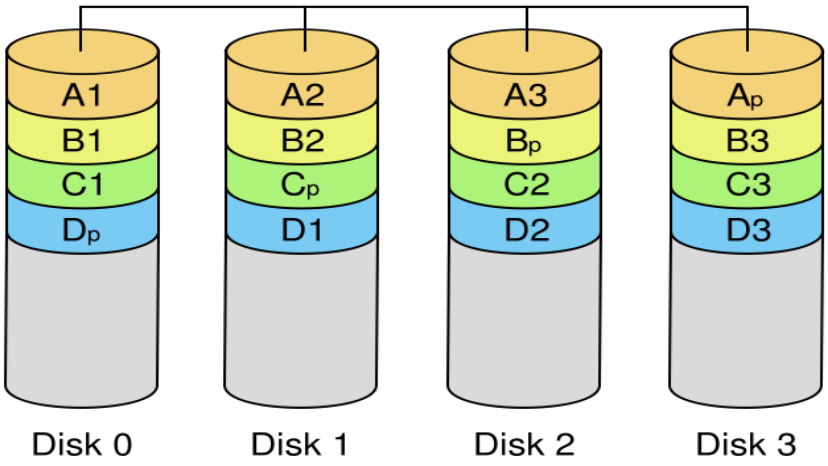
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

«Δημιουργία εφαρμογής παρακολούθησης software-raid για λειτουργικό σύστημα Linux και desktop environment KDE»



Του Φοιτητή:
Ελευθερίου Αλέξανδρου
Αρ. Μητρώου: 03/2310

Επιβλέπων καθηγητής:
Κλεφτούρης Δημήτριος

Θεσσαλονίκη, Νοέμβριος 2011

Πρόλογος

Οι σκληροί δίσκοι στους ηλεκτρονικούς υπολογιστές είναι από τις πιο ευαίσθητες συσκευές. Αυτό, σε συνδυασμό με τον όγκο δεδομένων που μπορούν να αποθηκεύσουν, τους κάνουν ιδιαίτερα επικίνδυνους σε εφαρμογές υψηλής σημασίας. Ειδικά στις μέρες μας, που έχουν φτάσει σε μεγέθη 3 terra byte σε μία μονό συσκευή, η απώλεια έστω και ενός δίσκου θα έχει ώς συνέπεια τεράστια απώλεια δεδομένων. Η κυκλοφορία των δίσκων SSD δεν βοήθησε σε αυτό το θέμα, αφού και αποδείχθηκαν περισσότερο επιρρεπείς σε λάθη αλλά και τα chip από τα οποία αποτελούνται έχουν συγκεκριμένο αριθμό εγγραφών [1]. Στους SSD δίσκους επίσης, η ανάκτηση δεδομένων σε περίπτωση βλάβης είναι σχεδόν πάντα αδύνατη, σε αντίθεση των παραδοσιακών που στις περισσότερες περιπτώσεις μπορούμε να διασώσουμε έστω και μέρος των δεδομένων μας. Για να αποφύγουμε μια τέτοια πιθανότητα μπορούμε, είτε να κρατάμε αρκετά συχνά backup των δεδομένων μας, είτε να χρησιμοποιήσουμε μια συστοιχία δίσκων που θα λειτουργούν σαν μια οντότητα και θα έχουν ως κύριο σκοπό να προστατεύουν τα δεδομένα μας. Η επιλογή του backup έχει αρκετά προβλήματα. Ορισμένα από αυτά είναι:

1) δεν μπορούμε να κάνουμε συνεχώς backup, με αποτέλεσμα να χάσουμε όσα δεδομένα δεν έχουμε κάνει και

2) σε περίπτωση που χρειαζόμαστε να παραμείνει λειτουργικό το μηχάνημα στο οποίο χάλασε ο δίσκο, δεν υπάρχει δυνατότητα να το κάνουμε.

Η επόμενη λύση λοιπόν είναι οι συστοιχίες RAID. Οι συστοιχίες raid (raid arrays) έχουν ως κύριο σκοπό να προφυλάσσουν τα δεδομένα μας από απώλεια σε περίπτωση αστοχίας υλικού (hardware failure). Το πρόβλημα σε αυτή την περίπτωση είναι ότι μία συστοιχία δίσκων μπορεί να έχει ακόμα μεγαλύτερο όγκο δεδομένων από ότι ένας δίσκος πράγμα το οποίο σημαίνει ότι σε περίπτωση βλάβης της συστοιχίας κινδυνεύουμε να χάσουμε ακόμα μεγαλύτερο όγκο αρχείων. Όσο προηγμένη και αν είναι μία συστοιχία raid όμως εξακολουθούν να

υπάρχουν πιθανότητες βλάβης. Για να προβλέψουμε και να αποφύγουμε περιπτώσεις βλάβης χρειάζονται συνεχής παρακολούθηση και συντήρηση. Ο στόχος αυτής της πτυχιακής εργασίας είναι να φτιάξουμε ένα εργαλείο που θα μας βοηθάει να έχουμε καλύτερη εικόνα του software raid σε λειτουργικά linux για να μας βοηθήσει στην παρακολούθηση και συντήρηση αυτών.

Περιεχόμενα

Ευρετήριο Εικόνων

1. Εισαγωγή

Το RAID (“Redundant Arrays of Inexpensive Disks”) ήταν ένα project που ξεκίνησε από το πανεπιστήμιο του Berkley με σκοπό την αξιοποίηση απλών σκληρών δίσκων για τη δημιουργία συστοιχιών οι οποίες θα ξεπερνούσαν σε απόδοση αλλά και σε αξιοπιστία πολύ ακριβότερους enterprise δίσκους [2]. Αυτό το project εξελίχθηκε με τα χρόνια σε πολλές παραλλαγές με την υποστήριξη μεγάλων εταιριών όπως η IBM και κατέληξε να γίνει απαραίτητο κομμάτι σε όλους τους σημερινούς server. Η υλοποίηση του RAID ξεκίνησε με τη χρήση ακριβών controller που ενσωμάτωναν αρκετά γρήγορους, για την εποχή, επεξεργαστές οι οποίοι αναλάμβαναν να εκτελέσουν όλες τις διεργασίες που χρειαζόταν για τη διαχείριση αυτών το συστοιχιών. Με το πέρασμα των χρόνων εξελίχθηκαν οι controller και κυκλοφόρησαν και άλλοι πολύ οικονομικότεροι, αλλά ταυτόχρονα ποιο αργοί και λιγότερο αξιόπιστοί. Σήμερα συναντάμε τέτοιους controller στις περισσότερες motherboard ακόμα και απλών οικιακών Η/Υ. Ταυτόχρονα, με τη κυκλοφορία του linux kernel 2.4 ενσωματώθηκε η δυνατότητα δημιουργίας τέτοιων συστοιχιών σε επίπεδο software, το λεγόμενο software raid [3]. Το software raid αναλαμβάνει να υλοποιήσει συστοιχίες raid οποιουδήποτε είδους, χωρίς κανένα επιπλέον hardware, αλλά με τη χρήση απλά του κεντρικού επεξεργαστή του συστήματος. Με την αύξηση της ταχύτητας των επεξεργαστών, η χρήση του software raid κατέληξε να αποτελεί μια πολύ καλή εναλλακτική και σε πολλές περιπτώσεις μια πιο αξιόπιστη λύση από αντίστοιχες hardware εφαρμογές [4] [5]. Αυτό γίνεται γιατί πάντα υπάρχει η πιθανότητα να χαλάσει ο raid controller και να χάσουμε τα δεδομένα μας, ακόμα και αν οι δίσκοι μας είναι πλήρως λειτουργικοί.Με τη λύση sofware raid εξαλείφουμε ακόμα ένα “point of failure” αφού ακόμα και να χαλάσει όλος ο υπολογιστής μπορούμε να μεταφέρουμε τους δίσκους μας σε άλλο καινούριο hardware και να δουλεύουν όλα κανονικά, χωρίς κανένα πρόβλημα.

Η υλοποίηση RAID κάνει τους δίσκους μας να λειτουργούν σαν μία οντότητα/συσκευή με πολλά θετικά αποτελέσματα, όπως καλύτερη ενιαία

διαχείριση και μεγαλύτερη αξιοπιστία ή ταχύτητα (ή και τα δύο). Υπάρχει και ένα αρνητικό όμως. Σε περίπτωση που χαλάσει το array (η συστοιχία) χάνουμε τα δεδομένα από όλους τους δίσκους μας και είναι από πολύ δύσκολο έως αδύνατο να τα ανακτήσουμε.

Γενικά υπάρχουν αρκετές υλοποιήσεις RAID με πλεονεκτήματα και μειονεκτήματα η κάθε μία. Οι κυριότερες είναι οι 0,1,5,6 και 10. [3]

- Η 0 ή stripe δίνει έμφαση μόνο σε ταχύτητα, μοιράζοντας τα δεδομένα ομοιόμορφα σε 2 ή περισσότερους δίσκους. Αν και η ταχύτητα ανάγνωσης και της εγγραφής είναι πολλαπλάσια των αρχικών δίσκων, δεν υπάρχει καμία προφύλαξη σε περίπτωση βλάβης. Έτσι σε περίπτωση που χαλάσει έστω και ένας από τους δίσκους, καταστρέφεται όλο το array με αποτέλεσμα να έχουμε απώλεια όλων των δεδομένων του.
- Η 1 ή mirroring δίνει έμφαση μόνο στην ασφάλεια και εφαρμόζεται σε ζευγάρια δίσκων οι οποίοι γίνονται κλώνοι ο ένας του άλλου, με απολύτως ίδια δεδομένα. Έτσι αν χαλάσει ο ένας από τους δύο δίσκους, το σύστημα συνεχίζει να δουλεύει χωρίς κανένα πρόβλημα και χωρίς καν να καταλάβει ο χρήστης τι έγινε. Τα μειονεκτήματα σε αυτή την υλοποίηση είναι ότι δεν υπάρχει καθόλου αύξηση της απόδοσης και αξιοποιούμε μόνο τη μισή χωρητικότητα των δίσκων μας.
- Η 5 συνδυάζει τα θετικά και των δυο παραπάνω με ταυτόχρονη αύξηση και ταχύτητας και αξιοπιστίας χρησιμοποιώντας 3 ή περισσότερους δίσκους. Με αυτή την υλοποίηση τα δεδομένα μοιράζονται σε όλους τους δίσκους (για αύξηση ταχύτητας), εκτός από έναν στον οποίο γράφεται το parity για ασφάλεια, σε περίπτωση που χαλάσει ένας δίσκος. Έτσι έχουμε καλύτερη αξιοποίηση χώρου από το raid 1, χάνοντας τη χωρητικότητα ενός μόνο δίσκου από όλους του της συστοιχίας. Σε περίπτωση τώρα που χαλάσει ένας από τους δίσκους συνεχίζει να δουλεύει κανονικά το σύστημα και χωρίς απώλεια δεδομένων. Τα μειονεκτήματα σε αυτή την περίπτωση είναι το μεγάλο overhead για τη δημιουργία του parity, η χαμένη χωρητικότητα του ενός από τους δίσκους και τέλος ότι σε περίπτωση που χαλάσει και 2^{ος} δίσκος χάνουμε τα δεδομένα όλης της συστοιχίας.

- Η 6 είναι παρόμοια με την 5 με διαφορά ότι δημιουργεί διπλό parity και έτσι έχει ανεκτικότητα 2 χαλασμένων δίσκων. Τα αρνητικά εδώ είναι το διπλάσιο overhead του 5 και ο διπλάσιος χαμένος χώρος (όσος δύο δίσκων).

Όπως βλέπουμε υπάρχει πιθανότητα σε όλες τη υλοποιήσεις να χάσουμε τα δεδομένα ολόκληρης της συστοιχίας μας σε περίπτωση που αμελήσουμε ή απλά δεν πέσει στην υποψία μας ότι χάλασε ένας δίσκος. Υπάρχει φυσικά και περίπτωση χαλασμένου controller όταν έχουμε hardware raid αλλά δεν θα ασχοληθούμε με αυτή την περίπτωση.

Η πιθανότητα να μη καταλάβουμε ένα πρόβλημα στο raid, όπως ένα χαλασμένο δίσκο, είναι πολύ μεγάλη από τη στιγμή που δεν υπάρχει καμία ενημέρωση από το σύστημα, ούτε κάποια αλλαγή/πρόβλημα στη λειτουργία του συστήματος. Έτσι είναι πολύ συχνό φαινόμενο να χάνονται δεδομένα από φαινομενικά πολύ ασφαλείς υλοποιήσεις raid.

Για να αποφύγουμε τέτοιες περιπτώσεις και να μη χρειάζεται να ελέγχουμε συνέχεια το status του raid θα δημιουργήσουμε ένα “non intrusive” πρόγραμμα που δεν θα μας ενοχλεί παρά μόνο όταν βρει οποιοδήποτε πρόβλημα στο raid μας. Στην περίπτωση όμως που υπάρξει πρόβλημα στο raid, θα προσπαθήσει να μας τραβήξει την προσοχή με οπτικά μέσα.
Όλα τα δεδομένα για την κατάσταση του software raid υπάρχουν στο αρχείο /proc/mdstat. Έτσι θα χρειαστούμε ένα πρόγραμμα που να ελέγχει ανά τακτά χρονικά διαστήματα αυτό το αρχείο, να κρατάει μόνο τα χρήσιμα στοιχεία από αυτό και να μας τα εμφανίζει σε ένα χρηστικό και ευκολοδιάβαστο γραφικό περιβάλλον (GUI). Επίσης όταν βρει κάποιο πρόβλημα να μας ενημερώνει για να επιλυθούμε του θέματος.

2. Υλοποίηση

Η υλοποίηση θα γίνει με γλώσσα προγραμματισμού υψηλού επιπέδου python. Η python έχει πάρα πολλά πλεονεκτήματα σε σύγκριση με άλλες γλώσσες προγραμματισμού και ελάχιστα μειονεκτήματα

Τα βασικά πλεονεκτήματα της είναι:

- Πολύ εύκολη σε εξοικείωση
- Πάρα πολύ γρήγορη σε ανάπτυξη
- Ελαστική στον τρόπο ανάπτυξης, όπου μπορούμε να χρησιμοποιήσουμε είτε object oriented (αντικειμενοστραφή) programming είτε structured programming
- Εύκολα αναγνώσιμη αφού, γράφεται σε πολύ λιγότερες γραμμές, έχει σύντομες εντολές με απλή σύνταξη και δεν έχει σχεδόν καθόλου «άχρηστα» σημεία στίξης (ερωτηματικά ή παρενθέσεις).

Ένα από τα κυριότερα όμως πλεονεκτήματα είναι ότι δεν χρειάζεται compile, οπότε μπορούμε να μεταφέρουμε το πρόγραμμα μας σε οπουδήποτε λειτουργικό σύστημα, οποιασδήποτε αρχιτεκτονικής και λειτουργεί κανονικά χωρίς καμία αλλαγή.

Τα μειονεκτήματά της είναι:

- Δεν έχει την χαμηλού επιπέδου διαχείριση της μνήμης που έχει πχ η C++
- Είναι λίγο πιο αργή από κάποιες γλώσσες όπως η C++ επειδή είναι scripting γλώσσα και δεν χρειάζεται compile
- Δεν μπορεί να γίνει compile, οπότε πρώτον, χρειάζεται να έχουμε το runtime environment της python και δεύτερον, οποιοσδήποτε μπορεί να δει και να αλλάξει τον κώδικα μας αφού είναι γραμμένος σε plain text.

Τα τελευταία δύο όμως είναι αμφιλεγόμενα. Η ταχύτητα, γιατί στις ταχύτητες των σημερινών υπολογιστών δύσκολα καταλαβαίνεις διαφορά σε ένα πρόγραμμα μικρών έως μεσαίων διαστάσεων. Το ότι δεν μπορεί να γίνει compile από την άλλη

είναι πρόβλημα μόνο σε commercial εφαρμογές, αλλά και αυτό εξαλείφεται σε συγκεκριμένες περιπτώσεις που μπορούμε με συγκεκριμένα εργαλεία να μετατρέψουμε τον κώδικά μας σε executable (εκτελέσιμο) το οποίο να μπορεί να τρέξει χωρίς καν να υπάρχει το runtime environment της python. [6]

Για την ανάπτυξη του GUI θα χρησιμοποιήσουμε βιβλιοθήκες της QT4 που είναι μία από τις πιο ολοκληρωμένες βιβλιοθήκες ανάπτυξης γραφικού περιβάλλοντος για python, C++ και άλλες γλώσσες προγραμματισμού.

3. Δημιουργία software Raid

Προτού ξεκινήσουμε την ανάπτυξη του προγράμματος πρέπει όπως είναι λογικό, να φτιάξουμε ένα raid array. Όπως αναφέρθηκε και προηγουμένως θα χρησιμοποιήσουμε τα δεδομένα του αρχείου /proc/mdstat για την απόσπαση των δεδομένων που μας χρειάζονται. Αν δοκιμάσουμε να το ανοίξουμε (cat /proc/mdstat) χωρίς να έχουμε κάποιο raid array μας δίνει σαν αποτέλεσμα τα παρακάτω:

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
unused devices: <none>
```

1. Κενό mdstat

Το αποτέλεσμα που μας βγάζει είναι μία λίστα από “Personalities” τα οποία είναι και οι επιλογές μας στα είδη Raid που μπορούμε να φτιάξουμε. Το επόμενο βήμα είναι να βρούμε από ποιους δίσκους θέλουμε να αποτελείται το array μας. Για να βρούμε τους δίσκους που έχουμε διαθέσιμους στο σύστημα μας, εκτελούμε το “fdisk -l | grep Disk\ /dev” και μας εμφανίζει μια λίστα με τις συσκευές όπως αυτή που φαίνεται παρακάτω:

```
$fdisk -l | grep Disk\ /dev
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
Disk /dev/sdb: 750.2 GB, 750156374016 bytes
Disk /dev/sdc: 1000.2 GB, 1000204886016 bytes
Disk /dev/sdd: 750.2 GB, 750156374016 bytes
Disk /dev/sde: 1000.2 GB, 1000204886016 bytes
Disk /dev/sdg: 1027 MB, 1027604480 bytes
Disk /dev/sdh: 1056 MB, 1056178176 bytes
```

2. Λίστα συσκευών

Για να μπορέσουμε να κάνουμε τις δοκιμές μας θα χρησιμοποιήσουμε τις συσκευές sdg και sdh οι οποίες είναι flash drives και θα μας βοηθήσουν έτσι ώστε να αναπαράγουμε μία κατάσταση με failed drive.

Αρχικά λοιπόν, θα δημιουργήσουμε ένα raid 1 (mirroring) array. Για να δημιουργήσουμε ένα software raid στο linux υπάρχουν δύο διαφορετικοί τρόποι:

1) Να κάνουμε raid ολόκληρους του δίσκους σαν συσκευές
2) Να φτιάξουμε τα partition που θέλουμε σε κάθε δίσκο, στο μέγεθος που θέλουμε και να κάνουμε raid επάνω σε αυτά.

Και οι δύο τρόποι είναι σωστοί απλά ο δεύτερος μας δίνει απεριόριστες δυνατότητες στα raid που μπορούμε να φτιάξουμε χωρίς κανένα περιορισμό στο μέγεθος, τον τύπο ή αριθμό των δίσκων μας. Θα μπορούσαμε για παράδειγμα να φτιάξουμε ένα raid5 από ένα δίσκο ATA, ένα sata και ένα flash drive χωρίς κανένα περιορισμό στο μέγεθος των συσκευών. Απλά δημιουργώντας 3 partition με το ίδιο μέγεθος. Με τον τρόπο αυτό θα είχαμε επιπλέον και τη δυνατότητα να χρησιμοποιήσουμε τα υπόλοιπα κομμάτια των δίσκων για οποιαδήποτε άλλη χρήση (ή άλλο raid).

Θα δοκιμάσουμε όμως και τους δύο τρόπους. Και για τους δύο τρόπους θα χρησιμοποιήσουμε το εργαλείο mdadm

3.1 Raid σε συσκευές

Πρώτα θα φτιάξουμε ένα raid 1 array. Για να φτιάξουμε ένα array σε συσκευές θα χρησιμοποιήσουμε το mdadm με αντικείμενα τις δύο συσκευές μας. Η εντολή θα είναι :

mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdh /dev/sdg

οι παράμετροι που χρειάστηκαν για την δημιουργία του raid είναι οι εξής:

- **create /dev/md0:** για να δημιουργήσουμε το array /dev/md0 . Το md0 είναι το όνομα της συσκευής που θα είναι το array μας το οποίο πρέπει να ξεκινάει πάντα με md
- **level=1:** είναι ο τύπος του raid που θα δημιουργήσουμε. Στην συγκεκριμένη περίπτωση raid1
- **raid devices=2 /dev/sdh /dev/sdg:** ο αριθμός των συσκευών από τις οποίες θα αποτελείται το array καθώς και ποιές είναι αυτές.

Αν εκτελέσουμε αυτή την εντολή σε drives τα οποία δεν έχουν ακριβώς την ίδια χωρητικότητα, εμφανίζετε η παρακάτω προειδοποίηση που μας ενημερώνει για αυτή την ανισότητα και αν θέλουμε να συνεχίσουμε με αυτές τις συσκευές

```
$mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdh /dev/sdg
mdadm: /dev/sdh appears to be part of a raid array:
    level=raid1 devices=2 ctime=Thu Oct 27 23:46:00 2011
mdadm: /dev/sdg appears to be part of a raid array:
    level=raid1 devices=2 ctime=Thu Oct 27 23:46:00 2011
mdadm: largest drive (/dev/sdh) exceed size (1003456K) by more than 1%
Continue creating array?
```

3. Δημιουργία raid 1 σε άνισες συσκευές

Αφού επιβεβαιώσουμε και προχωρήσουμε βλέπουμε ότι δημιουργείται ένας εικονικός δίσκος με όνομα md0 και μέγεθος τόσο, όσο αυτό της μικρότερης συσκευής μας. Το υπόλοιπο κομμάτι της μεγαλύτερης συσκευής μας χάνεται και δεν μπορούμε να το χρησιμοποιήσουμε πια. Στην περίπτωση μας είναι μόνο 25Kbyte αλλά σε άλλες περιπτώσεις θα μπορούσε να είναι αρκετά Gigabyte, που σε διαφορετική περίπτωση θα μπορούσαμε να τα χρησιμοποιήσουμε σε ένα άλλο partition. Παρακάτω βλέπουμε τη νέα συσκευή μας και τη χωρητικότητά της σε σχέση με τα μέλη της.

Αφού δημιουργήσουμε το array μπορούμε να δούμε τι γίνεται με το νέο raid μας στο αρχείο mdstat:

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdg[1] sdh[0]
      1003456 blocks [2/2] [UU]
      [=============>.......]  resync = 65.5% (657408/1003456) finish=1.2min speed=4502K/sec

unused devices: <none>
```

4. Αρχικό resync ενός Raid 1

Όπως βλέπουμε από τη στιγμή που δημιουργούμε το array ξεκινάει το initialisation που στην συγκεκριμένη περίπτωση μας είναι το resync. Έτσι όταν τελειώσει το resync τα δεδομένα των δύο συσκευών θα είναι ακριβώς ίδια και θα μπορούμε να

χρησιμοποιήσουμε κανονικά το raid μας.

Συνεχίζοντας θα κάνουμε ένα raid 0(stripe) array. Για να χρησιμοποιήσουμε πάλι τις ίδιες συσκευές, έτσι ώστε να φτιάξουμε το νέο array που θέλουμε, θα πρέπει αρχικά να καταργήσουμε τελείως το παλιό. Η διαδικασία αυτή γίνεται σε τρία στάδια:

1) σταματάμε τη λειτουργία του md0 (mdadm --stop /dev/md0)
2) αφαιρούμε τη συσκευή md0 (mdadm --remove /dev/md0)
3) σβήνουμε το superblock στις συσκευές μας, στο οποίο κρατάει κάποιες πληροφορίες για να μπορούμε να φτιάξουμε ξανά το raid array μας (mdadm --zero-superblock /dev/sdg και mdadm --zero-superblock /dev/sdh).

Η εντολή που θα χρησιμοποιήσουμε τώρα για να δημιουργήσουμε το raid μας είναι σχεδόν ίδια, με μόνη διαφορά τον τύπο του raid που γίνεται τώρα:
mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/sdg /dev/sdh

Από την στιγμή που θα εκτελέσουμε την εντολή αυτή δεν εμφανίζεται καμιά άλλη ερώτηση παρά μόνο μια ενημέρωση ότι το array μας έχει ξεκινήσει (“array /dev/md0 started”)

```
$mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sdf1 /dev/sdf2 /dev/sdg1
mdadm: /dev/sdf1 appears to contain an ext2fs file system
    size=613768K  mtime=Sat Oct 29 22:33:12 2011
mdadm: /dev/sdg1 appears to contain an ext2fs file system
    size=613768K  mtime=Sat Oct 29 22:33:12 2011
mdadm: largest drive (/dev/sdf1) exceed size (505280K) by more than 1%
Continue creating array? y
mdadm: array /dev/md0 started.
```

5. Δημιουργία raid 0 σε συσκευές

Σε αυτή την περίπτωση θα δούμε στο output του /proc/mdstat ότι δημιουργείται σχεδόν αμέσως το array και μας δίνει και λίγες παραπάνω πληροφορίες για αυτό, όπως το chunk size

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid0 sdg[1] sdh[0]
      2034816 blocks 64k chunks
```

6. Κατάσταση Raid 0

Από την στιγμή που έχουμε τελειώσει με τη δημιουργία κάποιου raid μπορούμε να προχωρήσουμε στο επόμενο βήμα. Σκοπός μας τώρα είναι να δημιουργήσουμε ένα partition στη συσκευή αυτή για να μπορούμε να την χρησιμοποιήσουμε. Όπως θα κάναμε και με μία φυσική συσκευή, χρησιμοποιούμε fdisk στην εικονική μας συσκευή για να κάνουμε partition και στη συνέχεια χρησιμοποιήσουμε (στην περίπτωση μας) το mkds.ext3 για να διαμορφώσουμε το νέο μας partition στον τύπο που επιθυμούμε.

3.2 Raid σε partition

Για να εξαλείψουμε πιθανό χαμένο χώρο στις συσκευές μας και να αυξήσουμε τις δυνατότητες δημιουργίας raid θα δοκιμάσουμε και την 2η μέθοδο που αναφέραμε νωρίτερα, κατά την οποία αντί να δημιουργήσουμε array συσκευών, που είναι και το παραδοσιακό raid, θα κάνουμε array απο partition. Χρησιμοποιώντας το fdisk θα δημιουργήσουμε ένα 1 partition ίδιας χωρητικότητας σε κάθε drive , στα οποία θα κάνουμε raid στη συνέχεια, ενώ το υπόλοιπο κομμάτι θα μείνει άθικτο και θα μπορούμε το χρησιμοποιήσουμε για όποιον άλλο λόγο θέλουμε.

Ξεκινώντας λοιπόν, δημιουργούμε τα partition με το fdisk. Τρέχοντας το fdisk με παράμετρο το δίσκο μας (/dev/sdf για τον πρώτο και μετά /dev/sdg για τον επόμενο) μας εμφανίζει ένα μενού με τις επιλογές μας. Στην περίπτωσή μας θα εκτελέσουμε πρώτα την επιλογή “o” για καθαρίσει το partition table από τα όποια άχρηστα partiton είχαν δημιουργηθεί προηγουμένως και να έχουμε ένα κενό partition table για να δημιουργήσουμε τα νέα μας partition. Στη συνέχεια με την επιλογή “n” θα δημιουργήσουμε ένα νέο patition και θα επιλέξουμε:

1) να είναι primary partition (“p”)
2) να είναι το πρώτο partition (“Partition Number: 1”)
3) να ξεκινάει από την αρχή του δίσκου (“First Cylinder: 1”)
4) να είναι 512MB (“Last cylinder : +512M”)

```
$fdisk /dev/sdf

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').

Command (m for help): o
Building a new DOS disklabel with disk identifier 0x6bd30053.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1008, default 1): 1
Last cylinder, +cylinders or +size{K,M,G} (1-1008, default 1008): +512M

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

7. Δημιουργία partition με Fdisk

Μόλις ολοκληρώσουμε την παραπάνω διαδικασία έχουμε ένα partition μεγέθους 512MB και κενό χώρο περίπου 500MB. Για να κατοχυρώσουμε τις αλλαγές και να βγούμε από το μενού του fdisk πατάμε w. Κάνουμε το ίδιο με το δεύτερο drive και καταλήγουμε να έχουμε 2 partition τα /dev/sdf1 και /dev/sdg1 τα οποία θα χρησιμοποιήσουμε στη συνέχεια σαν devices του raid array που θα δημιουργήσουμε.

3.2.1 RAID 1

Για τη δημιουργία raid 1 σε partition μπορούμε να χρησιμοποιήσουμε ένα ήδη ενεργό partition με δεδομένα χωρίς να δημιουργήσουμε κανένα πρόβλημα σε αυτά. Έτσι για να δοκιμάσουμε και την ισχύ των παραπάνω φτιάχνουμε:

1) το file system των δύο partition (mkfs.ext3 /dev/sdf1 και mkfs.ext3 /dev/sdg1)
2) κάνουμε mount το /dev/sdf1 μιας και θα είναι το κύριο device του raid μας (επειδή το βάζουμε πρώτο στο option devices)

3) γράφουμε ορισμένα δοκιμαστικά δεδομένα.

Για να δημιουργηθεί το raid 0 που θέλουμε απλά εκτελούμε το mdadm με παραμέτρους ίδιες με τις προηγούμενες με την διαφορά ότι σαν raid devices θα βάλουμε ένα partition στην θέση των συσκευών. Η εντολή μας θα είναι δηλαδή:
“mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdf1 /dev/sdg1”

Αυτή τη φορά όπως θα δούμε τα sizes είναι ακριβώς ίδια και μας ρωτάει απλά αν είμαστε σίγουροι ότι θέλουμε να συνεχίσουμε. Συνεχίζοντας έχουμε μία συσκευή md0 που αν την κάνουμε mount θα δούμε ότι περιέχει τα δεδομένα μας (του partition /dev/sdf1) ανέπαφα.

3.2.2 RAID 0

Για το raid 0 σε partition, όπως και προηγουμένως, δημιουργούμε τα partition με το fdisk χωρίς όμως να τα διαμορφώσουμε και δημιουργούμε το raid μας εκτελώντας το mdadm ως εξής:
“mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdf1 /dev/sdg1”

Σε αυτή την περίπτωση αν χρησιμοποιήσουμε ένα ήδη υπάρχον partition θα καταστραφούν τα δεδομένα μας αφού γίνεται striping των partition και πρέπει ούτως ή άλλως να το διαμορφώσουμε από την αρχή με την εντολή mkfs.ext3 (ή με το αντίστοιχο εργαλείο για το format που θέλουμε).

3.2.3 RAID 5

Με τον ίδιο τρόπο που δημιουργήσαμε τα προηγούμενα raid arrays με τα raid devices να είναι partition μπορούμε να φτιάξουμε και ένα raid 5 δημιουργώντας απλά ένα ακόμα partition σε μία από τις δύο συσκευές μας. Αυτό θα ήταν τελείως λάθος λειτουργικά από την στιγμή που δεν θα είχαμε κανένα κέρδος κάνοντας το, αλλά θα μας βοηθήσει στις δοκιμές μας. Έτσι δημιουργούμε ένα ακόμα partition όπως προηγουμένως στο κενό χώρο του sdf που μας είχε περισσέψει και

δημιουργούμε ένα raid 5 με τα τρία partition που έχουμε:

“mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sdf1 /dev/sdf2 /dev/sdg1”

Όπως βλέπουμε παρακάτω δεν μας αναφέρει κανένα πρόβλημα εκτός από τη διαφορετική χωρητικότητα και αν ανοίξουμε το /proc/mdstat φαίνεται ότι δημιουργείται κανονικά το array μας.

```
$mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sdf1 /dev/sdf2 /dev/sdg1
mdadm: /dev/sdf1 appears to contain an ext2fs file system
    size=613768K  mtime=Sat Oct 29 22:33:12 2011
mdadm: /dev/sdg1 appears to contain an ext2fs file system
    size=613768K  mtime=Sat Oct 29 22:33:12 2011
mdadm: largest drive (/dev/sdf1) exceed size (505280K) by more than 1%
Continue creating array? y
mdadm: array /dev/md0 started.
```

8. Δημιουργία raid 5

3.3 Προβλήματα raid

Μέχρι στιγμής είδαμε τα πιο συνηθισμένα είδη raid και τον τρόπο με τον οποίο δημιουργούμε τέτοια array δίσκων. Το επόμενο στάδιο είναι να εξετάσουμε τα προβλήματα που μπορεί να παρουσιάσει ένα raid array.

Το κύριο πρόβλημα είναι να έχουμε ένα drive failure το οποίο μπορεί να συμβεί αν βγάλει πρόβλημα hardware ο δίσκος αυτός ή απλά αν δεν τον αναγνωρίσει για κάποιο λόγο το λειτουργικό σύστημα. Στην περίπτωση που προσπαθήσουμε να χρησιμοποιήσουμε ένα raid 5 ή raid 0 με drive failure δεν θα καταλάβουμε καμία διαφορά στη λειτουργικότητα και θα γραφτούν/αναγνωστούν κανονικά όλα τα δεδομένα μας χωρίς να ενημερωθούμε ότι υπάρχει κάποιο πρόβλημα. Εδώ βλέπουμε και το πραγματικό πρόβλημα το οποίο θέλουμε να λύσουμε με τη δημιουργία ενός προγράμματος που θα μπορεί να μας ενημερώνει για τέτοια προβλήματα το συντομότερο δυνατό. Στην περίπτωση που είχαμε ένα χαλασμένο δίσκο σε ένα raid 5 ή raid 0 array δεν θα υπήρχε κανένα πρόβλημα στη λειτουργία του array μας ούτε θα χάναμε κανένα μέρος των δεδομένων μας. Αυτή είναι και η

κύρια λειτουργία του raid και για αυτόν ακριβώς το λόγο το χρησιμοποιούμε. Σκεφτείτε όμως να είχαμε ένα raid 5 απο 4 δίσκους του 1 Tbyte. Αυτό θα σήμαινε ότι το array μας θα είχε 3Tbyte δεδομένα από τα οποία δεν θα χάναμε τίποτα σε περίπτωση που χαλούσε ο ένας δίσκος.
Στην περίπτωση όμως που χωρίς να το καταλάβουμε ότι υπάρχει πρόβλημα χαλάσει και δεύτερος δίσκος θα έχουμε χάσει 3Tbyte τα οποία είναι, από πολύ δύσκολο έως σχεδόν αδύνατο να τα επαναφέρουμε με οποιαδήποτε μέθοδο. Και επειδή συνήθως ίδιοι δίσκοι έχουν την τάση να χαλάνε σε σχετικά κοντινές χρονικές περιόδους (λόγο ακριβώς ίδιας χρήσης σε ίδιο περιβάλλον) το παραπάνω σενάριο δεν είναι μπορεί να θεωρηθεί καθόλου υπερβολικό.
Το satus των raid arrays όπως και τα προβλήματα που πιθανών να υπάρχουν σε αυτά βρίσκονται στο αρχείο /proc/mdstat. Ο μόνος τρόπος να ελέγχουμε αν δουλεύει το raid μας σωστά είναι να διαβάζουμε σε τακτά χρονικά διαστήματα το αρχείο αυτό.

Ας δούμε όμως τις διαφορετικές καταστάσεις ενός raid array για να μπορέσουμε έπειτα να τα ενσωματώσουμε στο πρόγραμμά μας.

3.4 Κατάσταση Failed drive

Αρχικά θα δούμε την περίπτωση κατά την οποία ένα drive δεν είναι διαθέσιμο, η οποία είναι και η πιο συνηθισμένη. Θα το δοκιμάσουμε και στα τρία είδη raid που είδαμε και προηγουμένως 0, 1 και 5. Ταυτόχρονα θα δούμε και πώς επιδιορθώνουμε τέτοια προβλήματα και τις διαφορετικές καταστάσεις που εμφανίζει το mdstat.
Ο τρόπος που θα εξομοιώσουμε ένα χαλασμένο δίσκο είναι να αφαιρέσουμε το ένα flash drive και να προσπαθήσουμε να γράψουμε κάτι στο raid array για να ενημερωθεί απευθείας η κατάσταση του.

3.4.1 Failed Drive σε raid 1

Έχοντας ένα λειτουργικό raid 1 βλέπουμε το mdstat ως εξής:

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdh1[1] sdg1[0]
      525632 blocks [2/2] [UU]

unused devices: <none>
```

9. Πλήρως λειτουργικό Raid 1

Για να το χρησιμοποιήσουμε το κάνουμε mount και γράφουμε δοκιμαστικά δεδομένα.

```
$mount /dev/md0 /mnt/raid/
$cp /tmp/testdata1 /mnt/raid/
$sync
```

10. Εγγραφή δοκιμαστικών δεδομένων

Το sync το εκτελούμε για μεταφερθούν ότι δεδομένα υπάρχουν στο buffer και δεν έχουν γραφτεί ακόμα στους δίσκους. Στη συνέχεια αποσυνδέουμε το ένα drive και δοκιμάζουμε να γράψουμε πάλι δεδομένα.

```
$cp /tmp/testdata2 /mnt/raid/
$sync
$
```

11. Εγγραφή δεδομένων σε Raid 1 με failed drive

Όπως βλέπουμε τα δεδομένα μας γράφτηκαν κανονικά και δεν είχαμε καμία απολύτως ενημέρωση για το το drive που λείπει από το array. Αν δούμε όμως το mdstat:

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdh1[1] sdg1[2](F)
      525632 blocks [2/1] [_U]
```

12. Mdstat από Raid 1 με failed drive

Βλέπουμε ότι το array είναι active, δηλαδή λειτουργεί κανονικά αλλά δίπλα στο

partition sdg1 βλέπουμε ότι έχει ένα (F) που σημαίνει Failed drive. Για να επιδιορθώσουμε το array θα πρέπει να βάλουμε ένα λειτουργικό “δίσκο” στη θέση του προβληματικού. Στη περίπτωσή μας θα ξαναβάλουμε το drive. Βλέπουμε όμως ότι έχει αλλάξει όνομα η συσκευή τώρα και έχει πάρει το όνομα sdj.

```
$fdisk -l | grep Disk\ /de

Disk /dev/md0 doesn't contain a valid partition table
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
Disk /dev/sdb: 750.2 GB, 750156374016 bytes
Disk /dev/sdc: 2021 MB, 2021654528 bytes
Disk /dev/sdd: 1000.2 GB, 1000204886016 bytes
Disk /dev/sde: 750.2 GB, 750156374016 bytes
Disk /dev/sdf: 1000.2 GB, 1000204886016 bytes
Disk /dev/sdh: 1027 MB, 1027604480 bytes
Disk /dev/md0: 538 MB, 538247168 bytes
Disk /dev/sdj: 1056 MB, 1056178176 bytes
```

13. Αλλαγμένο όνομα συσκευής

Επίσης το λειτουργικό αναγνωρίζει το sdj1 σαν νέο κομμάτι ενός διαφορετικού, χαλασμένου raid. Έτσι στο mdstat βλέπουμε τα παρακάτω:

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md_d0 : inactive sdj1[0](S)
      525696 blocks

md0 : active raid1 sdh1[1] sdg1[2](F)
      525632 blocks [2/1] [_U]
```

14. Mdstat σε Raid 0 με αλλαγμένο όνομα συσκευής

Για να χρησιμοποιήσουμε ξανά το drive που βγάλαμε πρέπει να το καθαρίσουμε από τα λάθος δεδομένα. Έτσι σταματάμε και σβήνουμε το λάθος raid array όπως είχαμε αναφέρει και προηγουμένως, καθώς επίσης σβήνουμε και το superblock του partition που θέλουμε να χρησιμοποιήσουμε. Για να προσθέσουμε το νέο drive στο array αφαιρούμε πρώτα το “Failed drive” και στη συνέχεια προσθέτουμε το νέο.

```
$mdadm --manage /dev/md0 --remove faulty
mdadm: hot removed 8:97
$mdadm --add /dev/md0 /dev/sdj1
mdadm: added /dev/sdj1
$
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdj1[2] sdh1[1]
      525632 blocks [2/1] [_U]
      [======>.............]  recovery = 31.7% (167168/525632) finish=2.7min speed=2140K/sec

unused devices: <none>
```

15. Προσθήκη δίσκου και recovery ενός Raid 0

Βλέπουμε ότι έχουν μείνει μόνο τα σωστά partition και ότι γίνεται recovery του raid array. Όταν τελειώσει το recovery βλέπουμε ότι όλα δουλεύουν κανονικά. Τα αρχεία μας εξακολουθούν να είναι ανέπαφα και συνεχίζουμε να έχουμε την ασφάλεια που μας προσφέρει το raid 1.

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdj1[0] sdh1[1]
      525632 blocks [2/2] [UU]

unused devices: <none>
$ls /mnt/raid/
lost+found  testdata1  testdata2
```

16. Επαναφορά πλήρους λειτουργικού Raid 1

3.4.2 Failed Drive σε raid 0

Σε ένα raid 0 τα πράγματα είναι λίγο διαφορετικά. Τα δεδομένα μας δεν θα είναι πια προσπελάσιμα σε περίπτωση που δεν δουλεύει ο ένας δίσκος.
Αρχικά ελέγχουμε ότι όλα δουλεύουν κανονικά και γράφουμε κάποια δοκιμαστικά δεδομένα για να μπορούμε να ελέγξουμε αν δουλεύουν μετά.

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid0 sdj1[1] sdh1[0]
      1051328 blocks 64k chunks

unused devices: <none>
$
$cp /tmp/testdata1 /mnt/raid/
$
$sync
```

17. Πλήρως λειτουργικό Raid 0

Στη συνέχεια όμως και στην προηγούμενη περίπτωση αφαιρούμε το ένα drive και προσπαθούμε να γράψουμε ή να διαβάσουμε τα δεδομένα.

```
$cp /tmp/testdata2 /mnt/raid/
$sync
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid0 sdh1[1] sdj1[0]
      1051328 blocks 64k chunks

unused devices: <none>
$cp /mnt/raid/testdata2 /tmp/testdata2_restored
cp: cannot stat `/mnt/raid/testdata2': Input/output error
```

18. Προβληματικό Raid 0 χωρίς να αναγνωρίζεται από το Mdadm

Περιέργως βλέπουμε ότι το λειτουργικό νομίζει ότι έχει μεταφέρει τα δεδομένα. Εκτελώντας όμως “echo 3 > /proc/sys/vm/drop_caches” για να καθαρίσουμε το cache και προσπαθώντας να τα επαναφέρουμε δεν καταφέρνει να τα διαβάσει. Επίσης, στη συνέχεια, δεν μπορούμε να προσθέσουμε ξανά το δίσκο μας, που ούτως ή άλλως αν ήταν χαλασμένος θα είχαμε χάσει τα δεδομένα μας, εξαιτίας του raid 0. Το μόνο που μπορούμε να κάνουμε είναι να σταματήσουμε το raid array και να το δημιουργήσουμε ξανά από την αρχή.

```
$cp /tmp/testdata2 /mnt/raid/testdata3
cp: cannot create regular file `/mnt/raid/testdata3': Read-only file system
$mdadm  /dev/md0 --fail /dev/sdh1
mdadm: cannot find /dev/sdh1: No such file or directory
$mdadm --re-add /dev/md0 /dev/sdk1
mdadm: add new device failed for /dev/sdk1 as 2: Invalid argument
$
$mdadm --stop /dev/md0
mdadm: stopped /dev/md0
$mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/sdj1 /dev/sdk1
mdadm: /dev/sdj1 appears to contain an ext2fs file system
    size=1051328K  mtime=Sun Nov  6 14:16:47 2011
mdadm: /dev/sdj1 appears to be part of a raid array:
    level=raid0 devices=2 ctime=Sun Nov  6 14:15:23 2011
mdadm: /dev/sdk1 appears to be part of a raid array:
    level=raid0 devices=2 ctime=Sun Nov  6 14:15:23 2011
Continue creating array? y
mdadm: array /dev/md0 started.
```

19. Επαναδημιουργία χαλασμένου Raid 0

Όπως θα δούμε στη συνέχεια το testdata1 έχει επαναφερθεί κανονικά, ενώ το testdata2 που μας έδειχνε ότι είχε γραφτεί κανονικά, στην πραγματικότητα είχε γραφτεί το μισό μόνο στο ένα drive, οπότε και επαναφέρθηκε κενό.

```
$ls -l /mnt/raid
total 27196
drwx------ 2 root root    16384 2011-11-06 14:16 lost+found
-rw-r--r-- 1 root root 27831186 2011-11-06 14:20 testdata1
-rw-r--r-- 1 root root        0 2011-11-06 14:20 testdata2
$md5sum /mnt/raid/testdata1 /tmp/testdata1
b21a0cb15414c3bd0e297dc68f777e47  /mnt/raid/testdata1
b21a0cb15414c3bd0e297dc68f777e47  /tmp/testdata1
```

20. Raid 0 δεδομένα που επαναφέρθηκαν

Όπως είδαμε δεν λειτουργεί σωστά η ενημέρωση του αρχείου mdstat για τα προβλήματα του raid 0, αλλά είναι δευτερεύουσας σημασίας καθώς τα προβλήματα του raid 0 τα καταλαβαίνουμε ούτως η άλλως, αφού σταματάει να είναι λειτουργικός. Επίσης raid 0 συναντάμε σχεδόν πάντα σε δεδομένα μικρής σημασίας τα οποία δεν είναι πολύ σημαντικό αν χαθούν.

3.4.3 Failed Drive σε raid 5

Ένα Failed Drive έχει ακριβώς ίδιες επιπτώσεις σε ένα raid 5 όπως και σε ένα raid 0. Πάλι αποσυνδέοντας ένα από τα τρία drive το array δουλεύει κανονικά. Μπορούμε να γράψουμε και να διαβάσουμε από αυτό χωρίς κανένα πρόβλημα, αλλά και χωρίς καμία ενημέρωση. Αν ανοίξουμε όμως το mdstat βλέπουμε τα παρακάτω:

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdg1[3](F) sdf2[1] sdf1[0]
      1010560 blocks level 5, 64k chunk, algorithm 2 [3/2] [UU_]

unused devices: <none>
```

21. Mdstat σε raid 5 με failed drive

Το raid array md0 είναι πάλι active, δηλαδή είναι πλήρως λειτουργικό, αλλά το sdg1 είναι Failed device. Για να προσθέσουμε ξανά το drive στο raid μας θα ακολουθήσουμε ακριβώς τα ίδια βήματα που κάναμε με το raid 1. Θα σταματήσουμε το md_d0 που έχει δημιουργηθεί όταν ξαναβάλαμε το drive και θα προσθέσουμε αυτό το partition στο array μας.

```
$mdadm --stop /dev/md_d0
mdadm: stopped /dev/md_d0
$mdadm --add /dev/md0 /dev/sdi1
mdadm: re-added /dev/sdi1
```

22. Επαναπροσθήκη δίσκου στο Raid 5

Στη συνέχεια μπορούμε να δούμε ότι γίνεται recovery αφού το array έχει πάλι τις συσκευές που χρειάζεται.

```
$cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid5 sdi1[3] sdg1[4](F) sdf2[1] sdf1[0]
      1010560 blocks level 5, 64k chunk, algorithm 2 [3/2] [UU_]
      [==>..................]  recovery = 14.9% (76024/505280) finish=1.5min speed=4472K/sec

unused devices: <none>
```

23. Recovery Raid 5 μετά από επαναφορά δίσκου

Για να αφαιρέσουμε το failed /dev/sdg1 που εχει κρατήσει εκτελούμε:

“mdadm --manage /dev/md0 --remove faulty”

```
$mdadm --manage /dev/md0 --remove faulty
mdadm: hot removed 8:97
```

24. Αφαίρεση faulty συσκευών από array

Έχοντας τα δεδομένα μας, που είναι οι διαφορετικές καταστάσεις που μπορεί να έχει ένα raid array, μπορούμε να προχωρήσουμε στην ανάπτυξη του προγράμματος μας.

Μέχρι στιγμής έχουμε δει τις εξής καταστάσεις:

1) όταν δουλεύει κανονικά το raid
2) όταν έχουμε failed drive που έχει “(F)” δίπλα στο drive
3) όταν γίνεται resync σε raid 1
4) όταν γίνεται recovery σε raid 5
5) όταν το array είναι inactive που έχει “(S)” στο τέλος

Πιθανών να υπάρχουν περισσότερες καταστάσεις οι οποίες όμως είναι αρκετά σπάνιες και αρκετά δύσκολο να τις εξομοιώσουμε.

4. Ανάπτυξη προγράμματος

Η ανάπτυξη του προγράμματος θα γίνει στη γλώσσα προγραμματισμού python. Η python σαν γλώσσα υψηλού επιπέδου είναι από τις ποιό ευέλικτες που υπάρχουν. Δεν χρειάζεται compile, τρέχει σε οποιοδήποτε λειτουργικό σύστημα, αρκεί να έχουμε runtime environment, και ταυτόχρονα δεν υστερεί ιδιαίτερα από πολλές άλλες. Επίσης είναι αρκετά εύκολη στην ανάπτυξη με σύντομες εντολές, ενιαίες μεταβλητές χωρίς κανένα περιορισμό και μεγάλη υποστήριξη από βιβλιοθήκες. Επίσης ο κώδικας μπορεί να γραφεί είτε σαν object oriented είτε ως structured που μας αφήνει να επιλέξουμε τα πλεονεκτήματα όποιου τρόπου χρειαζόμαστε [7]. Το συγκεκριμένο πρόγραμμα, επιλέξαμε να το αναπτύξουμε ως structured για να είναι

ευκολότερο σε ανάγνωση και κατανόηση από τρίτους αφού δεν θα έχει περιττό κώδικα, constructors και μεγάλες εντολές με αντικείμενα.

4.1 Περιγραφή του προγράμματος

Στην αρχή του προγράμματος υπάρχουν:

1) στην πρώτη γραμμή το environment στο οποίο είναι γραμμένο το πρόγραμμα για να μπορεί να τρέχει αυτόνομα, χωρίς να πρέπει να το καλέσουμε μέσω της python
2) στις αμέσως επόμενες γραμμές τα import, που στην περίπτωση μας είναι τα sys, time και gui που είναι το γραφικό μας περιβάλλον.
3) τα import για τις μεθόδους που θα χρειαστούμε, όπως και το γραφικό μας περιβάλλον gui4

```
#!/usr/bin/env python
import sys
import time
from gui4 import *
```

Στη συνέχεια υπάρχουν οι δύο μέθοδοι iconActivated(reason) και check_status(). Η πρώτη όταν εκτελεστεί και έχει ως είσοδο(reason) 2 ή 3 θα εμφανίζει το κεντρικό μας παράθυρο. Αυτή η μέθοδος θα μας χρησιμεύσει για εμφανίζει το παράθυρο όταν κάνουμε μονό ή διπλό click στο tray icon του προγράμματος. Επίσης βλέπουμε ότι στην πρώτη σειρά έχει μια σύντομη περιγραφή της μεθόδου ως docstring για να μπορεί να διαβαστεί και ως help για την κάθε μέθοδο.

Η μέθοδος checkStatus είναι το κύριο μέρος του προγράμματος μας. Η μέθοδος αυτή θα διαβάζει όλα τα στοιχεία των raid array μας και θα επιστρέφει όλα τα στοιχεία που είναι αποθηκευμένα σε ένα πίνακα, για να εμφανιστούν στη συνέχεια στο GUI (graphical user interface) μας. Το πρώτο στάδιο είναι να ανοίξει και διαβάσει τα δεδομένα του αρχείου /proc/mdstat. Επίσης έχουμε και κάποιες βοηθητικές μεταβλητές που θα μας χρησιμεύσουν αργότερα.

```
global fail
global wait
global failtray
if wait >0:
        wait -=1
nofailure=1
failreason=""
failtype=""
failed=""
raidnum = 0
mdstat = open(r'/proc/mdstat')
lines = mdstat.readlines()
mdstat.close()
```

Στο κομμάτι αυτό

1) ανοίγουμε το αρχείο /proc/mdstat για διάβασμα
2) διαβάζουμε και αποθηκεύουμε τα δεδομένα αυτού στη μεταβλητή lines και αμέσως μετά
3) κλείνουμε το αρχείο για να μην έχουμε άλλα προβλήματα ή αργές ενημερώσεις

Το επόμενο βήμα είναι να χωρίσουμε τα δεδομένα του αρχείου μας σε κομμάτια και να τα επεξεργαστούμε, έτσι ώστε να κρατήσουμε μόνο αυτά που χρειαζόμαστε. Αυτά στη συνέχεια θα τα αποθηκεύσουμε σε ένα πίνακα που θα έχει όλα αυτα τα χρήσιμα δεδομένα.

```
raids = []
        a = 0
        faildev=[]
        while a<len(lines):
```

Στο σημείο αυτό πρέπει να δημιουργήσουμε ένα κενό πίνακα που θα περιέχει τα δεδομένα των raid arrays, δηλώνουμε το a που θα μας χρησιμεύσει σαν μετρητής

γραμμών, μία ακόμα μεταβλητή (faildev) που θα χρησιμεύσει σαν μεταβατική και ξεκινάμε να διαβάζουμε μέχρι το τέλος των γραμμών του αρχείου.

```
if lines[a].startswith('md'):
        raidnum+=1
        temp=lines[a].rstrip()
        temp=temp.replace(' : ',' ')
        temp=temp.split(' ')
        raids.append([])
```

Ξεκινάμε κάνοντας έναν έλεγχο στη γραμμή για το αν ξεκινάει από “md”. Αν ξεκινάει από md σημαίνει ότι είναι γραμμή που περιγράφει raid array άρα πρέπει να αποθηκεύσουμε τα στοιχεία που έχει στη συνέχεια. Έπειτα έχουμε τον καθαρισμό από κενά στην αρχή και στο τέλος της σειράς, και ακολουθεί η αποθήκευση σε προσωρινό αρχείο. Γίνεται διαχωρισμός της γραμμής σε ξεχωριστά string με διαχωριστή το κενό χαρακτήρα και αποθηκεύονται όλα τα string σαν αντικείμενα ενός πίνακα στο προσωρινό αρχείο temp. Τέλος στον κεντρικό μας πίνακα “raids” προσθέτουμε ένα κενό στοιχείο, τύπου πίνακα, που θα χρειαστεί στην συνεχεία για τη πρόσθεση των στοιχείων του raid που βρέθηκε.

```
raids[(raidnum-1)].append(temp[0])
raids[(raidnum-1)].append(temp[1])                ]
if temp[2].startswith("hd") or temp[2].startswith("sd"):
        raids[(raidnum-1)].append("")             ]
elif temp[2].startswith("("):
        raids[(raidnum-1)].append(temp[3]+temp[2])
else:
        raids[(raidnum-1)].append(temp[2])
```

Όπως αναφέραμε και παραπάνω, ξεκινάμε προσθέτοντας πρώτα το όνομα του raid array, που είναι το πρώτο στοιχείο της γραμμής (temp[0]), στο τελευταίο στοιχείο του πίνακα raids(raidnum-1). Το επόμενο στοιχείο είναι η κατάσταση/status του array και θα το προσθέσουμε και αυτό όπως είναι στον

πίνακα, μετά το όνομα. Στην συνέχεια το στοιχείο που προσθέτουμε είναι κανονικά ο τύπος του raid, αλλά υπάρχουν ιδιαίτερες περιπτώσεις που, είτε μπορεί να μην υπάρχει καθόλου τύπος raid, είτε μπορεί να έχει ενδιάμεσα ένα στοιχείο “auto readonly”, Το τελευταίο συμβαίνει όταν δεν έχουμε χρησιμοποιήσει καθόλου το raid από την έναρξη του υπολογιστή. Έτσι θα χρησιμοποιήσουμε έναν έλεγχο που να ελέγχει τις πιθανότητες αυτές. Σε περίπτωση που αυτό το πεδίο ξεκινάει με hd ή sd (για ata ή sata δίσκους), ή ξεκινάει από “(“ σημαίνει ότι:

1) Το type δεν υπάρχει καθόλου γιατί πιθανότατα δεν έχει ξεκινήσει το raid, για κάποιο λόγο
2) Το array είναι auto-read-only που συνήθως σημαίνει ότι δεν το έχουμε χρησιμοποιήσει καθόλου από τη στιγμή που ξεκίνησε το σύστημα μας.

Αν δεν υπάρχει ο τύπος του raid γράφουμε ένα κενό χαρακτήρα στο σημείο εκείνο για να διατηρήσουμε τη σωστή αρίθμηση. Αν είναι auto-read-only τότε ενώνουμε το string “auto-read-only” με το επόμενο στοιχείο που είναι ο τύπος και τα προσθέτουμε στον πίνακα μας.

```
c=2
hdds=''
failreason=''
failtype=''
while c<len(temp):
        if temp[c].startswith("hd") or temp[c].startswith("sd"):
.........
.........
        c +=1
```

Στην αρχή έχουμε τις μεταβλητές c, hdd, failreason και failtype. Αυτές θα μας χρησιμεύσουν για:

→ **c:** μετρητής για την while loop που μετράει τα αντικείμενα/λέξεις της κάθε γραμμής που διαβάσαμε νωρίτερα και ξεκινάει απο το 2ο στοιχείο

→ **hdds:** για να αποθηκεύσουμε τους δίσκους που θα βρούμε

→ **failreason:** για να αποθηκεύουμε το λόγο για τον οποίο υπάρχει πρόβλημα

(πχ failed sdh)

→ **failtype:** για να αποθηκεύουμε τον τύπο του προβλήματος (πχ Array Stopped)

Στη συνέχεια ξεκινάει η while loop που ελέγχει ένα-ένα τα αντικείμενα της γραμμής, που έχουμε μετατρέψει σε πίνακα, temp. Σε περίπτωση που βρει ένα αντικείμενο που να ξεκινάει απο hd ή sd, δηλαδή σε περίπτωση που βρει δίσκο προχωράει στους περεταίρω ελέγχους που έχουμε στη συνέχεια, αλλιώς συνεχίζει στο επόμενο αντικείμενο.

```
if temp[c].endswith("(S)"):
        hdds += temp[c][:-3]+"  "
        failreason+= temp[c][:-3]+" Stopped "
        failtype="Array Stopped"
        failtray='<font color="red">'+raids[(raidnum-1)][0]+' Array Stopped</font>'
        nofailure =0
elif temp[c].endswith("(F)"):
        hdds += temp[c][:-3]+"  "
        failreason+= temp[c][:-3]+" Failed "
        failtype="Failed Drive"
        failtray='<font color="red">'+raids[(raidnum-1)][0]+' Failed Drive</font>'
        nofailure =0
else:
        hdds += temp[c][:-3]+"  "
```

Σε περίπτωση που το προηγούμενο if βρει κάποιο δίσκο ελέγχει την περίπτωση να είναι failed ή stopped ο συγκεκριμένος δίσκος, αν δηλαδή έχει μετά το όνομα (S) ή (F). Αν ισχύει κάποιο από τα δύο προσθέτει το δίσκο με το error του στη μεταβλητή failreason και ενημερώνει ο failtype του array. Σε κάθε περίπτωση όμως προσθέτει το δίσκο που βρήκε στη μεταβλητή με τους δίσκους. Το failtray υπάρχει για να αποθηκεύει πληροφορίες για το status των array που θα χρησιμοποιήσουμε αργότερα για να εμφανίζουμε σαν tool tip στο tray icon. Τέλος έχουμε και τη μεταβλητή nofailure που θα μας χρησιμεύσει στη συνέχεια για να γνωρίζουμε αν

υπήρξε κάποιο πρόβλημα σε οποιοδήποτε από τα array.

```
raids[(raidnum-1)].append(hdds)
if failreason != "":              #prosthiki pediou Fail Reason
        raids[(raidnum-1)].append(failreason)
        sysTray.setIcon( QtGui.QIcon('Warning.png'))
        if failtype != fail and wait==0:
                sysTray.showMessage("Warning!",raids[(raidnum-1)][0]+"
"+failtype+"\n"+failreason,2,int(9000))
                wait=20
                fail=failtype
        else:
                raids[(raidnum-1)].append("")
```

Συνεχίζοντας και αφού έχουμε βρει όλους τους δίσκους και τα πιθανά προβλήματα τα αποθηκεύουμε στον πίνακα με τα δεδομένα μας. Πρώτα προσθέτουμε τους δίσκους στον πίνακα και αν υπάρχει πρόβλημα, το προσθέτουμε στο τέλος. Σε διαφορετική περίπτωση προσθέτουμε κενό στο τέλος του, για να μη χαλάσει η αρίθμηση. Σε περίπτωση που υπάρχει πρόβλημα πρέπει να εμφανιστεί με κάποιον τρόπο που να τραβήξει περισσότερο την προσοχή του χρήστη. Για αυτό το λόγο εμφανίζουμε ένα baloon tip στο tray icon που θα περιγράφει το πρόβλημα για 9 δευτερόλεπτα ή μέχρι να το πατήσει ο χρήστης. Για να μην έχουμε προβλήματα με επαναλαμβανόμενες εμφανίσεις ελέγχουμε πριν εμφανίσουμε το baloon δυο μεταβλητές, την wait και την failtype. Έτσι εμφανίζεται μόνο εφόσον έχουμε αλλαγή στον τύπο του error και έχουν περάσει 20 δευτερόλεπτα από την εμφάνιση του προηγούμενου baloon. Στη συνέχεια αν εμφανιστεί ενημερώνουμε και τις δύο αυτές μεταβλητές.

Έχοντας τελειώσει με τις γραμμές που ξεκινάνε από md, δηλαδή από τις γραμμές που μας δίνουν βασικές πληροφορίες για την κατάσταση, τον τύπο και τα αντικείμενα που αποτελούνται τα raid array, προχωράμε στις επόμενες γραμμές για να πάρουμε περισσότερες πληροφορίες, σε περίπτωση που υπάρχουν. Οι επόμενες γραμμές, όταν δουλεύουν υπό φυσιολογικές ρυθμίσεις τα raid arrays,

περιέχουν πληροφορίες μικρής σημασίας, όπως ο αριθμός των block και το chunk size, τις οποίες δεν χρειαζόμαστε σχεδόν ποτέ. Σε κάποιες περιπτώσεις όμως περιέχουν χρήσιμες πληροφορίες όπως όταν γίνεται recovery ή resync. Παρακάτω θα αποσπάσουμε τις πληροφορίες που χρειαζόμαστε και θα τις αποθηκεύσουμε για να τις εμφανίσουμε αργότερα στο GUI μας.

```
        if 'resync' in lines[a]:
        nofailure=0
         sysTray.setIcon( QtGui.QIcon('Clock.png'))
        if '%' in lines[a]:
                temp=lines[a].split(' ')
                raids[(raidnum-1)][1]+='(resync)'
                if temp[10][:-3] == '':
                        raids[(raidnum-1)].append(temp[11][:-3]+"%")
                else:
                        raids[(raidnum-1)].append(temp[10][:-3]+"%")
```

Αρχικά ελέγχουμε αν υπάρχει η λέξη resync σε κάποιο σημείο της γραμμής. Σε περίπτωση που υπάρχει μηδενίζουμε το nofailure για να ξέρουμε ότι υπάρχει κάπου λάθος και αλλάζουμε το tray icon σε ρολόι για να ξέρουμε ότι γίνεται κάποια διεργασία επιδιόρθωσης στο raid, η οποία χρειάζεται ορισμένο χρόνο για να τελειώσει.

Στη συνέχεια ψάχνουμε για το ποσοστό ολοκλήρωσης διεργασίας χρησιμοποιώντας τον χαρακτήρα “%”. Σε περίπτωση που βρούμε ποσοστό στη γραμμή μας αλλάζουμε το πεδίο κατάστασης του raid προσθέτοντας στην υπάρχουσα κατάσταση το “(resync)”. Για να πάρουμε το ποσοστό της διεργασίας που εκτελείτε, χωρίζουμε πρώτα τη γραμμή που διαβάσαμε σε αντικείμενα με διαχωριστή το κενό χαρακτήρα και αποθηκεύουμε στον πίνακα temp. Επειδή, όταν το ποσοστό είναι μονοψήφιο, έχουμε διαφορετικό αριθμό αντικειμένων (λόγο του διαφορετικού αριθμού κενών διαστημάτων) ελέγχουμε αν το 10ο αντικείμενο είναι κενό. Σε περίπτωση που είναι κενό, το ποσοστό είναι το 11ο αντικείμενο, διαφορετικά είναι το 10ο. Έτσι προσθέτουμε και το ποσοστό του resync στο τέλος του πίνακα με τα raid μας.

Το επόμενο βήμα είναι φυσικά να ενημερώσουμε το χρήστη ότι γίνεται resync με

κάτι περισσότερο εμφανές απο το εικονίδιο του tray.

```
failtype="Resync"
failtray='<font color="yellow">'+raids[(raidnum-1)][0]+' Resync</font>'
if failtype != fail and wait==0:
        sysTray.showMessage("Attention!",raids[(raidnum-1)][0]+" Resync in
proccess",2,int(9000))
        wait=20
        fail=failtype
```

Αποθηκεύουμε στη μεταβλητή failtype τον τύπο του failure που έχουμε για να το συγκρίνουμε στη συνέχεια με το προηγούμενο failure (μεταβλητή fail), αν υπήρχε. Σε περίπτωση που έχουμε διαφορετικό failure και έχει τελειώσει το wait time εμφανίζεται balloon tip με “Attention” αυτή τη φορά που ενημερώνει το χρήστη για τη διεργασία.

Υπάρχει όμως ακόμα μία κατάσταση resync που μπορεί να έχουμε, το delayed resync. Τέτοια περίπτωση έχουμε σε περίπτωση που εμποδίζεται για κάποιο άλλο λόγο το resync ή γίνεται κάποια άλλη διεργασία και περιμένει το σύστημα να τελειώσει, για να ξεκινήσει το resync. Αυτό θα μπορούσε να γίνεται σε περίπτωση που είχαμε δύο arrays που έπρεπε να γίνουν resync και έτσι όσο εκτελούταν το πρώτο, το δεύτερο θα έμενε σε κατάσταση delayed resync. Για να συμπεριλάβουμε και αυτή την περίπτωση έχουμε μια εναλλακτική else if (elif στην python) στο if που έλεγχε αν υπάρχει ο χαρακτήρας “%”.

```
elif 'DELAYED' in lines[a]:
        raids[(raidnum-1)][1]+='(Delayed resync)'
        failtype="Delayed Resync"
        failtray='<font color="red">'+raids[(raidnum-1)][0]+' Delayed Resync</font>'
        if failtype != fail and wait==0:
                sysTray.showMessage("Warning!",raids[(raidnum-1)][0]+"
"+failtype,2,int(9000))
                fail=failtype
                wait=20
```

Σε περίπτωση που υπάρχει “DELAYED” κάπου στη γραμμή και αφού έχουμε ήδη ελέγξει αν υπάρχει και το resync δεν κάνουμε άλλους ελέγχους. Ενημερώνουμε την κατάσταση του raid array προσθέτοντας το “(Delayed Resync)” και αφού κάνουμε πάλι τους ελέγχους εμφανίζουμε το balloon tip.

Ο τελευταίος έλεγχος που μας έμεινε να ελέγξουμε είναι το recovery.

```
if 'recovery' in lines[a]:
        nofailure=0
         sysTray.setIcon( QtGui.QIcon('Clock.png'))
        if '%' in lines[a]:
                temp=lines[a].split(' ')
                raids[(raidnum-1)][1]+='(recovery)'
                if temp[10][:-3] == '':
                        raids[(raidnum-1)].append(temp[11][:-3]+"%")
                else:
                        raids[(raidnum-1)].append(temp[10][:-3]+"%")

        failtype="Recovering"
        if failtype != fail and wait==0:
                sysTray.showMessage("Attention!",raids[(raidnum-1)][0]+"
"+failtype,2,int(9000))
                wait=20
                fail=failtype
```

Εφαρμόζοντας τον ίδιο τρόπο με τον οποίο πήραμε το ποσοστό από το resync, ελέγχουμε αν το 10ο αντικείμενο είναι κενό και παίρνουμε ανάλογα με το αποτέλεσμα, το αντικείμενο που έχει το ποσοστό της διεργασίας. Προσθέτουμε αυτό το ποσοστό στο τέλος του πίνακα των raid array και αφού κάνουμε πάλι του αντίστοιχους ελέγχους για αλλαγή κατάστασης εμφανίζουμε το balloon tip.

Έχοντας τελειώσει με όλους τους ελέγχους και έχοντας αποθηκεύσει όλα τα δεδομένα που μας χρειάζονται σε ένα πίνακα, μας μένει να τα εμφανίσουμε οργανωμένα σε ένα GUI. Πριν το κάνουμε αυτό όμως, πρέπει να δούμε και την περίπτωση να είναι όλα εντάξει και να μην έχουμε κανένα failure. Αυτό θα το κάνουμε ελέγχοντας τη μεταβλητή nofailure που θέταμε σε κάθε έλεγχο που είχαμε failure. Σε περίπτωση που δεν πέρασε κανένα έλεγχο η μεταβλητή nofailure θα είναι 1. Έτσι ελέγχοντας αυτή τη μεταβλητή αλλάζουμε το tray icon σε “check” και σβήνουμε τα δεδομένα που μπορεί να είχε η μεταβλητή fail.

```
if nofailure == 1:
        sysTray.setIcon( QtGui.QIcon('Checkmark.png'))
        fail=""
        failtray='<font color="green">Healthy</font>'
```

Για να εμφανίσουμε τώρα τα δεδομένα που έχουμε πάρει οργανωμένα και να μην έχουμε πρόβλημα με το μέγεθος του πίνακα (αριθμό των raid arrays) θα χρησιμοποιήσουμε ένα widget της βιβλιοθήκης Qt που ονομάζεται QTreeWidget. Το QTreeWidget είναι μια δομή που περιέχει δεδομένα τα οποία τα εμφανίζει σε στυλ δέντρου. Στην πραγματικότητα είναι μία δομή τύπου πίνακα που εμφανίζει αντικείμενα τύπου QTreeWidgetItem. Για να προσθέσουμε τα δεδομένα μας στον “πίνακα” αυτό, θα χρειαστεί να μετατρέψουμε πρώτα την κάθε γραμμή του πίνακα σε αντικείμενο QTreeWidgetItem και στη συνέχεια να την προσθέσουμε στο QtreeWidget.

```
ui.treel.clear()
size1=len(raids)
while size1 > 0:
        size1 -=1
        item=QtGui.QTreeWidgetItem(raids[size1])
        ui.treel.addTopLevelItem(item)
```

Αρχικά καθαρίζουμε τα δεδομένα που μπορεί να έχει το widget. Στη συνέχεια φτιάχνουμε μια βοηθητική μεταβλητή για να την έχουμε ως μετρητή και ξεκινάμε

την while loop. Την επανάληψη την ξεκινάμε από το τέλος του πίνακα μας προς την αρχή για να εμφανιστούν τα δεδομένα μας με τη σειρά που βρίσκονται στο αρχείο /proc/mdstat. Στο επόμενο βήμα δημιουργούμε αντικείμενα τύπου QtreeWidgetItem από τα δεδομένα του πίνακα raids και τέλος προσθέτουμε τα αντικείμενα αυτά στο QtreeWidget “treel” που βρίσκεται στο κεντρικό μας GUI.

```
sysTray.setToolTip("""Status: """+ failtray +"""<BR><B>Raid Status Indicator</B><BR>Created by Eleftheriou Alexandros<BR>for ATEI Thessaloniki""")
```

Η τελευταία αλλάζει το tooltip του tray icon ώστε να μπορούμε να δούμε την κατάσταση των raid μας χωρίς να ανοίξουμε το main window, απλά πηγαίνοντας τον κέρσορα πάνω στο εικονίδιο του προγράμματος.

Έχοντας τελειώσει την περιγραφή της κύριας μεθόδου μας, θα ξεκινήσουμε με τη περιγραφή του κεντρικού μέρος του προγράμματος (την main όπως θα λέγαμε σε άλλες γλώσσες προγραμματισμού).

4.2 Περιγραφή κεντρικού μέρους του προγράμματος

Ξεκινάμε με αρχικοποιήσεις μεταβλητών που θα χρησιμοποιήσουμε αργότερα. Αυτές δεν είναι απαραίτητο να είναι στην αρχή του προγράμματος απλά βοηθάει κάποιες φορές έτσι ώστε να είναι ποιο ευανάγνωστος και οργανωμένος ο κώδικας.

```
c=0
wait=0
size=1
fail=""
failtray=""
```

Στη συνέχεια δημιουργούμε ένα application τύπου QtGui.QApplication(sys.argv) και ένα MainWindow τύπου QtGui.QMainWindow. Αυτά τα δύο είναι ή κύρια

εφαρμογή μας που είναι βασισμένη σε βιβλιοθήκες Qt και το κύριο παράθυρο που θα χρησιμοποιήσουμε. Το main window θα το δημιουργήσουμε στο Qt Designer στη συνέχεια.

```
app = QtGui.QApplication(sys.argv)
MainWindow = QtGui.QMainWindow()
```

Μετά ξεκινάμε την ανάπτυξη του tray icon

```
sysTray = QtGui.QSystemTrayIcon()
sysTray.setIcon( QtGui.QIcon('Checkmark.png'))
sysTray.connect(sysTray,
QtCore.SIGNAL("activated(QSystemTrayIcon::ActivationReason)"),
iconActivated)
```

Δημιουργούμε ένα αντικείμενο sysTray τύπου QtGui.QSystemTrayIcon που θα είναι το tray icon μας στο οποίο θα δουλέψουμε. Αλλάζουμε το εικονίδιο του σε “checkmark” που θα παίρνει όταν ξεκινάει το πρόγραμμα και θα μένει έτσι μέχρι να αλλάξει από κάποιο failure. Συνδέουμε επίσης στο εικονίδιο ένα signal που όταν θα κάνουμε click σε αυτό θα εκτελεί την μέθοδο iconActivate με παράμετρο το τρόπο που καλέστηκε (πχ διπλο click).

Για να γίνει περισσότερο λειτουργικό από απλή ένδειξη θα βάλουμε ένα context menu (menu που εμφανίζεται με δεξί click).

```
menu = QtGui.QMenu()
exitAction = menu.addAction("Exit")
showwindow = menu.addAction("Show")
exitAction.triggered.connect(sys.exit)
showwindow.triggered.connect(MainWindow.show)
sysTray.setContextMenu(menu)
sysTray.setVisible(True)
```

Δημιουργούμε αρχικά ένα αντικείμενο menu τύπου QtGui.QMenu το οποίο θα το

συνδέσουμε στο τέλος με tray icon sysTray. Πριν το συνδέσουμε το menu προσθέτουμε σε αυτό δύο “actions”, τα “Exit” και “Show”. Σε κάθε action πρέπει να συνδέσουμε μία διεργασία που θα εκτελεί. Στο “Exit” συνδέουμε το sys.exit που κλείνει το πρόγραμμα και στο “Show” συνδέουμε το MainWindow που θα εμφανίζει το κεντρικό μας παράθυρο. Αφού τελειώσουμε με τις προσθήκες στο menu, το θέτουμε ως context menu στο sysTray. Τέλος έμεινε να κάνουμε το εικονίδιο visible.

Στη συνέχεια θα δημιουργήσουμε ένα αντικείμενο ui τύπου Ui_MainWindow. Η κλάση Ui_MainWindow είναι αυτή που θα μας δώσει το Qt Designer όταν σχεδιάσουμε το γραφικό περιβάλλον μας.

```
ui = Ui_MainWindow()
ui.setupUi(MainWindow)
ui.actionExit.triggered.connect(sys.exit)
ui.action_minimise.triggered.connect(MainWindow.hide)
MainWindow.show()
```

Χρησιμοποιώντας το αντικείμενο ui με είσοδο το MainWindow που είχαμε φτιάξει νωρίτερα δημιουργούμε το τελικό μας παράθυρο. Συνδέουμε τις δύο ενέργειες που θα έχουμε στο menu του main window για ελαχιστοποίηση και έξοδο. Και τέλος εμφανίζουμε το παράθυρο μας με το MainWindow.show().

Αυτό που μας έχει μείνει, είναι να βάλουμε την κεντρική μας μέθοδο να εκτελείτε ανά συχνές περιόδους . Αυτό θα το κάνουμε παρακάμπτοντας την main loop και φτιάχνοντας μία δικιά μας.

```
e=101
while c<200:
        if e>100:
                check_status()
                e=0
        QtGui.QApplication.processEvents()
```

```
        time.sleep(0.01)
        e+=1
```

Έτσι φτιάχνουμε μία endless loop που θα εκτελεί το QtGui.QApplication.processEvents() το οποίο θα ενημερώνει το γραφικό μας περιβάλλον κάθε φορά που θα τρέχει. Για να έχουμε μια σχετικά ομαλή ενημέρωση του gui και ταυτόχρονα να έχουμε πολύ μικρές απαιτήσεις επεξεργαστικής ισχύς, θα βάλουμε ένα sleep σε κάθε loop, για να την καθυστερεί και επίσης θα βάλουμε μια εσωτερική loop που να τρέχει μόνο κάθε 100 φορές που θα έχει γίνει update το γραφικό περιβάλλον.
Σε περίπτωση που αφήναμε τη loop να τρέχει χωρίς χρονοκαθυστέρηση, θα έτρεχε realtime, όσο επέτρεπε ο επεξεργαστής μας και θα είχαμε φοβερή κατανάλωση επεξεργαστικής ισχύος

Στο κεντρικό κομμάτι του προγράμματος μας είχαμε συνδέσει ένα signal στο trayIcon να εκτελεί τη μέθοδο iconActivated σε περίπτωση που γίνεται click. Αυτή η μέθοδος θα βρίσκεται πριν το κεντρικό μας κομμάτι και θα εμφανίζει το κεντρικό μας παράθυρο ανάλογα με τι παραμέτρους που δέχεται.

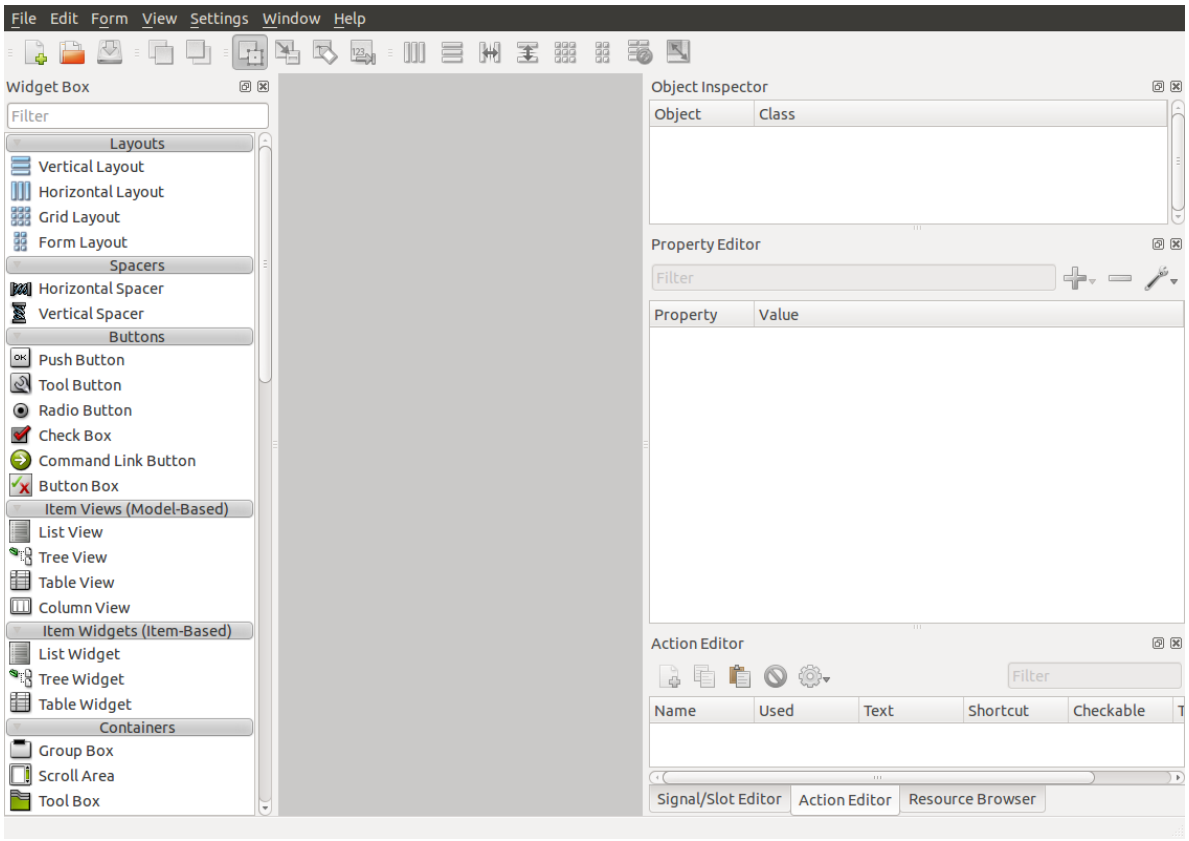
```
def iconActivated(reason):
        """Shows the MainWindow if the TrayIcon is either single or double Clicked"""
        if reason == 2 or reason == 3:
                MainWindow.show()
```

Σε περίπτωση που έχουμε σαν είσοδο 2 ή 3 που μεταφράζεται σε μόνο ή διπλό click θα εμφανίζεται το MainWindow.

4.3 Δημιουργία γραφικού περιβάλλοντος με Qt4 Designer

Για τη σχεδίαση του γραφικού περιβάλλοντος μας θα χρησιμοποιήσουμε ένα εργαλείο της QT 4, το Designer. Το QT4 Designer είναι ένα πολύ χρήσιμο εργαλείο

που κάνει τη δημιουργία γραφικού περιβάλλοντος πολύ εύκολη και γρήγορη. Μπορούμε με αυτό να φτιάξουμε Main Windows, widgets, dialog boxes και άλλα. Φτιάχνουμε πολύ εύκολα signals που μπορούμε να τα συνδέσουμε σε αντικείμενα του GUI που να αλλάζουν την κατάσταση άλλων. Μπορούμε παραδείγματος χάρη να έχουμε δύο μπάρες επιλογής που η τιμή της μίας να αλλάζει με οποιοδήποτε τρόπο την τιμή της άλλης μόνο με μερικά click και χωρίς καθόλου κώδικα. Μπορούμε να βάζουμε έτοιμα widgets στο πρόγραμμά μας και επεξεργαζόμαστε όλα τα στοιχεία αυτών ανάλογα με το πώς μας βολεύει και πολλά άλλα.
Τέλος το designer δημιουργεί αρχεία τύπου .ui τα οποία μπορούμε με πολύ απλή μετατροπή να γίνουν κώδικας c++, java, python ή άλλος και έτσι μπορούμε να έχουμε ένα εργαλείο για οποιαδήποτε γλώσσα θέλουμε να χρησιμοποιήσουμε.

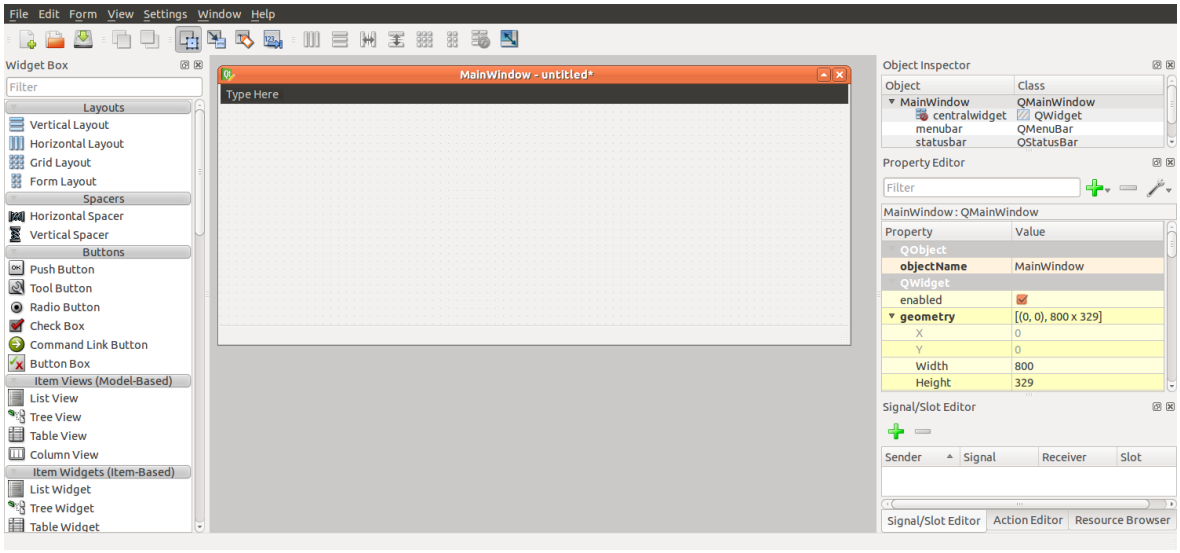


25. QT4 Designer

4.3.1 Ανάπτυξη GUI

Δημιουργώντας ένα νέο Main Window μας εμφανίζει το παρακάτω default

παράθυρο



26. Κεντρικό παράθυρο του QT4 Designer

Έχουμε ένα άδειο παράθυρο χωρίς όνομα, με κενό menubar και statusbar το οποίο μπορούμε να το κάνουμε resize, όπως ακριβώς θα κάναμε και σε ένα οποιοδήποτε παράθυρο. Αριστερά βλέπουμε τα αντικείμενα ή αλλιώς widget που μπορούμε να εισάγουμε στο παράθυρο και στο δεξί panel βλέπουμε τρία πεδία. Έχουμε λοιπόν:

→ το Object Inspector που έχει όλα τα αντικείμενα μας

→ το Property Editor που έχει όλες τις παραμέτρους του object που έχουμε επιλεγμένο και

→ το Signal/Slot Editor που μας βοηθάει να φτιάξουμε εύκολα ένα signal ή slot και να τα συνδέσουμε σε κάποιο αντικείμενο.

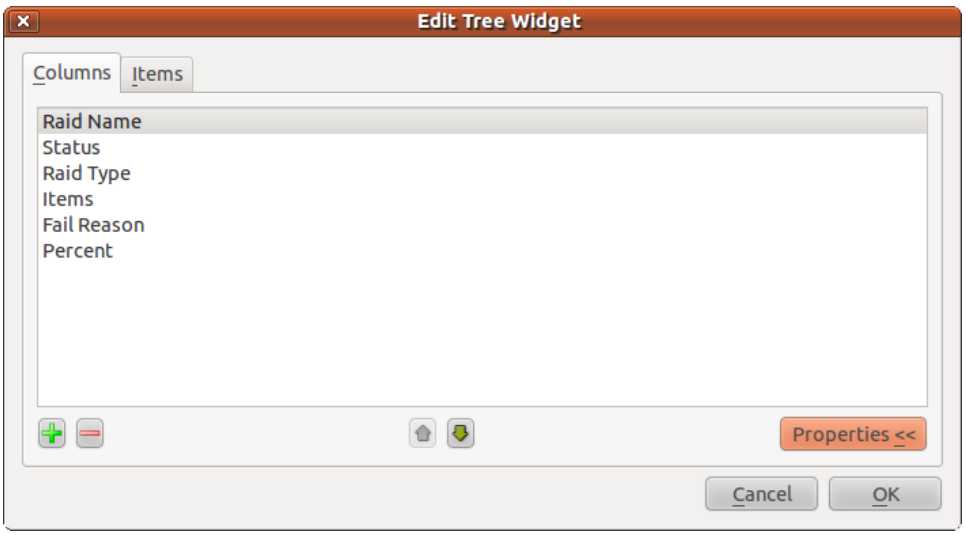
Το παράθυρο μας θέλουμε να είναι απλό και περιεκτικό, χωρίς να καταλαμβάνει ιδιαίτερα μεγάλο χώρο, ώστε αν θέλει κάποιος να μπορεί να το αφήνει ανοιχτό χωρίς να του δημιουργεί ιδιαίτερο πρόβλημα. Αρχικά φτιάχνουμε τα βασικά στοιχεία του παραθύρου:

1) Αλλάζουμε το όνομα του MainWindow απο το WindowTitle που θα βρούμε στο object inspector σε “Raid Status Indicator” που θέλουμε να εμφανίζεται.
2) Κάνουμε resize στο μέγεθος που επιθυμούμε
3) Αφαιρούμε το status bar για να μην καταλαμβάνει άχρηστο χώρο, από την στιγμή που δεν θα το χρησιμοποιήσουμε
4) Φτιάχνουμε το βασικό File menu κάνοντας add τις επιλογές minimise και

exit με shortcut key σε όλα.

5) Βάζουμε tooltip στο πρόγραμμα με βασικές πληροφορίες για το πρόγραμμα και ποιος το έχει φτιάξει.

Στη συνέχεια πρέπει να βάλουμε μία δομή για να εμφανίζει τα δεδομένα μας. Για αυτό το λόγο θα χρησιμοποιήσουμε ένα Tree Widget απο το αριστερό panel. Αυτό το widget θα είναι και το κεντρικό μας, οπότε θέλουμε να πιάνει όλο το παράθυρο. Επίσης θέλουμε να ακολουθεί και το μέγεθος του κεντρικού παραθύρου σε περίπτωση που αλλάξουμε το μέγεθος του. Για να το κάνουμε αυτό ενεργοποιούμε το “Layout in a Grid” κάνοντας δεξί click στο παράθυρο μας και αλλάζουμε το size policy του widget σε “expanding” και στις δύο διαστάσεις του.
Προχωρώντας παρακάτω, το επόμενο βήμα είναι να φτιάξουμε στήλες για τα δεδομένα μας και να βάλουμε τίτλους σε αυτές. Έτσι κάνουμε edit items στο widgets και προσθέτουμε όλα τα στοιχεία που έχουμε κρατήσει από τους ελέγχους που κάναμε στο πρόγραμμά μας.



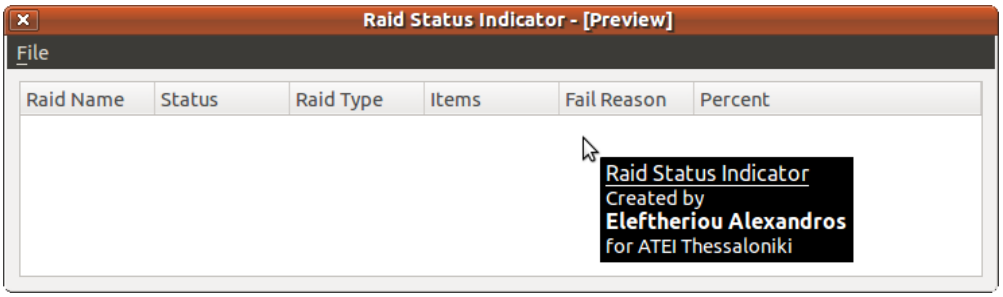
27. Μορφοποίηση των στοιχείων του tree widget

Έτσι έχουμε

1) Το όνομα του raid array (πχ md0)
2) Το status του raid array (πχ ready)
3) Το είδος του raid που (πχ raid 5)
4) Τα στοιχεία από τα οποία αποτελείται (πχ /dev/sdb /dev/sdc)

5) Το είδος του failure σε περίπτωση που υπάρχει (πχ stopped)
6) Το ποσοστό τη διεργασίας, αν διενεργείται πχ resync ή recover

Αφού κάνουμε όλα τα παραπάνω θα έχουμε ένα παράθυρο με ένα “File” menu τον τίτλο “Raid Status Indicator”, το κεντρικό μας widget με τα raid arrays και το tooltip με τα στοιχεία μας. Μπορούμε να κάνουμε Preview απο το designer για να δούμε πώς θα φαίνεται το παράθυρο μας πριν το χρησιμοποιήσουμε.



28. Προεπισκόπηση του παραθύρου μας

Βλέποντας ένα ικανοποιητικό αποτέλεσμα στο preview συνεχίζουμε στην εισαγωγή αυτού στο πρόγραμμά μας. Πρώτα αποθηκεύουμε το πρόγραμμα σε μορφή .ui που είναι μια γενική μορφή τύπου xml με όλα τα στοιχεία που αποτελούν το γραφικό περιβάλλον μας. Αυτή η γενική μορφή μπορεί να μετατραπεί πολύ εύκολα και να χρησιμοποιηθεί σε αρκετές γλώσσες προγραμματισμού. Εμείς θα χρησιμοποιήσουμε το tool pyuic4 για να το μετατρέψουμε σε κώδικα python. Χρησιμοποιώντας το pyuic4 με τον εξής τρόπο “pyuic4 gui4.ui -o gui4.py” μετατρέπει το gui4.ui αρχείο που αποθηκεύσαμε από το Designer σε κώδικα python και τον αποθηκεύει στο gui4.py. Για να χρησιμοποιήσουμε τώρα το gui4.py από το πρόγραμμά μας αρκεί να κάνουμε import το gui4.py, να φτιάξουμε ένα αντικείμενο τύπου Ui_MainWindow που είναι η κλάση που περιέχει το gui4.py και να το χρησιμοποιήσουμε αυτό για φτιάξουμε το κεντρικό μας παράθυρο. Έχοντας ενσωματώσει το γραφικό αυτό περιβάλλον στο πρόγραμμά μας μπορούμε να δούμε το τελικό αποτέλεσμα.

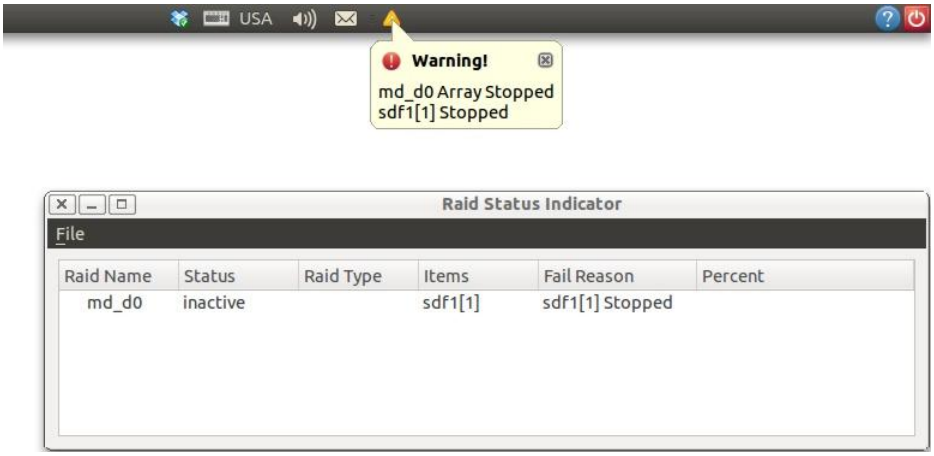
4.3.2 Δοκιμή εμφάνισης προγράμματος σε διάφορες καταστάσεις

Θα δοκιμάσουμε τώρα κάποιες καταστάσεις του προγράμματος για να δούμε αν ανταποκρίνεται στις απαιτήσεις μας. Αρχικά θα δοκιμάσουμε ένα raid 0 το οποίο είναι λειτουργικό χωρίς προβλήματα.

29. Εμφάνιση σωστής λειτουργία του Raid 0

Βλέπουμε ότι εμφανίζει κανονικά όλα στοιχεία του array και το tray icon επάνω είναι πράσινο checkmark. Αυτό μας δείχνει ότι όλα δουλεύουν σωστά.

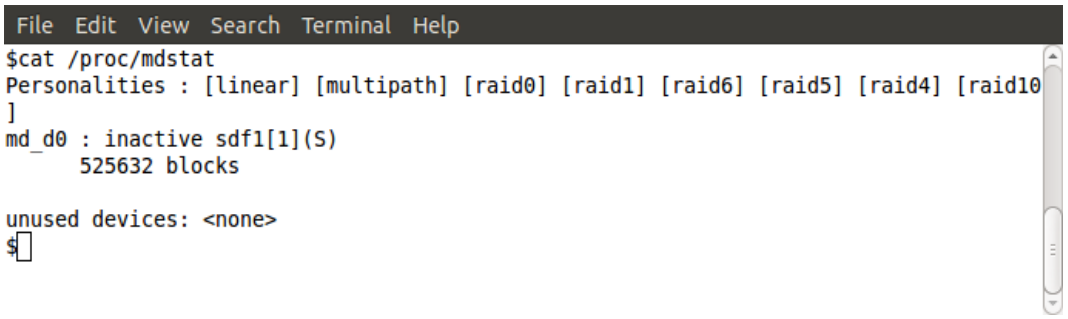
Σε περίπτωση που έχουμε failed drive σε raid 0 θα έχουμε το παρακάτω.



30. Εμφάνιση stopped Raid 0

Το εικονίδιο του tray icon από πράσινο που ήταν στην προηγουμένη αλλάζει σε κίτρινο θαυμαστικό και μας ειδοποιεί με balloon tip για το πρόβλημα που υπάρχει. Επειδή δεν έχει βρει το δεύτερο δίσκο που αποτελείται το array δεν ξεκινάει το raid

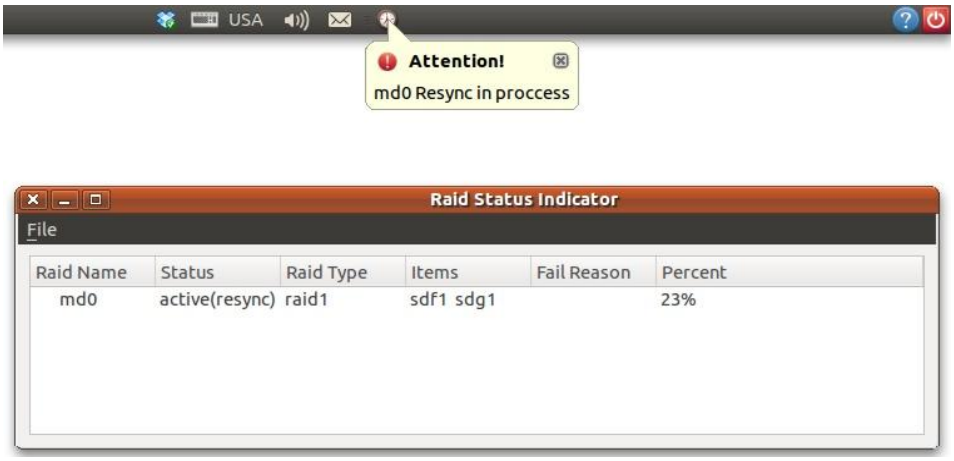
array. Έτσι βλέπουμε το sdf1 ως stopped και το array ως inactive. Στο raid type δεν βλέπουμε να έχει τον τύπο του raid επειδή δεν υπάρχει ούτε στο /proc/mdstat.



31. Έλλειψη τύπου RAID στο mdstat

Αυτό συμβαίνει στις περιπτώσεις που το array έιναι inactive. Επομένως το πρόγραμμα άντλησε σωστά τα δεδομένα και μας τα εμφάνισε.

Σε περίπτωση που έχουμε resync εμφανίζεται:



32. Εμφάνιση Resync

Όλα τα πεδία εμφανίζονται κανονικά με το ποσοστό να αλλάζει σε πραγματικό χρόνο. Το πεδίο Fail Reason δεν εμφανίζει κάτι επειδή το resync δεν το έχουμε βάλει να θεωρείται ως failure και το status είναι κανονικά active.

Σε περίπτωση recover ενός raid 5 θα έχουμε τα παρακάτω:

33. Εμφάνιση recovering σε Raid 5

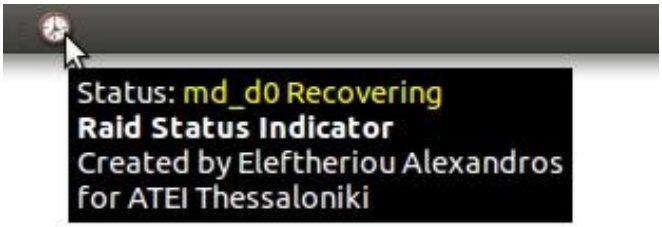
Και σε αυτή την περίπτωση βλέπουμε ότι εμφανίστηκαν όλα τα πεδία εκτός από το fail reason για τον ίδιο λόγο που είπαμε και στο resync. Το ποσοστό εκτέλεσης τι διεργασίας και σε αυτή τη περίπτωση ανανεώνεται αυτόματα.

Τέλος θα δούμε και το tooltip του tray icon αν λειτουργεί σωστά σε τρείς περιπτώσεις:

1) Όταν όλα τα raid arrays είναι πλήρως λειτουργικά

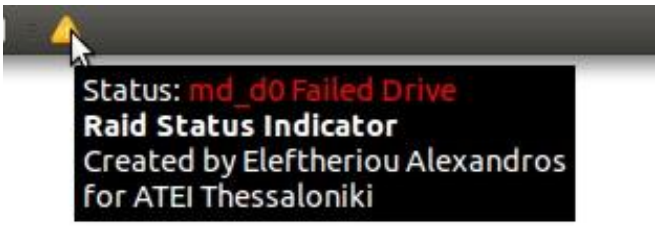
34. Tooltip σε κατάσταση healthy

2) Όταν γίνεται recovery σε ένα raid array



35. Tooltip σε κατάσταση recovering

3) Όταν έχουμε ένα προβληματικό/failed drive



36. Tooltip με failed drive

Όπως βλέπουμε εμφανίζει κανονικά το status των raid arrays με πληροφορίες όταν υπάρχει κάποιο πρόβλημα. Επίσης εμφανίζει διαφορετικά χρώματα ανάλογα με την κρισιμότητα της ενημέρωσης, όπως ακριβώς το είχαμε αποθηκεύσει με HTML στη μεταβλητή failtray.

5. Συμπεράσματα και Προτάσεις.

Όπως διαπιστώσαμε οι διατάξεις Raid έχουν πάρα πολλά πλεονεκτήματα και σε πολλές περιπτώσεις μας γλυτώνουν από καταστροφικά αποτελέσματα. Ειδικά αν το σύστημα στο οποίο εφαρμόζεται εκτελεί διεργασίες καίριας σημασίας για τη λειτουργία μιας εταιρίας, παραδείγματος χάριν, μπορεί να μας γλυτώσει από μια πολύωρη διακοπή λειτουργίας ή απώλεια σημαντικών δεδομένων. Τέτοια παραδείγματα θα μπορούσαν να είναι:

1) Web Server ενός ηλεκτρονικού καταστήματος: Σε περίπτωση που σταματούσε να δουλεύει ο δίσκος του λειτουργικού συστήματος ή της ηλεκτρονικής σελίδας (αν ήταν διαφορετικός) θα είχαμε κλείσιμο της σελίδας μέχρι να επαναφέρουμε το backup μας και να ξανά ρυθμίσουμε αρκετά κομμάτια του λειτουργικού. Αυτό θα σήμαινε φοβερή οικονομική απώλεια και ίσως και απώλεια πελατών. Σε περίπτωση όμως που είχαμε ένα raid 1 στο συγκεκριμένο δίσκο δεν θα είχαμε καμία διακοπή λειτουργίας του Server
2) Σε ένα Server συστήματος ERP: Σε αυτή την περίπτωση θα μπορούσε να σημαίνει απώλεια δεδομένων πελατολογίου, πληροφορίες αποθεμάτων και άλλων. Ταυτόχρονα πιθανότατα να μη μπορούσαν να εκδοθούν τιμολόγια που θα σήμαινε φοβερές καθυστερήσεις στη λειτουργία της επιχείρησης. Και σε αυτή την περίπτωση θα είχαν αποφευχθεί τέτοια προβλήματα με συστοιχίες raid 1 ή raid 5
3) Ένα ακόμα παράδειγμα θα μπορούσε να είναι ένα τηλεφωνικό κέντρο βασισμένο σε υπολογιστή όπως ένα asterisk. Θα είχαμε ολοκληρωτική διακοπή της τηλεφωνικής επικοινωνίας της εταιρίας που θα σήμαινε από δυσλειτουργία έως και παύση λειτουργίας της επιχείρησης και φυσικά πτώση του κύρους της εταιρίας.

Βλέπουμε ότι το Raid μπορεί να μας γλιτώσει από σημαντικά προβλήματα άλλα ταυτόχρονα φέρνει και κάποια δικά του.

Σε ένα harware raid ένα πρόβλημα στον raid controller, ακόμα και αν οι δίσκοι δεν είχαν κανένα απολύτως πρόβλημα, σημαίνει πάλι διακοπή λειτουργίας του συστήματος και σε κάποιες περιπτώσεις απώλεια δεδομένων.

Σε software raid εξαλείφεται και αυτό το πρόβλημα και απλά έχουμε μια μικρή μείωση ταχύτητας και κάποια χρήση επεξεργαστικής ισχύος.

Αυτό όμως που είναι κοινό και στις δύο περιπτώσεις και είδαμε και αναλυτικά καθ' όλη την ανάπτυξη της πτυχιακής εργασίας είναι ότι δεν έχουμε καμία απολύτως ενημέρωση σε περίπτωση που υπάρχει κάποιο πρόβλημα με το raid μας. Αυτό ταυτόχρονα με τη σιγουριά που μας προσφέρει ένα software raid θα μπορούσε να οδηγήσει σε καταστροφικά αποτελέσματα. Θα μπορούσαμε παραδείγματός χάρη να μη κρατάμε backup όντας σίγουροι για την αξιοπιστία του raid μας και ξαφνικά να βρισκόμασταν με δύο χαλασμένους δίσκους, ένα μη λειτουργικό raid και απώλεια όλων των δεδομένων μας. Με την ανάπτυξη αυτού του προγράμματος ερχόμαστε να προσθέσουμε ακόμα μια προφύλαξη από τέτοιες καταστάσεις. Έχοντας φτιάξει ένα πρόγραμμα που θα ελέγχει σε τακτά χρονικά διαστήματα την κατάσταση των raid μας, καταναλώνοντας μηδαμινούς πώρους συστήματος και όντας αρκετά διακριτικό χωρίς να πιάνει χώρο στην επιφάνεια εργασίας μας και να μας ενοχλεί, είμαστε ξέγνοιαστοι ότι θα ενημερωθούμε αμέσως για οποιοδήποτε πρόβλημα δημιουργηθεί.

Η δημιουργία του προγράμματος δεν ήρθε όμως χωρίς προβλήματα. Το σημαντικότερο πρόβλημα ήταν η ανάπτυξη ενός αξιόλογου γραφικού περιβάλλοντος χωρίς περιορισμούς και προβλήματα. Αυτό έγινε πολύ εντονότερο με την έλλειψη εγγράφων και αναφορών ανάπτυξης GUI με Qt4 και python. Οι αναφορές που υπήρχαν ήταν κυρίως για C++ και λιγότερο για Java, το οποίο έκανε ιδιαίτερα δύσκολη την μετάφραση της χρήσης signals και slots όπως και την ενσωμάτωση του gui που δημιουργεί η Qt στο πρόγραμμα μας. Ταυτόχρονα η έλλειψη παραμετροποίησης της main loop μας οδηγούσε σε φοβερά προβλήματα λειτουργικότητας όπως και υψηλής χρήσης πόρων, το οποίο μας ανάγκασε να δημιουργήσουμε μια δικιά μας loop για να επιλύσουμε αυτά τα προβλήματα. Ο κώδικας επίσης σιγά - σιγά βελτιστοποιήθηκε και μίκρυνε δείχνοντας για ακόμα μια φορά την δύναμη της python. Το επόμενο δύσκολο αλλά και πολύ χρονοβόρο

τμήμα ήταν η δημιουργία όλων των ειδών raid καθώς και η εξομοίωση όλων των καταστάσεων και προβλημάτων που μπορεί να προκύψουν. Αυτό ξεκινώντας με πραγματικούς σκληρούς δίσκους αποδείχθηκε δύσκολη υπόθεση οπότε εδώ η λύση ήρθε με χρήση flash drives, που έκαναν πολύ ευκολότερη την εξομοίωση όλων των καταστάσεων.

Η χρήση της python παρόλα τα προβλήματα που μας δημιούργησε στο γραφικό περιβάλλον, μας αντάμειψε με ένα πολύ καθαρό και συνεπτυγμένο κώδικα που είναι και ταυτόχρονα πολύ εύκολα παραμετροποίησιμος. Ο κώδικας αυτός μπορεί να εκτελεστεί σε οποιοδήποτε λειτουργικό (Linux, Windows ..) οποιασδήποτε αρχιτεκτονικής (x32 ή x64) χωρίς κανένα πρόβλημα. Ταυτόχρονα επειδή δεν γίνεται compile το πρόγραμμα, μπορεί όποιος θέλει να δει τον κώδικα και να τον παραμετροποιήσει ανάλογα με τις απαιτήσεις του.

Το πρόγραμμα αυτό επίσης μπορεί να χρησιμοποιηθεί σαν βάση για φτιαχτεί ένα μεγαλύτερο project με επιπλέον δυνατότητες. Θα ήταν πολύ χρήσιμο να το τρέχουμε σε ένα απλό υπολογιστή με λειτουργικό windows ή linux και να μας ενημερώνει για την κατάσταση όλων των server μας, συνδέοντας με αυτούς μέσω δικτύου. Αυτό μπορεί να γίνει ακόμα πιο απλά αν έχουμε samba share στους server μας και απλά αλλάξουμε το path από όπου διαβάζει το mdstat στον κώδικα. Θα μπορούσε ακόμα με μικρές αλλαγές, λόγο της python, να τρέξει ακόμα σε κινητό τηλέφονο που να υποστηρίζει python ή android. Έτσι θα είχαμε έλεγχο των raid μας όπου και να ήμασταν.
Σίγουρα υπάρχουν πολλές ακόμα εφαρμογές που δεν καλύφθηκαν από αυτή την πτυχιακή και ίσως και bug που δεν καταφέραμε να εντοπίσουμε, αλλά πετύχαμε να καλύψουμε ένα μεγάλο κενό ασφαλείας που δεν έχει δοθεί όση έμφαση που αξίζει. Ας ελπίσουμε αυτό το πρόγραμμα να είναι μια αρχή για ένα πολύ μεγαλύτερο open source project που θα κάνει τη ζωή τόσο των linux administrator όσο και χρηστών με raid στον προσωπικό υπολογιστή τους αρκετά ευκολότερη, γλιτώνοντας τους χρόνο και αποφεύγοντας άσχημες καταστάσεις

6. Αναφορές

[1] HP, “Solid state storage technology for ProLiant servers”, 2nd edition, http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01580706/c01580706.pdf

[2] “Raid History Information”, http://www.krollontrack.com/data-recovery/raid-history-information/

[3] Jakob Østergaard and Emilio Bueso, "The Software-RAID HOWTO", http://tldp.org/HOWTO/Software-RAID-HOWTO.html

[4] “Software Vs Hardware RAID”, http://www.cyberciti.biz/tips/raid-hardware-vs-raid-software.html

[5] Adaptec Storage Solutions, “Hardware RAID vs. Software RAID: Which Implementation is Best for my Application?”, http://www.adaptec.com/nr/rdonlyres/14b2fd84-f7a0-4ac5-a07a-214123ea3dd6/0/4423_sw_hwraid_10.pdf

[6] Python py3exe, http://www.py2exe.org/

[7] Mark Lutz (2009) “Learing Python”, 4th edition, USA: O'Reilly

7. Βιβλιογραφία

- Mark Lutz (2009) “Learing Python”, 4th edition, USA: O'Reilly
- Daniel J. Barrett (2004) “Linux Pocket Guide”, USA: O'Reilly
- Python Documentation. http://docs.python.org/
- Qt4 Online Reference Documentation. http://doc.qt.nokia.com/
- “About PyQt4”, http://wiki.python.org/moin/PyQt4
- “The PyQt4 Tutorial”, http://zetcode.com/tutorials/pyqt4/
- “Introduction to PyQt4”, http://www.rkblog.rk.edu.pl/w/p/introduction-pyqt4/
- Alex Fedosov, “Creating GUI Applications in Python with QT”, http://www.cs.usfca.edu/~afedosov/qttut/
- “Linux: Why Sofware Raid?”, http://linux.yyz.us/why-software-raid.html
- Wikipedia, “RAID”, http://en.wikipedia.org/wiki/Redundant_array_of_independent_disks
- “Software Raid | Raid Arrays | mdadm on Linux”, http://www.review-ninja.com/2009/05/software-raid-raid-arrays-mdadm-on.html
- Open Icon Library, http://openiconlibrary.sourceforge.net/
- “Intel's X25-M solid-state drive”, http://techreport.com/articles.x/15433

8. Παράρτημα

8.1 Κώδικας κεντρικού προγράμματος

```
#!/usr/bin/env python

import sys

import time

#from gui3 import *

from gui4 import *



def iconActivated(reason):

    """Shows the MainWindow if the TrayIcon is either single or double Clicked"""

    if reason == 2 or reason == 3:          # an ginei diplo h mono click na emfanistei to main window

        MainWindow.show()
```

```
def check_status():

    """Reads /proc/mdstat and identifies all the arrays. It then checks the status of every array and

        1) It adds every array with its status to treel widget in ui

        2) If there is a problem it displays a balloon tip with the warning/attention and changes the icon accordingly

        3) It resizes the treel list widget to match MainWindow's dimensions

    """

    global fail                         #gia na elenxoume an allaksei to status

    global wait

    global failtray

    if wait >0:

        wait -=1

    nofailure=1

    failreason=""

    failtype=""

    failed=""
```

```
        raidnum = 0                     #arithmos arrays

        mdstat = open(r'/proc/mdstat')

#       mdstat = open(r'resync')        #arxeia gia dokimes emfanisis diaforon
error

#       mdstat = open(r'recovery')

#       mdstat = open(r'autoread')

        lines = mdstat.readlines()      #diavasma arxeiou

        mdstat.close()

        raids = []                      #pinakas me ta array

        a = 0                           # diavasma arxeioy se pinaka

        faildev=[]

        while a<len(lines):

            b = 0

                if lines[a].startswith('md'):           #anagnorisi array

                        raidnum+=1

                        temp=lines[a].rstrip()

                        temp=temp.replace(' : ',' ')    # Afairesi tou : gia efkoloteri
diaxeirisi
```

```
                temp=temp.split(' ')                # metatropi tis gramis se
stoixeia pinaka

                raids.append([])                    #apothikefsi ton stixion tou
raid stin proti grami tou raids

                raids[(raidnum-1)].append(temp[0])          #onoma array

                raids[(raidnum-1)].append(temp[1])          #status array

                if temp[2].startswith("hd") or temp[2].startswith("sd"):

                        raids[(raidnum-1)].append("")       #keno raidtype

                elif temp[2].startswith("("):               #elegxei an iparxei
kapoio extra status opos (auto-read-only)

                        raids[(raidnum-1)].append(temp[3]+temp[2])

                else:

                        raids[(raidnum-1)].append(temp[2])  #raid type



                c=2

                hdds=''

                failreason=''

                failtype=''

                while c<len(temp):
```

```
                    if temp[c].startswith("hd") or temp[c].startswith("sd"):
                        if temp[c].endswith("(S)"):
                            hdds += temp[c][:-3]+"  "
                            failreason+= temp[c][:-3]+" Stopped "
                            failtype="Array Stopped"
                            failtray='<font
color="red">'+raids[(raidnum-1)][0]+' Array Stopped</font>'
                            nofailure =0
                        elif temp[c].endswith("(F)"):
                            hdds += temp[c][:-3]+"  "
                            failreason+= temp[c][:-3]+" Failed "
                            failtype="Failed Drive"
                            failtray='<font
color="red">'+raids[(raidnum-1)][0]+' Failed Drive</font>'
                            nofailure =0
                        else:
                            hdds += temp[c][:-3]+"  "
                    c +=1
```

```
                raids[(raidnum-1)].append(hdds)
                if failreason != "":            #prosthiki pediou Fail Reason
                    raids[(raidnum-1)].append(failreason)
                    sysTray.setIcon( QtGui.QIcon('Warning.png'))
                    if failtype != fail and wait==0:       #elegxos gia na
emfanizete mono mia fora baloon

        sysTray.showMessage("Warning!",raids[(raidnum-1)][0]+"
"+failtype+"\n"+failreason,2,int(9000))
                        wait=20
                        fail=failtype
                else:
                    raids[(raidnum-1)].append("")
```

```
                if 'resync' in lines[a]:                                #elenxei an ginete
resync

                        nofailure=0

                    sysTray.setIcon( QtGui.QIcon('Clock.png'))

                        if '%' in lines[a]:

                                temp=lines[a].split(' ')                #xorizei tis
plirofories tou raid me delimiter to ' '

                                raids[(raidnum-1)][1]+='(resync)'       #enimeronei to
status

                                if temp[10][:-3] == '':                 #elenxos an
einai monopsifios

                                        raids[(raidnum-1)].append(temp[11][:-3]+"%")

                                else:

                                        raids[(raidnum-1)].append(temp[10][:-3]+"%")
        #apothikevei to pososto sto telos tou pinaka



                                failtype="Resync"

                                failtray='<font color="yellow">'+raids[(raidnum-1)][0]+'
Resync</font>'

                                if failtype != fail and wait==0:
```

sysTray.showMessage("Attention!",raids[(raidnum-1)][0]+" Resync in proccess",2,int(9000))

wait=20

fail=failtype
#alagi tis metavlitis pou krataei to proigoumeno status

elif 'DELAYED' in lines[a]:

raids[(raidnum-1)][1]+='(Delayed resync)' #elenxei an iparxei delayed resync kai kanei update to status

failtype="Delayed Resync" #elegxos an exei alaksei to status

failtray=''+raids[(raidnum-1)][0]+' Delayed Resync'

if failtype != fail and wait==0:

sysTray.showMessage("Warning!",raids[(raidnum-1)][0]+" "+failtype,2,int(9000))

fail=failtype
#alagi tis metavlitis pou krataei to proigoumeno status

wait=20

```
                if 'recovery' in lines[a]:                              #elenxei an ginete
recovery

                        nofailure=0

                    sysTray.setIcon( QtGui.QIcon('Clock.png'))

                        if '%' in lines[a]:

                                temp=lines[a].split(' ')                #xorizei tis plirofories
tou raid me delimiter to ' '

                                raids[(raidnum-1)][1]+='(recovery)'         #enimeronei to
status

                                if temp[10][:-3] == '':

                                        raids[(raidnum-1)].append(temp[11][:-3]+"%")

                                else:

                                        raids[(raidnum-1)].append(temp[10][:-3]+"%")
        #apothikevei to pososto sto telos tou pinaka



                        failtype="Recovering"
        #elegxos an exei alaksei to status

                        failtray='<font color="yellow">'+raids[(raidnum-1)][0]+'
Recovering</font>'

                        if failtype != fail and wait==0:
```

```
                    sysTray.showMessage("Attention!",raids[(raidnum-
1)][0]+" "+failtype,2,int(9000))

                    wait=20

                    fail=failtype
        #alagi tis metavlitis pou krataei to proigoumeno status



            a+=1

        if nofailure == 1:                          #epanaferei to eikonidio an
den iparxei provlima

            sysTray.setIcon( QtGui.QIcon('Checkmark.png'))

            fail=""

            failtray='<font color="green">Healthy</font>'



        ui.treel.clear()

        size1=len(raids)

        while size1 > 0:

            size1 -=1

            item=QtGui.QTreeWidgetItem(raids[size1])
```

```
                ui.treel.addTopLevelItem(item)




#       ui.treel.resize(MainWindow.width()-1,MainWindow.height()-25)       # kanei
resize to treeview stis diastaseis tou mainwindow

        sysTray.setToolTip("""Status: """+ failtray +"""<BR><B>Raid Status
Indicator</B><BR>Created by Eleftheriou Alexandros<BR>for ATEI
Thessaloniki""")




c=0

wait=0

size=1

fail=""                         #global metavliti gia na elenxoume an allazei to status

failtray=""




app = QtGui.QApplication(sys.argv)

MainWindow = QtGui.QMainWindow()
```

```
#-------------------TRAY

sysTray = QtGui.QSystemTrayIcon()

sysTray.setIcon( QtGui.QIcon('Checkmark.png'))

sysTray.connect(sysTray,
QtCore.SIGNAL("activated(QSystemTrayIcon::ActivationReason)"), iconActivated)

menu = QtGui.QMenu()

exitAction = menu.addAction("Exit")

showwindow = menu.addAction("Show")

exitAction.triggered.connect(sys.exit)

showwindow.triggered.connect(MainWindow.show)

sysTray.setContextMenu(menu)

sysTray.setVisible(True)

#-----------------TRAY



ui = Ui_MainWindow()

ui.setupUi(MainWindow)
```

```
ui.actionExit.triggered.connect(sys.exit)

ui.action_minimise.triggered.connect(MainWindow.hide)

MainWindow.show()



e=101

while c<200:

        if e>100:                               #tsekarei to arxeio mono kathe 100
enimeroseis tou GUI

                check_status()

                e=0

      QtGui.QApplication.processEvents()        #kanei update to GUI

        time.sleep(0.01)

        e+=1




sys.exit(app.exec_())
```

8.2 Κώδικας Γραφικού περιβάλλοντος σε μορφή ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>733</width>
    <height>190</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Raid Status Indicator</string>
```

```
 </property>
 <widget class="QWidget" name="centralwidget">
  <layout class="QGridLayout" name="gridLayout">
   <item row="0" column="0">
    <widget class="QTreeWidget" name="treel">
     <property name="toolTip">
      <string>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.0//EN&quot; &quot;http://www.w3.org/TR/REC-html40/strict.dtd&quot;&gt;

&lt;html&gt;&lt;head&gt;&lt;meta name=&quot;qrichtext&quot; content=&quot;1&quot; /&gt;&lt;style type=&quot;text/css&quot;&gt;

p, li { white-space: pre-wrap; }

&lt;/style&gt;&lt;/head&gt;&lt;body style=&quot; font-family:'Ubuntu'; font-size:11pt; font-weight:400; font-style:normal;&quot;&gt;

&lt;p style=&quot; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;span style=&quot; font-size:12pt; text-decoration: underline;&quot;&gt;Raid Status Indicator&lt;/span&gt;&lt;/p&gt;

&lt;p style=&quot; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;Created by &lt;/p&gt;

&lt;p style=&quot; margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;&quot;&gt;&lt;span style=&quot; font-
```

size:12pt; font-weight:600;">Eleftheriou Alexandros</p>

<p style=" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;">for ATEI Thessaloniki</p></body></html></string>

</property>

<property name="sortingEnabled">

<bool>false</bool>

</property>

<column>

<property name="text">

<string>Raid Name</string>

</property>

<property name="font">

<pointsize>11</pointsize>

</property>

</column>

```
<column>
 <property name="text">
  <string>Status</string>
 </property>
</column>
<column>
 <property name="text">
  <string>Raid Type</string>
 </property>
</column>
<column>
 <property name="text">
  <string>Items</string>
 </property>
</column>
<column>
 <property name="text">
```

```
        <string>Fail Reason</string>
       </property>
      </column>
      <column>
       <property name="text">
        <string>Percent</string>
       </property>
      </column>
     </widget>
    </item>
   </layout>
  </widget>
  <widget class="QMenuBar" name="menuBar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
```

```
    <width>733</width>
    <height>25</height>
   </rect>
  </property>
  <widget class="QMenu" name="menuMenu">
   <property name="title">
    <string>&amp;amp;File</string>
   </property>
   <addaction name="action_minimise"/>
   <addaction name="actionExit"/>
  </widget>
  <addaction name="menuMenu"/>
 </widget>
 <action name="actionExit">
  <property name="text">
   <string>E&amp;amp;xit</string>
  </property>
```

```
 </action>
 <action name="actionAbout">
  <property name="text">
   <string>About</string>
  </property>
 </action>
 <action name="action_minimise">
  <property name="text">
   <string>&amp;amp;Minimise</string>
  </property>
 </action>
</widget>
<resources/>
<connections/>
</ui>
```

8.3 Κώδικας Γραφικού περιβάλλοντος μετά τη μετατροπή σε python

```
# -*- coding: utf-8 -*-



# Form implementation generated from reading ui file 'gui4.ui'

#

# Created: Sat Nov 12 20:49:54 2011

#      by: PyQt4 UI code generator 4.7.4

#

# WARNING! All changes made in this file will be lost!



from PyQt4 import QtCore, QtGui



class Ui_MainWindow(object):

    def setupUi(self, MainWindow):

        MainWindow.setObjectName("MainWindow")
```

```
MainWindow.resize(733, 190)

self.centralwidget = QtGui.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.gridLayout = QtGui.QGridLayout(self.centralwidget)

self.gridLayout.setObjectName("gridLayout")

self.treel = QtGui.QTreeWidget(self.centralwidget)

self.treel.setObjectName("treel")

self.gridLayout.addWidget(self.treel, 0, 0, 1, 1)

MainWindow.setCentralWidget(self.centralwidget)

self.menuBar = QtGui.QMenuBar(MainWindow)

self.menuBar.setGeometry(QtCore.QRect(0, 0, 733, 25))

self.menuBar.setObjectName("menuBar")

self.menuMenu = QtGui.QMenu(self.menuBar)

self.menuMenu.setObjectName("menuMenu")

MainWindow.setMenuBar(self.menuBar)

self.actionExit = QtGui.QAction(MainWindow)

self.actionExit.setObjectName("actionExit")
```

```
        self.actionAbout = QtGui.QAction(MainWindow)

        self.actionAbout.setObjectName("actionAbout")

        self.action_minimise = QtGui.QAction(MainWindow)

        self.action_minimise.setObjectName("action_minimise")

        self.menuMenu.addAction(self.action_minimise)

        self.menuMenu.addAction(self.actionExit)

        self.menuBar.addAction(self.menuMenu.menuAction())



        self.retranslateUi(MainWindow)

        QtCore.QMetaObject.connectSlotsByName(MainWindow)



    def retranslateUi(self, MainWindow):

        MainWindow.setWindowTitle(QtGui.QApplication.translate("MainWindow",
"Raid Status Indicator", None, QtGui.QApplication.UnicodeUTF8))

        self.treeI.setToolTip(QtGui.QApplication.translate("MainWindow",
"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\"
\"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"

"<html><head><meta name=\"qrichtext\" content=\"1\" /><style
type=\"text/css\">\n"
```

"p, li { white-space: pre-wrap; }\n"

"</style></head><body style=\" font-family:\'Ubuntu\'; font-size:11pt; font-weight:400; font-style:normal;\">\n"

"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-size:12pt; text-decoration: underline;\">Raid Status Indicator</p>\n"

"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">Created by </p>\n"

"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\"><span style=\" font-size:12pt; font-weight:600;\">Eleftheriou Alexandros</p>\n"

"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">for ATEI Thessaloniki</p></body></html>", None, QtGui.QApplication.UnicodeUTF8))

self.treeI.setSortingEnabled(False)

self.treeI.headerItem().setText(0, QtGui.QApplication.translate("MainWindow", "Raid Name", None, QtGui.QApplication.UnicodeUTF8))

self.treeI.headerItem().setText(1, QtGui.QApplication.translate("MainWindow", "Status", None, QtGui.QApplication.UnicodeUTF8))

self.treeI.headerItem().setText(2, QtGui.QApplication.translate("MainWindow", "Raid Type", None, QtGui.QApplication.UnicodeUTF8))

```
        self.treeI.headerItem().setText(3,
QtGui.QApplication.translate("MainWindow", "Items", None,
QtGui.QApplication.UnicodeUTF8))

        self.treeI.headerItem().setText(4,
QtGui.QApplication.translate("MainWindow", "Fail Reason", None,
QtGui.QApplication.UnicodeUTF8))

        self.treeI.headerItem().setText(5,
QtGui.QApplication.translate("MainWindow", "Percent", None,
QtGui.QApplication.UnicodeUTF8))

        self.menuMenu.setTitle(QtGui.QApplication.translate("MainWindow", "&File",
None, QtGui.QApplication.UnicodeUTF8))

        self.actionExit.setText(QtGui.QApplication.translate("MainWindow", "E&xit",
None, QtGui.QApplication.UnicodeUTF8))

        self.actionAbout.setText(QtGui.QApplication.translate("MainWindow",
"About", None, QtGui.QApplication.UnicodeUTF8))

        self.action_minimise.setText(QtGui.QApplication.translate("MainWindow",
"&Minimise", None, QtGui.QApplication.UnicodeUTF8))
```