping_err μηδενίζεται για τον συγκεκριμένο ελεγκτή.
Η διεργασία τερματίζεται όταν η κατάσταση της συσκευής πάψει να είναι ACTIVE (πεδίο status της δομής vnbd_dev).

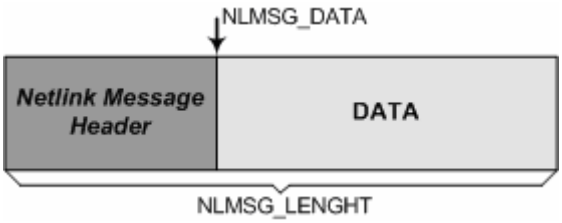
6.4 Διαχείριση της Συσκευής

Η διαχείριση της συσκευής γίνεται μέσω μιας εφαρμογή χώρου χρήστη η οποία ονομάζεται vnbd-client. Η διεργασία αυτή επικοινωνεί με τη συσκευή κυρίως μέσω netlink μηνυμάτων, γεγονός που σημαίνει ότι θα πρέπει να έχει οριστεί ένα πρωτόκολλο για την επικοινωνία. Επιπλέον στον χώρο του πυρήνα θα πρέπει να υπάρχει μία διεργασία για να λαμβάνει και να διαχειρίζεται τα μηνύματα του επιπέδου netlink.

6.4.1 Εφαρμογή vnbd-client

Η εφαρμογή vnbd-client δίνει στον χρήστη τη δυνατότητα να εκτελέσει μία από τις παρακάτω λειτουργίες:

- Έναρξη λειτουργίας της συσκευής [`vnbd-client start`]



Σχήμα 6.9 – Η δομή ενός μηνύματος του επιπέδου netlink

- Τερματισμός λειτουργίας της συσκευής [`vnbd-client stop`]
- Εκτύπωση πίνακα ελεγκτών [`vnbd-client -p`]
- Εισαγωγή ελεγκτή [`vnbd-client –i <ip> <inst_port> <weight>`]
- Διαγραφή ελεγκτή [`vnbd-client –d <id>`]
- Αποστολή αιτήματος σύνδεσης στον ελεγκτή [`vnbd-client –r <id>`]
- Κατάργηση σύνδεσης [`vnbd-client –g <id>`]
- Λήψη στατιστικών ελεγκτή [`vnbd-client –s <id>`]

Οι τρεις πρώτες λειτουργίες χρησιμοποιούν κλήσεις ioctl, ενώ για τις υπόλοιπες χρησιμοποιούνται μηνύματα netlink. Τα βήματα που ακολουθούνται σε κάθε μία από αυτές έχουν παρουσιαστεί αναλυτικά στην παράγραφο 5.4.

Κάθε φορά που η εφαρμογή θέλει να στείλει ένα μήνυμα στη συσκευή δημιουργεί ένα netlink socket με την κλήση:

```
nlsd = socket(AF_NETLINK,SOCK_RAW,NETLINK_VNBD)
```

Η τιμή NETLINK_VNBD καθορίζει το πρωτόκολλο που θα χρησιμοποιηθεί και πρέπει να έχει οριστεί στο αρχείο `<linux/netlink.h>`.

Στη συνέχεια αρχικοποιείται μία δομή τύπου struct nlmsghdr η οποία αντιστοιχεί στην επικεφαλίδα του μηνύματος netlink (σχήμα 6.9) και ορίζεται ως εξής:

```
struct nlmsghdr{
      __u32 nlmsg_len; // το μήκος του μηνύματος
      __u32 nlmsg_type; // ο τύπος του μηνύματος
      __u32 nlmsg_flags; // σημαίες ελέγχου
      __u32 nlmsg_seq;
      __u32 nlmsg_pid; //το pid της διεργασίας που στέλνει το μήνυμα
};
```

Ο τύπος του μηνύματος για την περίπτωση του NETLINK_VNBD πρωτοκόλλου μπορεί να πάρει μία από τις τιμές VNBD_ACTION ή VNBD_STAT. Τα μηνύματα stat χρησιμοποιούνται για την περίπτωση που ζητούνται πληροφορίες για τον ελεγκτή ενώ για όλες τις άλλες περιπτώσεις χρησιμοποιούνται τα μηνύματα action.

Από την άλλη, το μήκος του μηνύματος δεδομένων ορίζεται με τη βοήθεια της μακροεντολής `NLMSG_LENGTH` η οποία παίρνει σαν παράμετρο το μήκος της δομής που χρησιμοποιείται από το πρωτόκολλο σε bytes και επιστρέφει το μήκος του μηνύματος που συμπεριλαμβάνει

Τύπος Πεδίου	*Όνομα Πεδίου*	*Περιγραφή*
int	id	το id του ελεγκτή για τον οποίο ζητούνται τα στατιστικά
unsigned long int	ip	η ip του ελεγκτή
unsigned long int	uptime	ο χρόνος λειτουργίας του ελεγκτή
int	clients	ο αριθμός των πελατών που εξυπηρετεί ο ελεγκτής
int	msg to time	ο αριθμός των μηνυμάτων που ο ελεγκτής εξυπηρετεί στη μονάδα του χρόνου
int	error	ο κωδικός σφάλματος

Πίνακας 6.8– Τα πεδία της δομής `vnbd_nl_get_stat_msg`. Τα πεδία που δίνονται υπογραμμισμένα είναι τιμές που επιστρέφονται από τη συσκευή.

Τύπος Πεδίου	*Όνομα Πεδίου*	*Περιγραφή*
int	action	ο κωδικός της λειτουργίας
unsigned long int	ip	η ip του ελεγκτή
int	iport	η θύρα στην οποία ο ελεγκτής ακούει για μηνύματα ελέγχου
int	id	το id του ελεγκτή
int	weight	το βάρος για τον ελεγκτή
int	dport	η θύρα στην οποία ο ελεγκτής ακούει για μηνύματα δεδομένων
int	error	ο κωδικός σφάλματος

Πίνακας 6.9 – Τα πεδία της δομής `controller_struct`

και την επικεφαλίδα. Στην περίπτωση του NETLINK_VNBD πρωτοκόλλου χρησιμοποιούνται δύο δομές, μία για τα μηνύματα stat και μία για τα μηνύματα action.

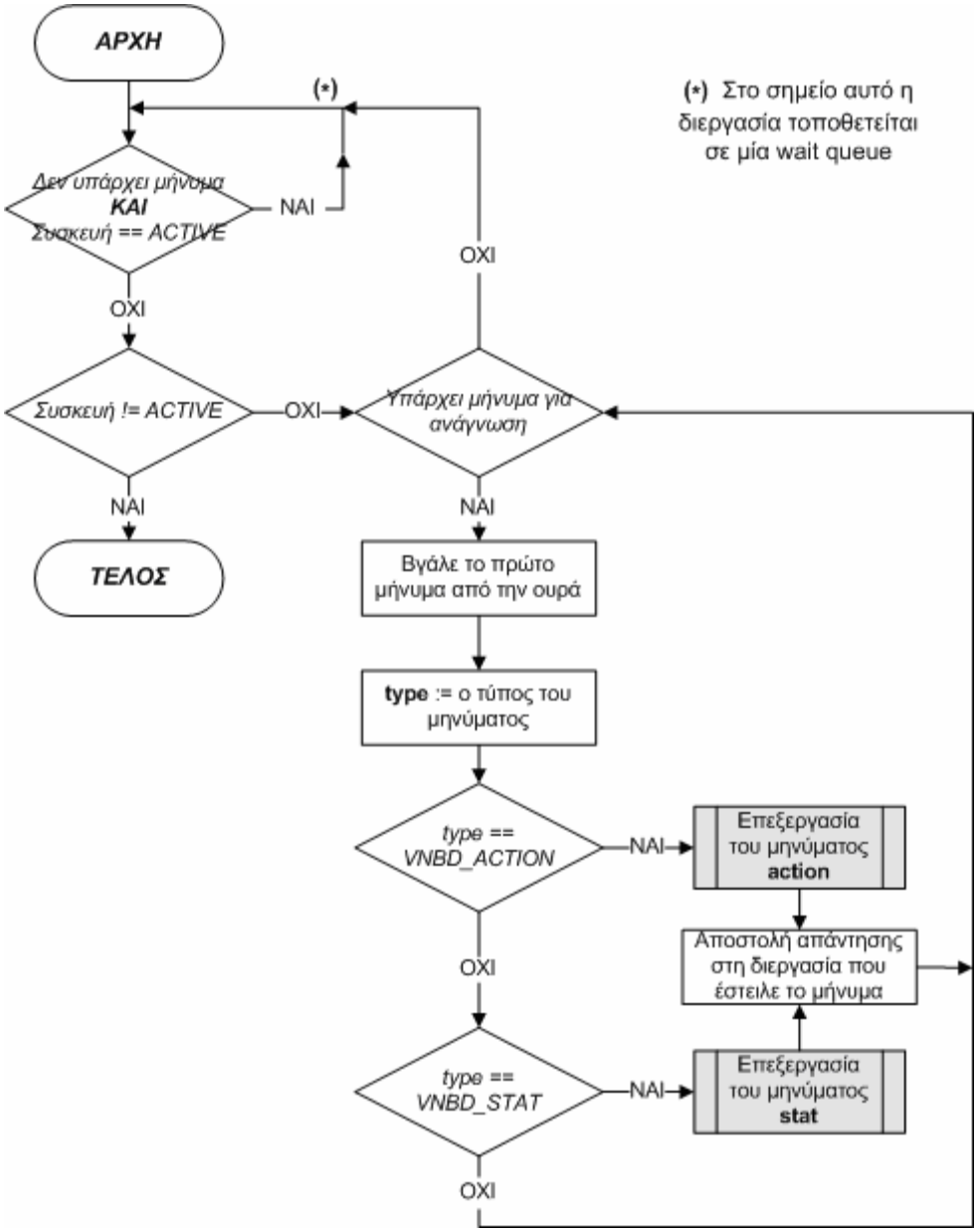
Έπειτα, με την μακροεντολή `NLMSG_DATA` που παίρνει σαν παράμετρο την `nlmsghdr` επιστρέφεται ένας δείκτη στη δομή που χρησιμοποιείται από το πρωτόκολλο και αποτελεί το τμήμα δεδομένων του μηνύματος.

Τέλος, εφόσον έχει αρχικοποιηθεί κατάλληλα τόσο η επικεφαλίδα όσο και το τμήμα δεδομένων το μήνυμα στέλνεται μέσω του netlink socket στον πυρήνα με κλήση της sendto όπως ακριβώς θα γινόταν και με ένα UDP socket, ενώ η απάντηση λαμβάνεται με κλήση της recvfrom.

6.4.2 Netlink Πρωτόκολλο της Συσκευής

Για την περίπτωση της λήψης στατιστικών στοιχείων το τμήμα δεδομένων του μηνύματος netlink αποτελείται από μία δομή vnbd_nl_get_stat_msg τα πεδία της οποίας δίνονται στον πίνακα 6.8. Αντίθετα, στην περίπτωση που ο τύπος του μηνύματος είναι VNBD_ACTION το τμήμα δεδομένων αποτελείται από μία δομή vnbd_nl_action_msg η οποία δίνεται στον πίνακα 6.9.

Καθώς η συσκευή χρησιμοποιεί την ίδια δομή δεδομένων που λαμβάνει μαζί με το μήνυμα netlink για να απαντήσει στην εφαρμογή χρήστη, στις δομές υπάρχουν και πεδία που δεν



Σχήμα 6.10 – Διάγραμμα ροής της συνάρτησης επεξεργασίας των netlink μηνυμάτων.

χρησιμοποιούνται κατά την αποστολή αλλά θα χρησιμοποιηθούν στη συνέχεια από τη συσκευή για να επιστρέψει τις ανάλογες για την κάθε λειτουργία τιμές.

6.4.3 Διαχείριση Netlink Μηνυμάτων

Κατά την αρχικοποίηση της συσκευής θα πρέπει να δημιουργηθεί ένα netlink socket στο οποίο θα λαμβάνονται τα μηνύματα που προορίζονται για τη συσκευή. Η δημιουργία ενός netlink socket στο χώρο του πυρήνα γίνεται με κλήση της συνάρτησης:

```
netlink_kernel_create(NETLINK_VNBD,0,vnbd_recv_sk,THIS_MODULE);
```

η οποία επιστρέφει έναν δείκτη στο socket που δημιουργήθηκε.

Η πρώτη παράμετρος είναι το πρωτόκολλο που χρησιμοποιείται ενώ η παράμετρος `vnbd_recv_sk` είναι η συνάρτηση που θα καλείται κάθε φορά που ένα μήνυμα του πρωτοκόλλου NETLINK_VNBD φτάνει στον πυρήνα. Το μόνο που κάνει αυτή η συνάρτηση είναι να «ξυπνά» τη διεργασία πυρήνα που είναι υπεύθυνη για τη διαχείριση των μηνυμάτων του επιπέδου netlink:

```
static void vnbd_recv_sk(struct sock *sk,int len)
{
      //Κατά την αρχικοποίηση του socket αποθηκεύουμε στον δείκτη
      //sk_user_data τη διεύθυνση της δομής που αναπαριστά τη συσκευή
      struct vnbd_dev *lo = (struct vnbd_dev *)sk->sk_user_data;
      wake_up_interruptible(&lo->thread_wq);
}
```

Αφού έχει δημιουργηθεί το netlink socket δε μένει παρά να δημιουργηθεί η διεργασία για τη διαχείριση των μηνυμάτων. Η διεργασία αυτή δημιουργείται με την κλήση:

```
kthread_run(vnbd_netlink_thread,&vnbd,"vnbd_netlink");
```

στην οποία ορίζεται ότι:

- η διεργασία θα εκτελέσει τη συνάρτηση vnbd_netlink_thread,
- το όρισμα της συνάρτησης θα είναι ένας δείκτης στη δομή της συσκευής και
- το λεκτικό που θα εμφανίζεται για τη διεργασία θα είναι το «vnbd_netlink».

Στο σχήμα 6.10 δίνεται το διάγραμμα ροής της συνάρτησης vnbd_netlink_thread.

7 *Επίδοση της συσκευής*

Για να αξιολογήσουμε την επίδοση της συσκευής χρησιμοποιήσαμε αναγνώσεις και εγγραφές διαφορετικού μεγέθους δεδομένων. Κατά το άνοιγμα του αρχείου της συσκευής χρησιμοποιείτε η παράμετρος O_DIRECT ώστε να απενεργοποιηθεί η χρήση των κρυφών μνημών κατά την πρόσβαση στη συσκευή.

Όπως έχει ήδη αναφερθεί, στην περίπτωση του υπάρχουν περισσότεροι του ενός ελεγκτές θα πρέπει να εφαρμοστούν μηχανισμοί συγχρονισμού των περιεχομένων των caches στους ελεγκτές, κάτι το οποίο στην παρούσα υλοποίηση του vRAID δε γίνεται. Έτσι, η επίδοση της συσκευής ελέγχεται χρησιμοποιώντας έναν nbd-server στη θέση του κάθε controller, και οι μετρήσεις αφορούν τον χρόνο που απαιτείται για την εξυπηρέτηση των αιτήσεων.

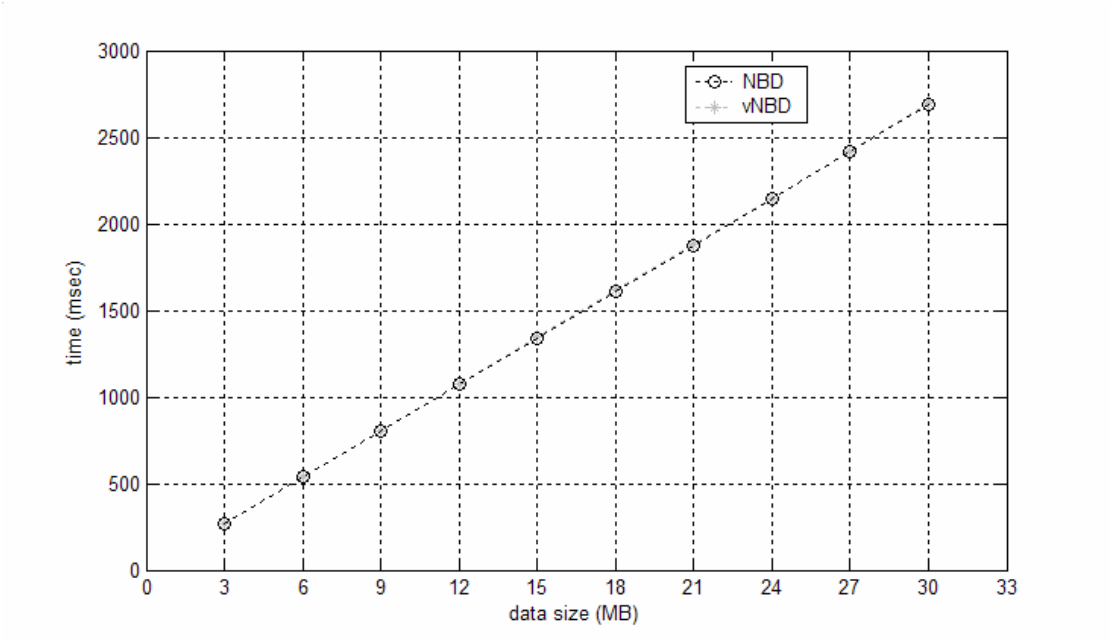
Αρχικά συγκρίνεται η επίδοση της συσκευής nbd σε σχέση με την συσκευή vnbd όταν η τελευταία χρησιμοποιεί έναν controller. Στη συνέχεια γίνονται μετρήσεις για την επίδοση της συσκευής vnbd όταν χρησιμοποιούνται περισσότεροι του ενός ελεγκτές.

7.1 Σύγκριση των Συσκευών NBD και vNBD

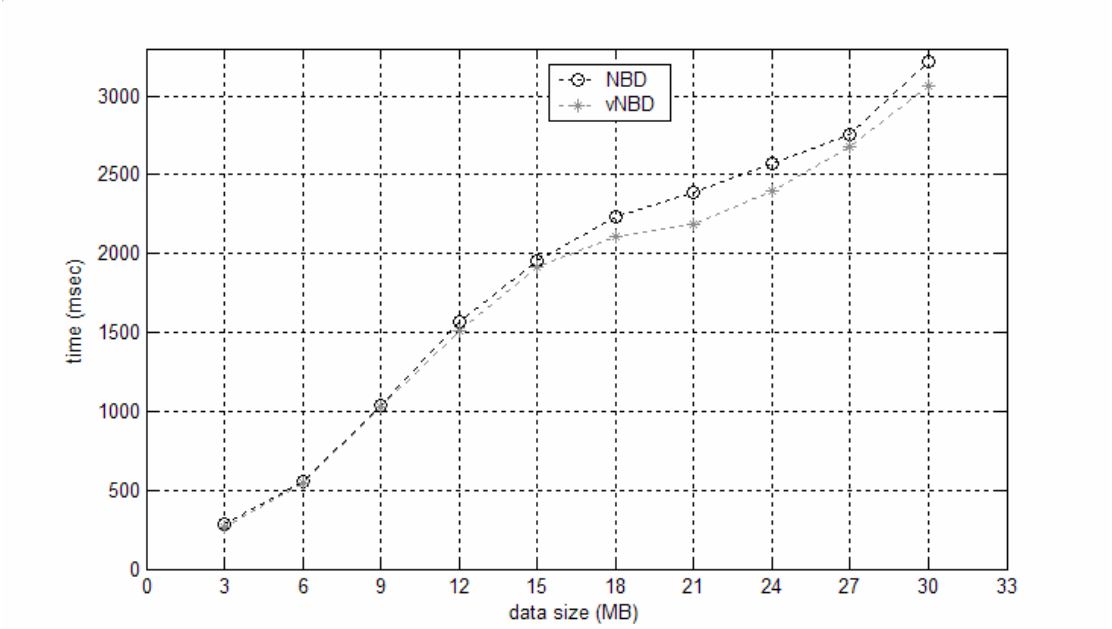
Στο σχήμα 7.1 δίνονται οι γραφικές παραστάσεις του χρόνου που απαιτείται για ανάγνωση ή εγγραφή, συναρτήσει του μεγέθους των δεδομένων που διαβάζονται ή γράφονται .

Η συσκευή vNBD με έναν συνδεδεμένο ελεγκτή για τον οποίο το βάρος έχει οριστεί ίσο με 100, συγκρίνεται με τη συσκευή NBD. Η τιμή του βάρους είναι μεγάλη για να αποφεύγονται οι περιττοί έλεγχοι που αφορούν την επιλογή controller στη συσκευή vNBD.

Όπως φαίνεται και από τις γραφικές παραστάσεις, οι επιπλέον έλεγχοι που έχουν εισαχθεί στη συσκευή vNBD δεν επηρεάζουν την απόδοση της, καθώς λειτουργεί στα ίδια επίπεδα με την NBD.



(α) – Ανάγνωση δεδομένων

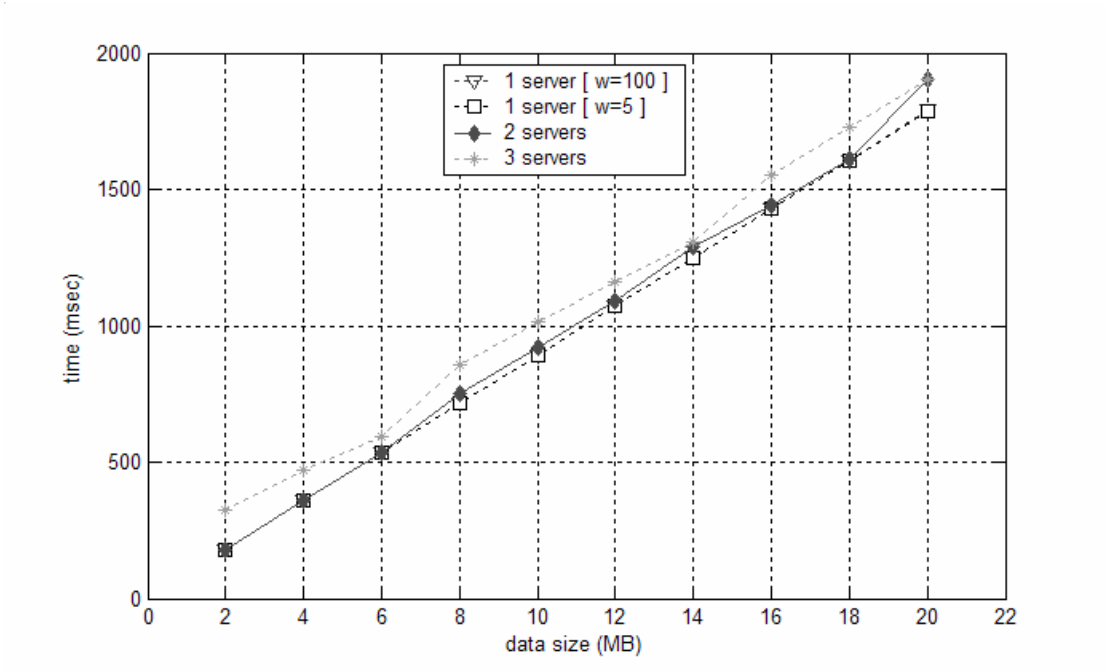


(β) – Εγγραφή δεδομένων

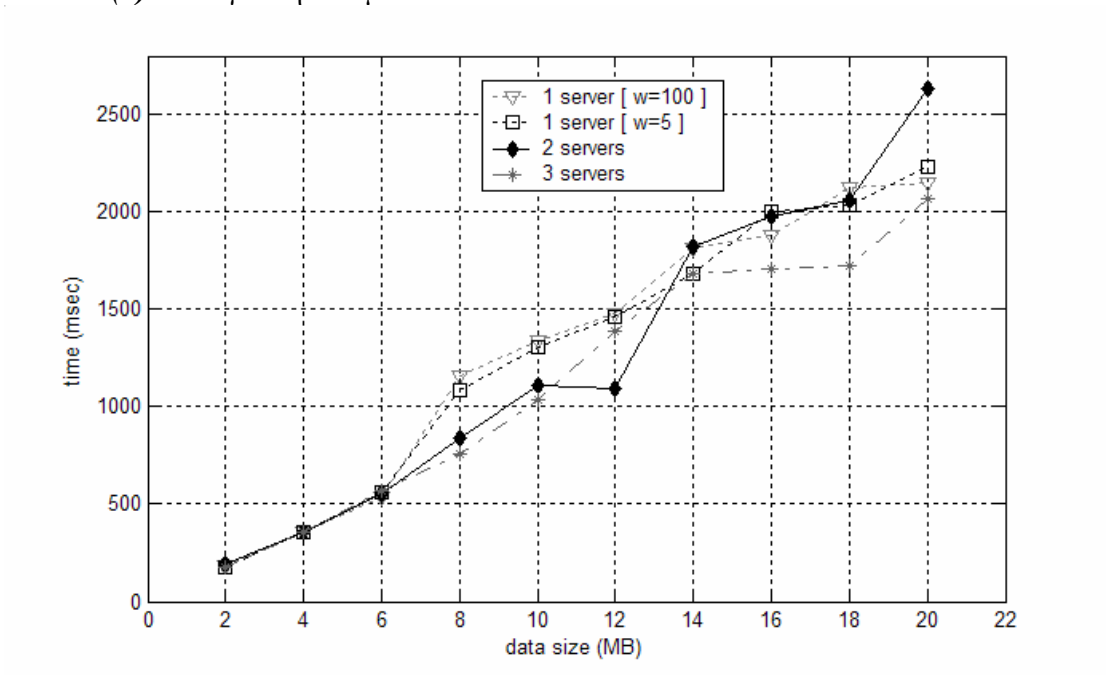
Σχήμα 7.1 – Σύγκριση επίδοσης των συσκευών NBD και vNBD

7.2 Συσκευή vNBD

Στη συνέχεια παίρνουμε μετρήσεις για αναγνώσεις και εγγραφές στην περίπτωση που η συσκευή vNBD χρησιμοποιεί έναν, δύο ή και τρεις εξυπηρετητές. Το βάρος είναι ίδιο για όλους και ίσο με πέντε(5). Στο σχήμα 7.2 δίνονται οι γραφικές παραστάσεις του χρόνου που απαιτείται για ανάγνωση ή εγγραφή, συναρτήσει του μεγέθους των δεδομένων που διαβάζονται ή γράφονται .



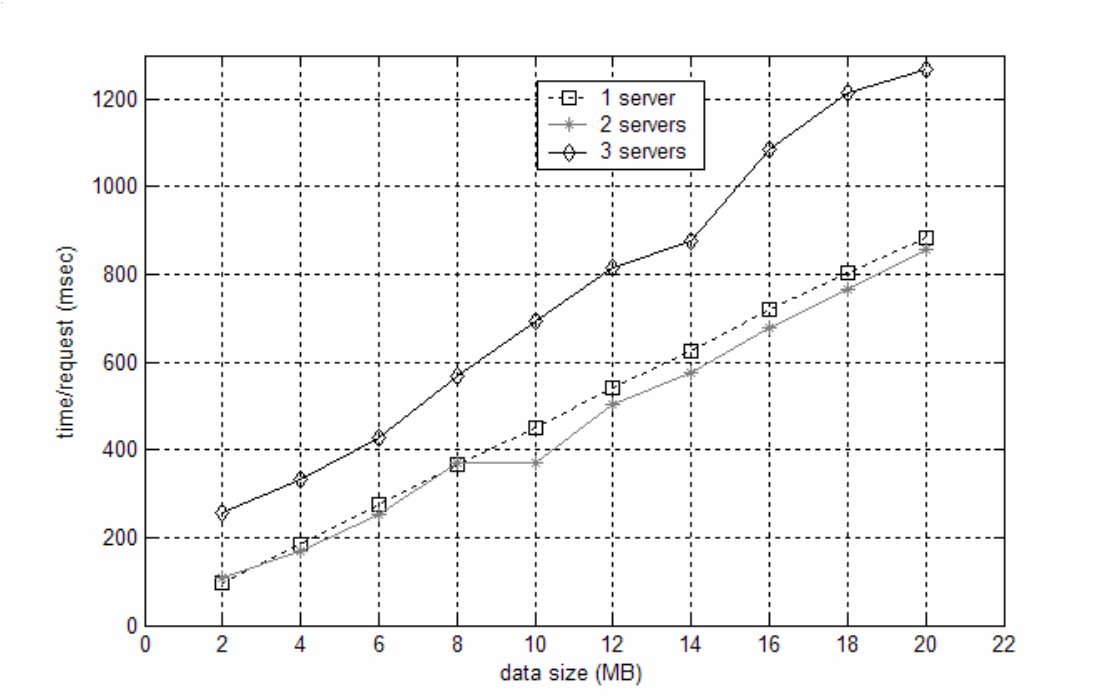
(α) – Ανάγνωση δεδομένων



(β) – Εγγραφή δεδομένων

Σχήμα 7.2 – Επίδοση της συσκευής vNBD κατά την χρήση περισσότερων του ενός ελεγκτών

Όπως φαίνεται από το διάγραμμα του σχήματος 7.2α, ο χρόνος που απαιτείται για την ανάγνωση της ίδιας ποσότητας δεδομένων αυξάνεται με την αύξηση του αριθμού των εξυπηρετητών. Καθώς στην πλευρά του server γίνεται χρήση της cache και τα δεδομένα που διαβάζονται είναι διαδοχικά, οι αιτήσεις εξυπηρετούνται πιο γρήγορα όταν όλες καταλήγουν στον ίδιο server. Επιπλέον, κάθε φορά που επιστρέφει μία απάντηση που αντιστοιχεί σε αίτηση ανάγνωσης έχει μαζί της και τα δεδομένα, γεγονός που σημαίνει ότι απαιτείται χρόνος



Σχήμα 7.3 – Χρόνος που απαιτείται για την εξυπηρέτηση μίας αίτησης στον ίδιο server στις τρεις περιπτώσεις που τα βάρη είναι ίσα με 5.

για την επεξεργασία της κάθε απάντησης. Στην περίπτωση που χρησιμοποιούνται τρεις εξυπηρετητές υπάρχουν στην πλευρά του πελάτη τρεις διεργασίες οι οποίες θα πρέπει να απασχολούν συνεχώς την ΚΜΕ για την ανάγνωση δεδομένων από τα TCP sockets.Αυτό μπορεί να επιβεβαιωθεί και από την μέτρηση του χρόνου που απαιτείται ανά αίτηση ανάγνωσης στον κοινό εξυπηρετητή για τις τρεις περιπτώσεις με βάρος 5.

Κάθε φορά που η απάντηση για κάποια αίτηση φτάνει στη συσκευή, υπολογίζεται ο χρόνος που χρειάστηκε αφαιρώντας από την τρέχουσα χρονική στιγμή τη χρονική στιγμή που η αίτηση τοποθετήθηκε στην ουρά αιτήσεων της συσκευής, μία πληροφορία που υπάρχει μέσα στην δομή request του πυρήνα. Για τον κάθε server οι χρόνοι αυτοί αθροίζονται και διαιρούνται με τον αριθμό των αιτήσεων. Τα αποτελέσματα των μετρήσεων δίνονται στο σχήμα 7.3.

Αν και πρόκειται για τον ίδιο server, στην περίπτωση που οι αιτήσεις δεν είναι διαδοχικές και χρησιμοποιούνται περισσότεροι του ενός servers απαιτείται περισσότερος χρόνος κατά μέσο όρο για τη εξυπηρέτηση της κάθε αίτησης ανάγνωσης.

Αντίθετα στην περίπτωση των εγγραφών φαίνεται από το διάγραμμα του σχήματος 7.2β ότι χρησιμοποιώντας τρεις εξυπηρετητές η απόδοση της συσκευής πλησιάζει την περίπτωση του ενός εξυπηρετητή και σε κάποιες περιπτώσεις βελτιώνεται. Στην περίπτωση αυτή οι απαντήσεις που επιστρέφονται από τους servers δεν περιέχουν δεδομένα, γεγονός που σημαίνει ότι οι διεργασίες που τις επεξεργάζονται δε δημιουργούν στην πλευρά του πελάτη μεγάλο φορτίο, με αποτέλεσμα να βελτιώνονται όλοι οι χρόνοι.

7.3 Συμπεράσματα – Προοπτικές

Από τις μετρήσεις που έγιναν φαίνεται ότι οι χρήση περισσότερων του ενός servers δεν αυξάνει την απόδοση τις συσκευής. Θα πρέπει παρόλ' αυτά να σημειωθεί ότι η συσκευή δεν έχει σχεδιαστεί ώστε να χρησιμοποιεί πολλούς nbd-servers. Στην πραγματικότητα, θα χρησιμοποιεί ένα σύνολο από controllers οι οποίοι θα διαχειρίζονται τα ίδια δεδομένα και θα αναφέρονται στον ίδιο χώρο διευθύνσεων.

Η υλοποίηση μηχανισμών συγχρονισμού των περιεχομένων των cashes στους controllers του vRAID θα επέτρεπε την χρήση της συγκεκριμένης συσκευής για την επικοινωνία του πελάτη με τους ελεγκτές. Στην περίπτωση αυτή τα μηνύματα stat θα μπορούσαν να χρησιμοποιηθούν από τη συσκευή ώστε το βάρος για τον κάθε ελεγκτή να μην είναι στατικό, αλλά να ορίζεται δυναμικά, ανάλογα με τη διαθεσιμότητα του κάθε ελεγκτή. Επιπλέον, θα μπορούσαν να χρησιμοποιηθούν πιο αποδοτικοί αλγόριθμοι για την κατανομή των αιτήσεων με σκοπό την βελτίωση του τρόπου λειτουργίας της συσκευής.

8 *Βιβλιογραφία*

[CRK05] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. Linux Device Drivers, 3rd Edition. O'Reilly, 2005

[BC05] Daniel P. Bovet, Marco Cesati. Understanding the Linux Kernel, 3rd Edition. O'Reilly 2005

[Tho06] Thorsten Zitterell, Diploma Thesis: Development of a Distributed Network Block Device for Wireless Clients, Albert-Ludwigs-University Freiburg, 2006

[AKK04] Αντώνιος-Κορνήλιος Β. Κούρτης, Διπλωματική Εργασία: Σχεδίαση και Υλοποίηση Δικτυακής Συσκευής Block στο Λειτουργικό Σύστημα Linux, Εθνικό Μετσόβιο Πολυτεχνείο, 2004

[Dou06] Ιωάννης Δ. Δούδαλης, Διπλωματική Εργασία: Αποδοτική Μεταφορά Δεδομένων σε Κατανεμημένα Συστήματα Μέσα Αποθήκευσης Υψηλών Επιδόσεων Αξιοποιώντας Δικτυακές Τεχνολογίες ATA-Over-Ethernet και Myrinet, Εθνικό Μετσόβιο Πολυτεχνείο, 2006

[JHP05] John L. Hennessy, David A. Patterson. Αρχιτεκτονική Υπολογιστών, 3η Έκδοση. Εκδόσεις Τζιόλα, 2005

[RL05] Robert Love. Linux Kernel Development, Second Edition. Novell, 2005

[SZ06] Dirk Von Suchodoletz, Thorsten Zitterell. A new mission for the network block device, Linux Magazine, 70, 62-70, 2006

[KH05] Kevin Kaichuan He. Why and How to Use Netlink Socket, Linux Journal, 2005 (www.linuxjournal.com/article/7356)