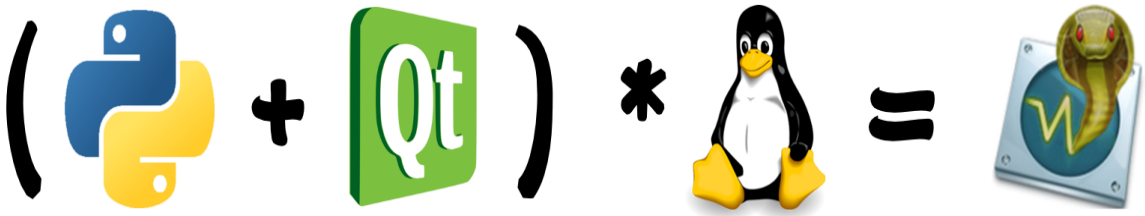
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή «task manager» για λειτουργικό σύστημα Linux και περιβάλλον εργασίας KDE



Του φοιτητή
Δημήτρη Διαμαντή
Αρ. Μητρώου: 03/2260

Επιβλέπων καθηγητής
Αντώνης Σιδηρόπουλος

Θεσσαλονίκη 2012

ΠΡΟΛΟΓΟΣ

Το θέμα της πτυχιακής εργασίας αφορά την παρακολούθηση και διαχείριση των διεργασιών του λειτουργικού συστήματος GNU/Linux καθώς και της απόδοσης και χρησιμοποίησης των βασικότερων πόρων του υπολογιστή από αυτό. Ο κύριος λόγος επιλογής αυτού του θέματος της πτυχιακής εργασίας ήταν η αγάπη για το λειτουργικό σύστημα GNU/Linux και το ελεύθερο λογισμικό γενικότερα. Επίσης ήταν και μια καλή ευκαιρία για την εκμάθηση νέων πραγμάτων όπως η γλώσσα προγραμματισμού python και η βιβλιοθήκη γραφικών Qt. Παρόλο που το θέμα αφορούσε μόνο το λειτουργικό σύστημα GNU/Linux, χάρη στη διαπλατφορμική γλώσσα προγραμματισμού και των βιβλιοθηκών που χρησιμοποιήθηκαν για την ανάπτυξη, το αποτέλεσμα είναι μια εφαρμογή που μπορεί να εγκατασταθεί και να εκτελεστεί σε όλα τα λειτουργικά συστήματα. Παρόλα αυτά η πτυχιακή εργασία θα αναλύσει και θα εμβαθύνει μόνο στις λειτουργίες και στις μεθόδους για το λειτουργικό σύστημα GNU/Linux όπως ο τίτλος επιβάλλει.

ΠΕΡΙΛΗΨΗ

Το λειτουργικό σύστημα GNU/Linux αποτελεί ένα από τα λαμπρότερα παραδείγματα Ελεύθερου Λογισμικού. Δηλαδή του λογισμικού που εκτός των κανόνων και των περιορισμών της άδειας χρήσης που το διακατέχουν ώστε να εξασφαλίσουν την ελευθερία του, η φιλοσοφία ανάπτυξης του είναι τέτοια που το αποτέλεσμα μπορεί εύκολα να ξαναχρησιμοποιηθεί, να βελτιστοποιηθεί και να επεκταθεί με περισσότερες λειτουργίες από όποιον το επιθυμεί.

Ένα από τα κυριότερα χαρακτηριστικά του GNU/Linux είναι ότι όλα τα στοιχεία που το απαρτίζουν είναι κατάλογοι και αρχεία που στην πλειοψηφία τους μπορούν να αναγνωστούν με ένα απλό κειμενογράφο. Έτσι, και όλες οι πληροφορίες του λειτουργικού συστήματος και των διεργασιών του “χρόνου εκτέλεσης” του συστήματος αποθηκεύονται σε αρχεία και καταλόγους μέσα σε ένα κύριο κατάλογο, κάτω από τη “ρίζα” του δέντρου των καταλόγων του λειτουργικού συστήματος, τον /proc. Το χαρακτηριστικό αυτό μας δίνει την δυνατότητα να απομονώσουμε και να χρησιμοποιήσουμε προγραμματιστικά όποιες πληροφορίες του συστήματος μας ενδιαφέρουν ώστε να τις παρουσιάσουμε αργότερα σε μια πιο φιλική για τον άνθρωπο μορφή.

Η γλώσσα python είναι μια από τις ισχυρότερες διαπλατφορικές, επεκτάσιμες και ελεύθερου λογισμικού γλώσσες προγραμματισμού. Επίσης και ένα από τα πλέον αναπόσπαστα κομμάτια του λειτουργικού συστήματος GNU/Linux μιας και πληθώρα εφαρμογών αυτού έχουν αναπτυχθεί σε python. Οι δυνατότητες της σε συνδυασμό τις πρότυπες και μη βιβλιοθήκες αρθρωμάτων που υπάρχουν διαθέσιμες στο διαδίκτυο επιτρέπουν στον προγραμματιστή την εύκολη ανάπτυξη ισχυρών εφαρμογών με λίγες γραμμές κώδικα.

Ο συνδυασμός των πρότυπων αρθρωμάτων της python καθώς και των psutil και pyqt μας δίνει την δυνατότητα της εύκολης ανάπτυξης εφαρμογής η οποία εμφανίζει πληροφορίες που αφορούν τις διεργασίες και την απόδοση του συστήματος σε γραφική διεπαφή χρήστη. Αναλυτικότερα, οι μέθοδοι του

αρθρώματος psutil ανασύρουν τις πληροφορίες κυρίως από τα περιεχόμενα του καταλόγου, του Λ.Σ. GNU/Linux, /proc και τις περνάνε σε μεταβλητές τις οποίες στην συνέχεια διαβάζουν οι μέθοδοι του αρθρώματος pyqt και τις εμφανίζουν σε μορφή γραφικής διεπαφής χρήστη.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Αντώνη Σιδηρόπουλο για την ανάθεση και στήριξη στην εκπόνηση της πτυχιακής εργασίας.

Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή κ. Στέφανο Χαρχαλάκη για την αρχική ανάθεση και σύλληψη τις ιδέας του θέματος της πτυχιακής.

Ευχαριστώ ακόμη τις ομάδες ανάπτυξης των έργων python, psutil, oxygenicons, pyqt, eclipse καθώς και την ευρύτερη παγκόσμια κοινότητα του ΕΛ/ΛΑΚ για την δουλειά τους ή και την στήριξη που μου παρείχαν όταν την χρειάστηκα κατά την διάρκεια της ανάπτυξης της εφαρμογής της πτυχιακής.

Τέλος θα ήθελα να ευχαριστήσω όλους τους φίλους και τα άτομα του κοντινού περιβάλλοντος μου για την ηθική ή και τεχνική στήριξη που μου παρείχαν μέχρι το τέλος της ανάπτυξης και της εκπόνησης της πτυχιακής εργασίας.

Κατάλογος περιεχομένων

ΕΙΣΑΓΩΓΗ

Ο σκοπός της πτυχιακής εργασίας, όπως αναφέρει και ο τίτλος της, είναι η ανάπτυξη μιας εφαρμογής "*task-manager*" για την παρακολούθηση της λειτουργίας του συστήματος και των εφαρμογών για λειτουργικό σύστημα Linux (ή GNU/Linux πιο σωστά) και το γραφικό περιβάλλον KDE. Η ονομασία "*task-manager*" παραπέμπει στην πιο δημοφιλή εφαρμογή του είδους, τον "*windows task manager*". Η ελληνική μετάφραση της φράσης αυτής είναι διαχειριστής εργασιών. Γενικότερα συναντάει κανείς παρόμοιες εφαρμογές με περιγραφές όπως "*system monitor*" που σημαίνει οθόνη ελέγχου συστήματος, "*process monitor*" που σημαίνει οθόνη ελέγχου διεργασιών, κτλ...

Το κύριο χαρακτηριστικό όλων αυτών των εφαρμογών είναι η εμφάνιση, συνήθως με την μορφή πίνακα, όλων των διεργασιών του συστήματος συνοδευόμενες από πληροφορίες που τις αφορούν όπως η χρήση μνήμης, ο κωδικός αριθμός, το όνομα του χρήστη που την εκκίνησε κ.α. Στις περισσότερες εξ' αυτών συνήθως παρατηρείται και η παρουσία πληροφοριών ή και γραφημάτων ιστορικού για την χρησιμοποίηση των πόρων του συστήματος συνολικά καθώς και δυνατότητες αλλαγής διάφορων παραμέτρων του συστήματος.

Στο λειτουργικό σύστημα GNU/Linux υπάρχει μια πληθώρα αντίστοιχων εφαρμογών γραφικών και μη, με πρώτη και καλύτερη την εφαρμογή "*top*" η οποία εκτελείται σε περιβάλλον γραμμής εντολών. Από τις δημοφιλέστερες και ισχυρότερες που φέρουν γραφικό περιβάλλον είναι η εφαρμογή "*ksysguard*" η οποία είναι και η επίσημη για το γραφικό περιβάλλον KDE, χωρίς όμως να αποκλείεται και η δυνατότητα εκτέλεσης σε οποιοδήποτε άλλο γραφικό περιβάλλον του GNU/Linux.

Ποιος ο λόγος όμως για την ανάπτυξη μιας εφαρμογής που αντίστοιχή της υπάρχει ήδη; Οι λόγοι είναι δυο. Δεν υπάρχει άλλη αντίστοιχη που να έχει αναπτυχθεί σε γλώσσα python και να φέρει εκτός των άλλων "κλασσικών" χαρακτηριστικών την δυνατότητα παρακολούθησης μέσω γραφημάτων των αναλυτικότερων

πληροφοριών των επιμέρους διεργασιών του συστήματος. Αυτό το χαρακτηριστικό είναι ιδιαίτερα σημαντικό καθώς μέσω αυτού μπορεί κανείς να μάθει επιπλέον πληροφορίες για κάθε διεργασία και να εντοπίσει πιθανές δυσλειτουργίες όπως η κακή χρήση της μνήμης ή της ΚΜΕ του συστήματος από μια διεργασία.

Στα κεφάλαια που θα ακολουθήσουν θα παρουσιαστεί το λειτουργικό σύστημα GNU/Linux και τα κυριότερα σημεία του που αφορούν την παρούσα εργασία και στην συνέχεια θα γίνει το ίδιο για την γλώσσα προγραμματισμού python και τα αρθρώματα της. Αργότερα θα παρουσιαστεί ο τρόπος ανάπτυξης και λειτουργίας “κάτω από το καπό” της εφαρμογής *Psymon* (το όνομα της εφαρμογής που αναπτύχθηκε από τα αρχικά Python System Monitor) και τέλος θα γίνει παρουσίαση του αποτελέσματος της ανάπτυξης όπου θα περιγραφεί η λειτουργία και πως μπορεί να χρησιμοποιηθεί στη πράξη η εφαρμογή καθώς και πληροφορίες για την άδεια χρήσης και το μοντέλο ανάπτυξης που χρησιμοποιήθηκε. Στην ουσία το προτελευταίο κεφάλαιο αποτελεί κατά κάποιο τρόπο και το εγχειρίδιο χρήσης της εφαρμογής στην ελληνική γλώσσα.

ΚΕΦΑΛΑΙΟ 1

Το Λειτουργικό Σύστημα GNU/Linux

Το GNU/Linux είναι ένα λειτουργικό σύστημα υπολογιστή αποτελούμενο από τον πυρήνα Linux και τις εφαρμογές του έργου GNU. Το έργο GNU χρονολογείται από τα τέλη του 1983 και είχε ως σκοπό την ανάπτυξη ενός λειτουργικού συστήματος το οποίο θα ήταν ελεύθερο (με την έννοια της *ελευθερίας του λόγου* και όχι της *δωρεάν μπύρας)* και ταυτόχρονα συμβατό με το τότε λειτουργικό σύστημα Unix. Ο εμπνευστής αυτής της προσπάθειας ήταν ο *Ριτσαρτν Στώλλμαν (Richard Matthew Stallman)*, γνωστός "χακερ" της εποχής και ακτιβιστής της ελευθερίας του λογισμικού μετέπειτα.

Το έτος 1991 και ενώ το έργο GNU έχει φτάσει σε σημείο να έχει αναπτύξει όλα τα βασικά κομμάτια ενός λειτουργικού συστήματος πλην όμως του πυρήνα, στην Φινλανδία ο τότε φοιτητής *Λάινους Τορβαλντς (Linus Torvalds)* ξεκινάει την ανάπτυξη ενός πυρήνα λειτουργικού συστήματος βασισμένο στο MINIX (λειτουργικό παρόμοιο του Unix το οποίο αναπτύχθηκε για εκπαιδευτικούς σκοπούς).

Εικόνα 1.1:
Η μασκότ του Linux, ο Tux

Στις αρχές του 1992 ο *Τορβαλντς* αφού έχει φτάσει το Linux σε σημείο να είναι πλήρως λειτουργικό και χωρίς δεσμούς με το MINIX, αποφασίζει να το διανέμει με την άδεια GPL η οποία είχε δημιουργηθεί με σκοπό να προστατέψει την ελευθερία του έργου GNU. Έτσι αμέσως έχουμε και την δημιουργία ενός νέου πλήρους λειτουργικού συστήματος, με τις εφαρμογές του έργου GNU και τον πυρήνα Linux, το GNU/Linux ή όπως έχει επικρατήσει για λόγους ευκολίας απλά Linux (γενικά όταν θέλουμε να αναφερθούμε σε όλο το Λ.Σ. συνηθίζεται να χρησιμοποιούμε απλά την λέξη "Linux", ενώ όταν θέλουμε να αναφερθούμε στον πυρήνα χρησιμοποιούμε την φράση "πυρήνας Linux"). Λόγο της ελευθερίας που δίνει η άδεια του λειτουργικού αυτού έχουμε και την δημιουργία των λεγόμενων "διανομών" οι οποίες

αναπτύσσονται είτε από απλούς εθελοντές είτε από εταιρίες. Οι διανομές είναι συλλογές εφαρμογών GNU και του πυρήνα Linux.

Η ανάπτυξη από εταιρίες και εθελοντές προγραμματιστές όλα αυτά τα χρόνια έχει φτάσει το GNU/Linux να είναι ένα λειτουργικό το οποίο βλέπουμε να λειτουργεί πίσω από πλειάδα ηλεκτρονικών συσκευών όπως ψυγεία, τηλεοράσεις και κινητά τηλέφωνα. Στους ηλεκτρονικούς υπολογιστές οι οποίοι είναι και το κομμάτι στο οποίο έχει δοθεί και η μεγαλύτερη προσοχή βλέπουμε το GNU/Linux να κυριαρχεί στους εξυπηρετητές, κυρίως ιστού, αλλά και στα υπερ-υπολογιστικά συστήματα. Στους υπολογιστές γραφείου η ανάπτυξη άρχισε με αργά βήματα, αλλά πλέον η ταχύτητα εξέλιξης και αποδοχής αυξάνεται με γεωμετρική πρόοδο. Εδώ, σημαντικό κομμάτι αποτελεί το γραφικό περιβάλλον του λειτουργικού το οποίο δεν είναι μόνο ένα αλλά περισσότερα από πέντε. Τα δημοφιλέστερα εξ' αυτών είναι το Gnome και το KDE με το οποίο θα ασχοληθούμε και παρακάτω.

1.1 Πόροι συστήματος και διαχείριση

Οι πόροι του συστήματος περιλαμβάνουν το υλικό του υπολογιστή και κυρίως τα πιο βασικά του στοιχεία όπως επεξεργαστές, μνήμη, δίσκους (ή άλλες αποθηκευτικές μονάδες), διαύλους PCI, USB και δίκτυο. Για την κάθε ομάδα πόρων το Linux παρέχει αρκετούς τρόπους πληροφόρησης αλλά και επέμβασης για την βελτιστοποίηση της απόδοσης του συστήματος.

Οι κεντρικοί επεξεργαστές είναι το κύριο χαρακτηριστικό ενός υπολογιστή. Οι περισσότεροι υπολογιστές έχουν έναν κεντρικό επεξεργαστή αν και έχουν αρχίσει τελευταία να γίνονται πιο διαδεδομένοι και οι υπολογιστές με περισσότερους από έναν. Το Linux μπορεί να τους χρησιμοποιήσει όλους και να εκμεταλλευτεί τον καθένα ξεχωριστά, π.χ. διαμοιράζοντας τα προγράμματα να τρέχουν σε πολλούς επεξεργαστές, ώστε να μην έχουμε πτώση της απόδοσης όταν τρέχουμε πολλά προγράμματα. Ο φόρτος του επεξεργαστή, δηλαδή το ποσοστό λειτουργίας του,

έχει συνδεθεί με κάποιους μετρητές. Οι μετρητές αυτοί δείχνουν την τιμή μηδέν στην περίπτωση που δεν γίνεται χρήση του επεξεργαστή και αυξάνονται κατα ένα κάθε φορά που μια διεργασία απασχολεί τον επεξεργαστή. Εδώ πρέπει να σημειωθεί ότι σε κάθε στιγμή στο σύστημα τρέχουν πολλές δεκάδες διεργασίες. Τις περισσότερες φορές όμως οι διεργασίες αυτές είναι σε κατάσταση αναμονής, περιμένοντας κάποιο γεγονός (event) να συμβεί, οπότε ξυπνούν και αρχίζουν την επεξεργασία του. Τότε θεωρούμε ότι ο επεξεργαστής έχει φόρτο από τις συγκεκριμένες διεργασίες. Όσο περισσότερες διεργασίες προσπαθούν να δεσμεύσουν τον επεξεργαστή τόσο υψηλότερο φόρτο έχουμε.

Η μνήμη του συστήματος είναι επίσης ένα βασικό στοιχείο του υπολογιστή. Θα πρέπει να υπάρχει σε αφθονία (με κάποιο μέτρο φυσικά) ώστε ο υπολογιστής να μη χρειαστεί να χρησιμοποιήσει το αρχείο εναλλαγής μνήμης (swap file), καθώς η χρήση του μειώνει αισθητά την απόδοση του υπολογιστή. Κάτι τέτοιο ίσως δεν είναι σημαντικό για οικιακή χρήση, αλλά για επαγγελματικά περιβάλλοντα είναι σημαντικός παράγοντας καθυστέρησης.

Η διαχείριση των σκληρών δίσκων σε ένα σύστημα είναι επίσης σημαντική υπόθεση. Είναι σημαντικό να βεβαιωθούμε ότι ένα διαμέρισμα έχει αρκετό διαθέσιμο χώρο, ώστε το σύστημα να συνεχίσει να λειτουργεί χωρίς πρόβλημα. Κάτι τέτοιο είναι ιδιαίτερα σημαντικό για το διαμέρισμα που φιλοξενεί τον βασικό κατάλογο root, τον κατάλογο /var και τον κατάλογο /tmp καθώς τα περιεχόμενα αυτών των καταλόγων μεταβάλλονται συχνά.

Το UNIX χρησιμοποιείται αυτή τη στιγμή ως η σπονδυλική στήλη (backbone) ολόκληρου του Internet, καθώς όλες σχεδόν οι ζωτικής σημασίας υπηρεσίες παρέχονται από τέτοια συστήματα. Στην πραγματικότητα το Internet αναπτύχθηκε για την επικοινωνία και ανταλλαγή δεδομένων μεταξύ συστημάτων UNIX. Είναι επόμενο να περιμένουμε ότι ένα σύστημα UNIX και κατά συνέπεια και το Linux, θα παρέχει μεγάλη ευελιξία στην διαχείριση των παραμέτρων ενός δικτύου. Το Linux, ως πυρήνας, έχει τη δυνατότητα υποστήριξης πολλών πρωτοκόλλων επικοινωνίας, πέρα από το βασικό TCP/IP (και τη νέα υλοποίηση με IPv6) που

χρησιμοποιείται στο Internet. Ανάμεσα σε άλλα, υποστηρίζει IPX, Appletalk, DECnet και X.25. Αυτό πρακτικά σημαίνει ότι το Linux μπορεί να επικοινωνήσει ακόμη και με τα πιο εξωτικά και εξειδικευμένα συστήματα, όπως δρομολογητές (routers), συσκευές RIP, συστήματα VAX, κλπ.

1.2 Διεργασίες

Έχει ήδη χρησιμοποιηθεί ο όρος “διεργασία” αρκετές φορές, χωρίς όμως να έχει τεκμηριώθεί ικανοποιητικά. Τι ακριβώς είναι οι διεργασίες και πως τις αντιλαμβάνεται το λειτουργικό σύστημα και ο επεξεργαστής;

Η εξέλιξη των υπολογιστών, η εμφάνιση γρήγορων επεξεργαστών και οι αυξημένες απαιτήσεις οδήγησαν στην εμφάνιση πολυδιεργασικών λειτουργικών συστημάτων (multitasking operating systems) τα οποία επέτρεπαν την παράλληλη εκτέλεση πολλών προγραμμάτων. Αυτά τα προγράμματα που εκτελούνται παράλληλα σε ένα τέτοιο λειτουργικό σύστημα αποτελούν τις διεργασίες. Στην πραγματικότητα δεν τρέχουν πραγματικά παράλληλα, αλλά το λειτουργικό σύστημα μοιράζει το χρόνο του επεξεργαστή έτσι που κάθε στιγμή τρέχει μόνο μια διεργασία, και για εκείνη τη στιγμή έχει σχεδόν τον πλήρη έλεγχο του επεξεργαστή.

Το λειτουργικό σύστημα πραγματοποιεί μερικές εκατοντάδες ως χιλιάδες εναλλαγές το δευτερόλεπτο, έτσι που στο χρήστη φαίνεται ότι εκτελούνται παράλληλα. Ακόμη και έτσι όμως, ο χρόνος που είναι διαθέσιμος από τον επεξεργαστή εξαρτάται από το φόρτο του, ή για την ακρίβεια από την απαίτηση της κάθε διεργασίας για επεξεργαστική ισχύ. Αυτό έχει ως συνέπεια τη φαινομενική καθυστέρηση του συστήματος όταν τρέχουμε πολλά προγράμματα. Φαινομενική, γιατί το σύστημα δεν είναι πιο αργό ούτε ο επεξεργαστής επεξεργάζεται λιγότερη πληροφορία. Απλώς ο χρόνος του μοιράζεται σε περισσότερες διεργασίες και αυτό δίνει την εικόνα σε κάθε διεργασία ότι ο υπολογιστής είναι αργός.

Σε όλα τα πολυδιεργασικά λειτουργικά συστήματα, όπως τα συστήματα UNIX και Linux, την όλη διαχείριση των διεργασιών την αναλαμβάνει ο πυρήνας. Για το σκοπό αυτό, κρατά μια λίστα διεργασιών, από την οποία λαμβάνει με τη σειρά κάθε διεργασία και την εκτελεί για ένα χρονικό διάστημα (συνήθως κάποια χιλιοστά του δευτερολέπτου). Στη συνέχεια θέτει την διεργασία αυτή σε νάρκη (sleep mode) και εκτελεί την επόμενη, κ.ο.κ.

Οι ίδιες οι διεργασίες δεν έχουν έλεγχο για πόσο χρόνο θα δεσμεύσουν από τον επεξεργαστή και πότε. Παλαιότερα, κάθε διεργασία είχε ίση μεταχείριση από τον πυρήνα, δηλαδή δέσμευε τον ίδιο χρόνο με τις άλλες διεργασίες. Κάτι τέτοιο όμως δεν είναι ιδιαίτερα αποδοτικό, αφού συχνά κάποιες διεργασίες βρίσκονται σε κατάσταση αναμονής (wait mode) ενώ κάποιες άλλες χρειάζονται όσο το δυνατόν περισσότερη επεξεργαστική ισχύ (π.χ. κάποιο πρόγραμμα rendering). Αντίστροφα, σε ορισμένες διεργασίες ο χρόνος απόκρισης είναι ιδιαίτερα σημαντικός (π.χ. η διεργασία που χειρίζεται το πληκτρολόγιο ή τους δίσκους) ενώ σε άλλες δεν παίζει ρόλο. Για το σκοπό αυτό ορίστηκε ένα σύστημα προτεραιοτήτων που βοηθά τον πυρήνα στην επιλογή των σημαντικότερων διεργασιών για εκτέλεση. Με βάση αυτό εκτελούνται πρώτα οι διεργασίες με υψηλή προτεραιότητα ενώ οι υπόλοιπες διεργασίες λαμβάνουν τον εναπομένοντα χρόνο του επεξεργαστή. Μάλιστα, το Linux χρησιμοποιεί ένα υβρίδιο αυτού του συστήματος, όπου οι προτεραιότητες ορίζονται δυναμικά από το σύστημα με βάση ένα ιστορικό χρήσης των ίδιων των διεργασιών. Το σύστημα αυτό είναι πολύ αποδοτικό και κάνει καλή χρήση του χρόνου του επεξεργαστή, χωρίς σπατάλη.

Αφού έγινε μια εξήγηση για τις διεργασίες στο Linux και γενικότερα, στο UNIX, παρακάτω θα παρουσιαστεί ο τρόπος διαχείρισής τους και τα είδη διεργασιών που υπάρχουν. Γενικά, όσον αφορά τον πυρήνα, όλες οι διεργασίες είναι ίδιες. Το μόνο που επηρεάζει τον πυρήνα για την δέσμευση χρόνου για κάθε διεργασία είναι η προτεραιότητά της. Υπάρχουν όμως ορισμένα χαρακτηριστικά των διεργασιών που τις κατηγοριοποιούν, κυρίως για λόγους διαχείρισης.

Κάθε διεργασία που έχει κάποιου είδους άμεση αλληλεπίδραση με το χρήστη λέγεται διαδραστική. Οι μη διαδραστικές διεργασίες μπορεί να είναι προγράμματα που τρέχουν συνεχώς και αναλαμβάνουν κάποια υπηρεσία, όπως οι διακομιστές ιστοσελίδων, τα οποία καλούνται και δαίμονες. Κάθε διεργασία μπορεί να ξεκινήσει μια άλλη διεργασία. Οι δύο διεργασίες ονομάζονται αντίστοιχα γονική και θυγατρική (parent και child process). Όλες οι διεργασίες στο Linux έχουν μια τέτοια σχέση καθώς δημιουργούνται από την init, που είναι η γονική διεργασία όλων των υπολοίπων. Αν η θυγατρική διεργασία είναι αντίγραφο της γονικής, τότε η διαδικασία λέγεται διακλάδωση διεργασιών (process forking). Σε κάθε διεργασία αντιστοιχεί και ένας αριθμός ταυτότητας (process id) που πιο συχνά θα το δούμε ως PID. Σε κάθε στιγμή ο αριθμός αυτός είναι μοναδικός για την κάθε διεργασία. Ένας χρήστης δε μπορεί να τερματίσει ή να επηρεάσει μια διεργασία κάποιου άλλου χρήστη ούτε να της αλλάξει την προτεραιότητα, εκτός αν είναι ο χρήστης root.

1.3 Το σύστημα αρχείων /proc

Το /proc είναι ένα εικονικό σύστημα αρχείων το οποίο μας δίνει τη δυνατότητα να πάρουμε πληροφορίες από τις δομές δεδομένων του πυρήνα. Εδώ βρίσκονται τα στοιχεία που κρατά εσωτερικά ο πυρήνας για την κάθε διεργασία. Είναι εικονικό με την έννοια ότι τα αρχεία που βλέπουμε δεν έχουν κάποια φυσική υπόσταση (πχ δεν βρίσκονται σε κάποια συσκευή). Τα περισσότερα αρχεία μπορούν να ανοιχτούν μόνο για ανάγνωση.

Στο βασικό κατάλογο /proc υπάρχει ένα πλήθος από αρχεία και καταλόγους. Κάποια από αυτά περιέχουν ολόκληρες δομές πληροφοριών, ενώ άλλα απλώς την τιμή μιας συγκεκριμένης μεταβλητής του πυρήνα. Τα περισσότερα αρχεία έχουν ονόματα που αυτοεξηγούνται.

Κάποια κύρια είναι:

- **/proc/<num> (directory)** : Οι κατάλογοι που το όνομα τους είναι ένας αριθμός και περιέχουν πληροφορίες για τη διεργασία με τον συγκεκριμένο κωδικό διεργασίας (pid). Θα τους εξετάσουμε αναλυτικά σε λίγο.
- **/proc/self (directory)** : Συμβολικός σύνδεσμος στον κατάλογο που περιέχει πληροφορίες για την τρέχουσα διεργασία
- **/proc/ide (directory)** : Πληροφορίες για τις συσκευές που είναι συνδεδεμένες στο ide bus.
- **/proc/sys (directory)** : Πλήθος πληροφοριών για το σύστημα και τον πυρήνα.
- **/proc/cpuinfo** : Πληροφορίες για τον επεξεργαστή,
- **/proc/filesystems** : Λίστα από τα συστήματα αρχείων που υποστηρίζει ο πυρήνας.
- **/proc/meminfo** : Πληροφορίες για τη χρήση της μνήμης στο σύστημα.
- **/proc/modules** : Ποια αρθρώματα (modules) είναι φορτωμένα στον πυρήνα.
- **/proc/interrupts** : Από ποιες συσκευές πηγάζουν οι “διακοπές” υλικού.
- **/proc/partitions** : Λίστα με όλες τις κατατμήσεις στον υπολογιστή (άσχετα αν είναι προσαρτημένες ή όχι).
- **/proc/mounts** : Λίστα με όλες τις προσαρτήσεις.

Αν και το /proc περιέχει τεράστιο όγκο πληροφοριών, αυτές που είναι πραγματικά χρήσιμες είναι οι πληροφορίες περί διεργασιών. Αυτές βρίσκονται στον κατάλογο με αριθμό (όνομα) το pid της διεργασίας.

Τα κύρια περιεχόμενα του καταλόγου είναι:

- **/cwd (directory link)**: link στον τρέχον κατάλογο της διεργασίας (current working directory).
- **/fd (directory)** : Περιέχει links στα ανοικτά αρχεία της διεργασίας. Πχ το link για το αρχείο με file descriptor 2 είναι fd/2.
- **/root (directory link)**: link στο root directory του filesystem στο οποίο εκτελείται η διεργασία (συνήθως "/").
- **cmdline** : Η πλήρης γραμμή εντολής με την οποία κλήθηκε η διεργασία.
- **environ** : Λίστα με τις μεταβλητές περιβάλλοντος που βλέπει η διεργασία.
- **exe (link)** : Σύνδεσμος στο εκτελέσιμο από το οποίο δημιουργήθηκε η διεργασία.
- **maps** : Ο χάρτης μνήμης της διεργασίας.
- **mem** : Η ίδια η μνήμη της διεργασίας. Διαβάζοντας και γράφοντας στο αρχείο αυτό, προσπελαύνουμε κατευθείαν τη μνήμη της διεργασίας. Προφανώς πρέπει να έχουμε τα κατάλληλα δικαιώματα.
- **mounts** : Λίστα με όλες τις προσαρτήσεις.
- **stat** : Η κατάσταση της διεργασίας.
- **statm** : Πληροφορίες για την κατάσταση της μνήμης.
- **status** : Πληροφορίες από τα δύο παραπάνω αρχεία σε πιο ευανάγνωστη μορφή.

1.4 Γραφικό περιβάλλον KDE

Το γραφικό περιβάλλον εργασίας (desktop environment) KDE είναι ένα από τα πιο σύγχρονα και δημοφιλή περιβάλλοντα εργασίας στο λειτουργικό σύστημα GNU/Linux. Εκτός του GNU/Linux, μπορεί να εκτελεστεί και στα περισσότερα συστήματα Unix και Unix-συμβατά, όπως το BSD, AIX, Unixware, OpenServer και Solaris. Επίσης έχει μεταφερθεί στο Mac OS X όπου τρέχει πάνω από τον X11 και στα Microsoft Windows χρησιμοποιώντας το Cygwin.

Το KDE θεμελιώθηκε το 1996 από τον *Ματίας Έτριχ*, φοιτητή του Πανεπιστημίου του *Τύμπινγκεν*. Ο *Ματίας* απεφάσισε να χρησιμοποιήσει το Q Toolkit για το έργο KDE το οποίο αναπτυσσόταν από την εταιρία Trolltech και πλέον την Nokia.

Εικόνα 1.2: Επιφάνεια εργασίας KDE

Το Q Toolkit (Qt) είναι μία εργαλειοθήκη για την δημιουργία εφαρμογών με γραφική διεπαφή χρήστη (GUI) και η οποία προσφέρει τα εξής:

- Κλάσεις: Η βάση της Qt αποτελείται από 400+ κλάσεις για containers, δημιουργία και σχεδίαση GUI, βάσεις δεδομένων, δικτύωση, υποστήριξη XML και πολυγλωσσικότητας, OpenGL, κ.α.

- Qt Designer: Μια εφαρμογή οπτικής δημιουργίας GUI μέσω του ποντικιού, που επιτρέπει την γρήγορη ανάπτυξη γραφικών περιβαλλόντων που είναι ίδια σε όλες τις πλατφόρμες.

- Qt Linguist: Ένα σύνολο εργαλείων για εύκολη υποστήριξη πολλών γλωσσών, που διευκολύνει την μετάφραση των μηνυμάτων και των μενού κάθε εφαρμογής Qt.

- Qt Assistant: Μια εφαρμογή εύρεσης και προβολής της τεκμηρίωσης της Qt, που μπορεί να ενσωματώνεται με αλλαγές μέσα στις εφαρμογές Qt.

Το έργο KDE έχει δημιουργήσει τις δικές του βιβλιοθήκες βασισμένες στην Qt για την εύκολη ανάπτυξη και ομογενοποίηση των εφαρμογών που γράφονται για αυτό. Συνηθίζεται μάλιστα να χρησιμοποιείται το αρχικό λατινικό γράμμα “K” στην αρχή του ονόματος κάθε εφαρμογής που έχει γραφτεί με τις βιβλιοθήκες του KDE. Παρόλα αυτά αν μια εφαρμογή έχει γραφτεί καθαρά με τις πρότυπες βιβλιοθήκες της Qt, η διαδικασία ενσωμάτωσης στο έργο KDE είναι πολύ απλή υπόθεση αν το επιθυμούμε.

Επίλογος

Στο παρόν κεφάλαιο παρουσιάστηκαν για το λειτουργικό σύστημα GNU/Linux όλα τα στοιχεία που θα χρησιμεύσουν στην ανάγνωση των επόμενων κεφαλαίων που θα αφορούν το κύριο κομμάτι της ανάπτυξης της εφαρμογής. Αρχίζοντας από μια μικρή ιστορική αναδρομή και μια γενικότερη αναφορά έγινε ένα πέρασμα στα πιο συγκεκριμένα κομμάτια που αφορούν την εργασία και τα οποία είναι οι χρήση των πόρων, οι διεργασίες και τέλος το γραφικό περιβάλλον εργασίας KDE. Στο επόμενο κεφάλαιο αντίστοιχα, περιγράφεται η γλώσσα προγραμματισμού python

ώστε να γίνει η κατάλληλη προετοιμασία για την ανάγνωση του 3ου κεφαλαίου όπου θα γίνει η ανάλυση της ανάπτυξης της εφαρμογής πάνω στο λειτουργικό σύστημα GNU/Linux, με χρήση της python και της qt.

ΚΕΦΑΛΑΙΟ 2

Η γλώσσα προγραμματισμού Python

Η Python είναι μια από εκείνες τις σπάνιες γλώσσες που ισχυρίζονται ότι είναι και απλές και ισχυρές. δημιουργήθηκε από τον Ολλανδό Γκουίντο βαν Ρόσσουμ (Guido van Rossum) το 1990. Ο κύριος στόχος της είναι η αναγνωσιμότητα του κώδικά της και η ευκολία χρήσης της. Διακρίνεται λόγω του ότι έχει πολλές βιβλιοθήκες που διευκολύνουν ιδιαίτερα αρκετές συνηθισμένες εργασίες και για την ταχύτητα εκμάθησης της.

Εικόνα 2.1: Το λογότυπο της python

Η Python αναπτύσσεται ως λογισμικό ανοιχτού κώδικα και η διαχείρισή της γίνεται από τον μη κερδοσκοπικό οργανισμό Python Software Foundation. Ο κώδικας διανέμεται με την άδεια Python Software Foundation License η οποία είναι συμβατή με την GPL. Το όνομα της γλώσσας προέρχεται από την ομάδα άγγλων κωμικών Μόντυ Πάιθον. Αρχικά, η Python ήταν γλώσσα σεναρίων που χρησιμοποιούνταν στο λειτουργικό σύστημα Amoeba, ικανή και για κλήσεις συστήματος. Η Python 2.0 κυκλοφόρησε στις 16 Οκτωβρίου του 2000, ενώ στις 3 Δεκεμβρίου 2008 κυκλοφόρησε η έκδοση 3.0.

Η γλώσσα χρησιμοποιεί μεταγλωττιστή (compiler) για την δημιουργία του εκτελέσιμου κώδικα και σχετίζεται με τις γλώσσες προγραμματισμού Tcl, Perl, Scheme, Java και Ruby, καθώς και με την ABC η οποία υπήρξε η αρχική πηγή έμπνευσης για τη δημιουργία της.

Ένα από τα πιο απλά προγράμματα στην γλώσσα Python είναι η εμφάνιση ενός γραπτού αποτελέσματος (π.χ. Γεια σου, κόσμε!):

```
>>>print("Γεια σου, κόσμε!")

Γεια σου, κόσμε!
```

2.1 Χαρακτηριστικά

Απλή

Η Python είναι μια απλή και μινιμαλιστική γλώσσα. Το διάβασμα ενός καλού προγράμματος σε Python είναι σαν το διάβασμα των Αγγλικών, αλλά πολύ αυστηρών Αγγλικών! Αυτή η ομοιότητα της Python με ψευδοκώδικα είναι ένα από τα πιο ισχυρά σημεία της. Επιτρέπει τον προγραμματιστή να συγκεντρώνεστε στη λύση του προβλήματος αντί στην ίδια τη γλώσσα.

Εύκολη στην εκμάθηση

Είναι εξαιρετικά απλό να ξεκινήσετε με την Python. Η Python έχει μια ασυνήθιστα απλή σύνταξη, όπως έχει ήδη αναφερθεί.

Ελεύθερη και Ανοικτού Κώδικα

Η Python είναι ένα παράδειγμα *ΕΛΛΑΚ* (Ελεύθερο Λογισμικό και Λογισμικό Ανοικτού Κώδικα). Με απλά λόγια, μπορεί κανείς να διανείμει αντίγραφα αυτού του λογισμικού, να διαβάσει τον πηγαίο κώδικά του, να κάνει αλλαγές σ' αυτό και να χρησιμοποιήσει κομμάτια του σε νέα ελεύθερα προγράμματα. Το ΕΛΛΑΚ βασίζεται στην ιδέα μιας κοινότητας που μοιράζεται τη γνώση. Αυτός είναι ένας από τους λόγους για τους οποίους η Python είναι τόσο καλή. Δημιουργήθηκε και βελτιώνεται συνεχώς από μια κοινότητα που το μόνο που θέλει είναι μια καλύτερη Python.

Γλώσσα υψηλού επιπέδου

Κατά την συγγραφή προγραμμάτων στην Python, δε χρειάζεται ποτέ να ασχοληθεί κανείς για τις χαμηλού επιπέδου λεπτομέρειες όπως η διαχείριση της μνήμης που χρησιμοποιείται από τα προγράμματά σας, κ.λπ.

Φορητή

Λόγω του ανοικτού της κώδικα, η Python έχει υλοποιηθεί (δηλαδή αλλάχτηκε για να λειτουργεί) σε πολλές πλατφόρμες. Όλα τα Python προγράμματά μπορούν να

δουλέψουν σε οποιαδήποτε πλατφόρμα χωρίς να χρειάζονται καθόλου αλλαγές αν ο προγραμματιστής είναι αρκετά προσεκτικός ώστε να αποφύγει να χρησιμοποιήσει χαρακτηριστικά που εξαρτιούνται από κάθε σύστημα.

Η Python μπορεί να χρησιμοποιηθεί στο GNU/Linux, στα Windows, στο FreeBSD, σε Macintosh, στο Solaris, στο OS/2, στην Amiga, στο AROS, στο AS/400, στο BeOS, στο OS/390, στο z/OS, στο Palm OS, στο QNX, στο VMS, στο Psion, στο Acorn RISC OS, στο VxWorks, σε PlayStation, στο Sharp Zaurus, στα Windows CE ακόμα και σε PocketPC!

Διερμηνευόμενη

Εδώ χρειάζονται μερικές εξηγήσεις... Ένα πρόγραμμα που γράφεται σε μια μεταγλωττιζόμενη γλώσσα όπως η C ή η C++ μετατρέπεται από την πηγαία γλώσσα, για παράδειγμα τη C ή τη C++ σε μια γλώσσα που μιλάει ο υπολογιστής σας (δυαδικός κώδικας δηλαδή 0 και 1) χρησιμοποιώντας ένα μεταγλωττιστή με διάφορες σημαίες και επιλογές. Όταν τρέχετε το πρόγραμμα, ο φορτωτής (loader) αντιγράφει το πρόγραμμα στη μνήμη και αρχίζει να το τρέχει.

Η Python, από την άλλη, δε χρειάζεται μεταγλώττιση σε δυαδικό αρχείο απο τον προγραμματιστή. Ο προγραμματιστής απλά *τρέχει* το πρόγραμμα απ' ευθείας από τον πηγαίο κώδικα. Εσωτερικά, η Python μετατρέπει τον πηγαίο κώδικα σε μια ενδιάμεση μορφή που ονομάζεται bytecode και μετά το μεταφράζει στη γλώσσα του υπολογιστή και μετά το τρέχει. Όλο αυτό, στην πραγματικότητα κάνει τη χρήση της Python πολύ πιο εύκολη αφού δε χρειάζεται να ανησυχείτε για τη μεταγλώττιση του προγράμματος, τη σύνδεση με τις κατάλληλες βιβλιοθήκες, κλπ. Αυτό επίσης κάνει τα προγράμματα της Python εξαιρετικά φορητά, αφού μπορείτε απλά να αντιγράψετε το πρόγραμμα Python που φτιάξατε σε έναν άλλο υπολογιστή και να δουλέψει έτσι απλά!

Αντικειμενοστρεφής

Η Python υποστηρίζει τόσο το διαδικασιοστρεφή προγραμματισμό (procedure-oriented) όσο και τον αντικειμενοστρεφή προγραμματισμό (object-oriented). Στο *διαδικασιοστρεφή προγραμματισμό*, το πρόγραμμα δομείται πάνω σε διαδικασίες ή συναρτήσεις οι οποίες δεν είναι τίποτε άλλο από επαναχρησιμοποιήσιμα κομμάτια από προγράμματα. Στις *αντικειμενοστρεφείς* γλώσσες, τα προγράμματα δομούνται πάνω σε αντικείμενα τα οποία συνδυάζουν δεδομένα και λειτουργικότητα. Η Python έχει έναν πολύ ισχυρό αλλά πολύ απλό τρόπο για αντικειμενοστρεφή προγραμματισμό, ειδικά όταν συγκρίνεται με μεγάλες γλώσσες όπως η C++ ή η Java.

Επεκτάσιμη

Αν ο προγραμματιστής χρειάζεται ένα κρίσιμο κομμάτι κώδικα να τρέχει πολύ γρήγορα ή αν πρέπει να έχει ένα κομμάτι ενός αλγόριθμου που να μην είναι ανοικτό, τότε μπορεί να προγραμματίσει εκείνο το κομμάτι σε C ή C++ και μετά να το χρησιμοποιήσει από το Python πρόγραμμά του.

Ενσωματώσιμη

Η Python μπορεί να ενσωματωθεί μέσα στα προγράμματα σε C/C++ για να τους δώσει δυνατότητες 'scripting' για τους χρήστες.

Εκτεταμένες βιβλιοθήκες

Η πρότυπη βιβλιοθήκη της Python είναι πραγματικά τεράστια. Μπορεί να βοηθήσει τον προγραμματιστή να κάνει διάφορα πράγματα σχετικά με κανονικές εκφράσεις, δημιουργία τεκμηρίωσης, δοκιμές μονάδων, νημάτωση, βάσεις δεδομένων, περιηγητές ιστού, CGI, FTP, email, XML, XML-RPC, HTML, αρχεία WAV, κρυπτογράφηση, γραφικές διεπαφές χρήστη (GUI -graphical user interfaces), Tk, και άλλα πράγματα που εξαρτιούνται από το σύστημα. Όλα αυτά είναι διαθέσιμα όποτε είναι εγκατεστημένη η Python. Αυτό ονομάζεται φιλοσοφία 'Batteries Included' της Python.

2.2 Αρθρώματα

Τα αρθρώματα (modules) στην python είναι είτε αυτοτελή κομμάτια είτε μέρη βιβλιοθηκών τα οποία περιέχουν έτοιμες συναρτήσεις τις οποίες ο προγραμματιστής μπορεί να χρησιμοποιήσει στα προγράμματα που γράφει.

Για να μπορέσει να χρησιμοποιηθεί ένα άρθρωμα αρκεί πρώτα να εισαχθεί πριν την κλίση κάποιας μεθόδου του με την εντολή "`import`" ή "`from <όνομα αρθρώματος> import <μεταβλητή/ες>`".

Τα αρθρώματα συνήθως τοποθετούνται μέσα σε φακέλους μαζί με ένα ειδικό αρχείο "__init__.py" και οι οποίοι ονομάζονται πακέτα. Τα πακέτα πρέπει να τοποθετούνται στον ίδιο κατάλογο με το πρόγραμμα που τα εισάγει, ή να βρίσκονται σε έναν από τους καταλόγους που είναι στη λίστα της μεταβλητής περιβάλλοντος της python, την "PYTHONPATH".

2.3 Η βιβλιοθήκη αρθρωμάτων pyqt

Στο υποκεφάλαιο (1.4) περιγράφηκε η βιβλιοθήκη γραφικών Qt. Η βιβλιοθήκη αυτή είναι λογισμικό ανοιχτού κώδικα, διαπλατφορμική και έχει δημιουργηθεί σε C++ έχοντας ως κύριο σκοπό την χρήση της από την γλώσσα αυτή. Παρόλα αυτά μπορεί να χρησιμοποιηθεί από αρκετές επιπλέον γλώσσες προγραμματισμού μέσω των "συνδέσεων γλώσσας". Εκτός από τα χαρακτηριστικά γραφικής διεπαφής διαθέτει και άλλα επιπλέον όπως η πρόσβαση σε SQL βάσεις δεδομένων, ανάλυση XML, διαχείριση νημάτων, υποστήριξη δικτύωσης καθώς και ένα διαπλατφορικό API για τον χειρισμό αρχείων.

Η pyqt είναι μια βιβλιοθήκη αρθρωμάτων που παρέχει σύνδεση μεταξύ της python και της Qt. Τα περιεχόμενά της είναι τα εξής:

- Το άρθρωμα *QtCore* περιέχει τον πυρήνα των μη γραφικής-διεπαφής-χρήστη (graphical user interface) κλάσεων, συμπεριλαμβανομένου του

“event loop” και του μηχανισμού “signal and slot”. Επίσης περιλαμβάνει ανεξάρτητες από την εκάστοτε πλατφόρμα υλοποιήσεις για το διεθνές πρότυπο Unicode, νήματα, χαρτογραφημένα αρχεία, κοινόχρηστη μνήμη, κανονικές εκφράσεις καθώς και ρυθμίσεις χρήστη και εφαρμογής.

- Το άρθρωμα *QtGui* περιέχει την πλειοψηφία των κλάσεων γραφικής-διεπαφής-χρήστη. Αυτές περιέχουν έναν αριθμό από κλάσεις πινάκων, δέντρων και λιστών βασισμένων στο πρότυπο σχεδίασης “model-view-controller”. Επίσης παρέχεται ένας περίτεχνος 2Δ κανβάς “widget” ικανός να αποθηκεύσει χιλιάδες στοιχεία συνηθισμένων widget.

- Το άρθρωμα *QtNetwork* περιέχει τις κλάσεις για την ανάπτυξη UDP και TCP πελατών και εξυπηρετητών. Αυτές περιέχουν κλάσεις που υλοποιούν FTP και HTTP πελάτες και υποστηρίζουν αναζητήσεις DNS. Τα “event” Δικτύωσης είναι ενοποιημένα με τα “event loop” κάνοντας την ανάπτυξη δικτυακών εφαρμογών εύκολη υπόθεση.

- Το άρθρωμα *QtOpenGL* περιέχει κλάσεις που ενεργοποιούν τη χρήση του OpenGL για την απόδοση 3Δ γραφικών στις εφαρμογές pyqt.

- Το άρθρωμα *QtSql* περιέχει τις κλάσεις για την ενοποίηση με τις ανοιχτού κώδικα αλλά και τις ιδιοταγείς βάσεις δεδομένων. Αυτές περιέχουν επεξεργάσιμα μοντέλα δεδομένων για πίνακες βάσεων δεδομένων τα οποία μπορούν να χρησιμοποιηθούν με τις γραφικής-διεπαφής-χρήστη κλάσεις. Επίσης περιέχουν υλοποίηση της SQLite.

- Το άρθρωμα *QtSvg* περιέχει τις κλάσεις για την εμφάνιση περιεχομένων SVG αρχείων. Υποστηρίζει τα στατικά χαρακτηριστικά του SVG 1.2 Tiny.

- Το άρθρωμα *QtXml* υλοποιεί τις διεπαφές SAX και DOM σε Qt XML κατατμητή.

- Το άρθρωμα *QtMultimedia* υλοποιεί χαμηλού επιπέδου λειτουργία πολυμέσων. Οι προγραμματιστές συνήθως χρησιμοποιούν το άρθρωμα phonon.

- Το άρθρωμα *QtDesigner* περιέχει κλάσεις που επιτρέπουν τον Qt Designer να επεκταθεί χρησιμοποιώντας την pyqt.

- Το άρθρωμα *Qt* εδραιώνει τις κλάσεις που περιέχονται σε όλα τα αρθρώματα που περιγράφηκαν παραπάνω μέσα σε ένα και μόνο άρθρωμα. Η χρήση του έχει ως πλεονέκτημα το ότι δεν χρειάζεται να ανησυχούμε σε ποιο υπο-άρθρωμα περιέχεται μια συγκεκριμένη κλάση. Το μειονέκτημα είναι ότι φορτώνει όλο το πλαίσιο Qt συνεπάγοντας αύξηση της χρήσης της μνήμης από μια εφαρμογή.

- Το άρθρωμα *uic* υλοποιεί την υποστήριξη χειρισμού αρχείων XML τα οποία δημιουργούνται από τον Qt Designer και τα οποία περιγράφουν όλο το κομμάτι μιας γραφικής-διεπαφής-χρήστη. Περιέχει κλάσεις που φορτώνουν ένα αρχείο XML και το αποδίδουν απευθείας εμφανίζοντας τη γραφική διεπαφή χρήστη. Επίσης περιέχει κλάσεις που δημιουργούνε κώδικα python από ένα αρχείο XML για μετέπειτα εκτέλεση.

2.4 Το άρθρωμα psutil

Το άρθρωμα psutil είναι επίσης λογισμικό ανοιχτού κώδικα και παρέχει μια διεπαφή για την ανάκτηση πληροφοριών για όλες τις διεργασίες καθώς και τη χρήση του συστήματος (ΚΜΕ, δίσκων, μνήμης κτλ) με ένα φορητό τρόπο, υλοποιώντας πολλές λειτουργίες που παρέχονται από πολλά εργαλεία γραμμής εντολών όπως η ps, top, df, kill, free, uptime, lsof κ.α.

Αυτήν την στιγμή υποστηρίζει τα λειτουργικά συστήματα Linux, MS Windows, Mac OSX και FreeBSD στις 32bit αλλά και 64bit εκδόσεις τους. Το άρθρωμα μπορεί να

χρησιμοποιηθεί από την έκδοση 2.4 έως και την 3+ της python χρησιμοποιώντας μια και μόνο βάση κώδικα.

Η υποστήριξη σε Linux βασίζεται κυρίως στην ανάγνωση και χρήση των τιμών των περιεχομένων του συστήματος αρχείων /proc στο οποίο αναφερθήκαμε στο υποκεφάλαιο της παρούσας εργασίας (1.3). Για παράδειγμα η ανάκτηση της εναποθηκευμένης (cached) μνήμης γίνεται με την εξής μέθοδο της python:

```
def cached_phymem():
      """Return the amount of cached memory on the system, in bytes.
      This reflects the "cached" column of free command line utility.
      """
      f = open('/proc/meminfo', 'r')
      try:
            for line in f:
                  if line.startswith('Cached:'):
                        return int(line.split()[1]) * 1024
            raise RuntimeError("line not found")
      finally:
            f.close()
```

Η προγραμματιστική διεπαφή της εφαρμογής περιέχει τις εξαιρέσεις, τις κλάσεις, τις σχετιζόμενες με το σύστημα μεθόδους και τις σταθερές. Οι σχετιζόμενες με το σύστημα σταθερές χωρίζονται σε αυτές που αφορούν τις διεργασίες, την ΚΜΕ, την μνήμη, τους δίσκους και το δίκτυο. Στην διαδικασία ανάπτυξης της εφαρμογής της πτυχιακής εργασίας θα γίνει χρήση σχεδόν όλης της προγραμματιστικής διεπαφής του psutil!

Επίλογος

Η python είναι μια γλώσσα προγραμματισμού η οποία αξίζει προσοχής από κάθε προγραμματιστή είτε έμπειρου, είτε αρχάριου. Είτε φίλου του “scripting”, είτε του διαδικαστικού ή του αντικειμενοστραφή προγραμματισμού η python θα εντυπωσιάσει τον καθένα με την παραγωγικότητα και απλότητα που προσφέρει. Στο παρόν κεφάλαιο έγινε μια σύντομη, αλλά ταυτόχρονα πλήρους για τις μετέπειτα απαιτήσεις ανάγνωσης, περιγραφή της γλώσσας python και των αρθρωμάτων της. Στο επόμενο κεφάλαιο θα γίνει παρουσίαση της χρήση όλων όσων αναφέρθηκαν στα προηγούμενα κεφάλαια.

ΚΕΦΑΛΑΙΟ 3

Ανάπτυξη της εφαρμογής

Η διαδικασία της ανάπτυξης της εφαρμογής κράτησε σχεδόν τρεις μήνες. Αρχικά έγινε η μια καλύτερη κατανόηση των ζητούμενων της πτυχιακής και κατόπιν μια μελέτη διάφορων τεκμηριώσεων, δημοσιεύσεων και βιβλίων που αφορούσαν τα επιμέρους θέματα της εργασίας. Η μελέτη αυτή περιείχε εμβάθυνση στον τρόπο ανάκτησης των πληροφοριών αλλά και του τρόπου λειτουργίας του λειτουργικού συστήματος Linux μιας και ήδη υπήρχε μια κάποια εμπειρία, καθώς επίσης και εκμάθηση του τρόπου χρήσης της python και της Qt. Παράλληλα έγινε και μια αναζήτηση και μελέτη των ήδη υλοποιημένων παρόμοιων λύσεων όσο αναφορά το σύνολο της εφαρμογής αλλά και των επιμέρους κομματιών.

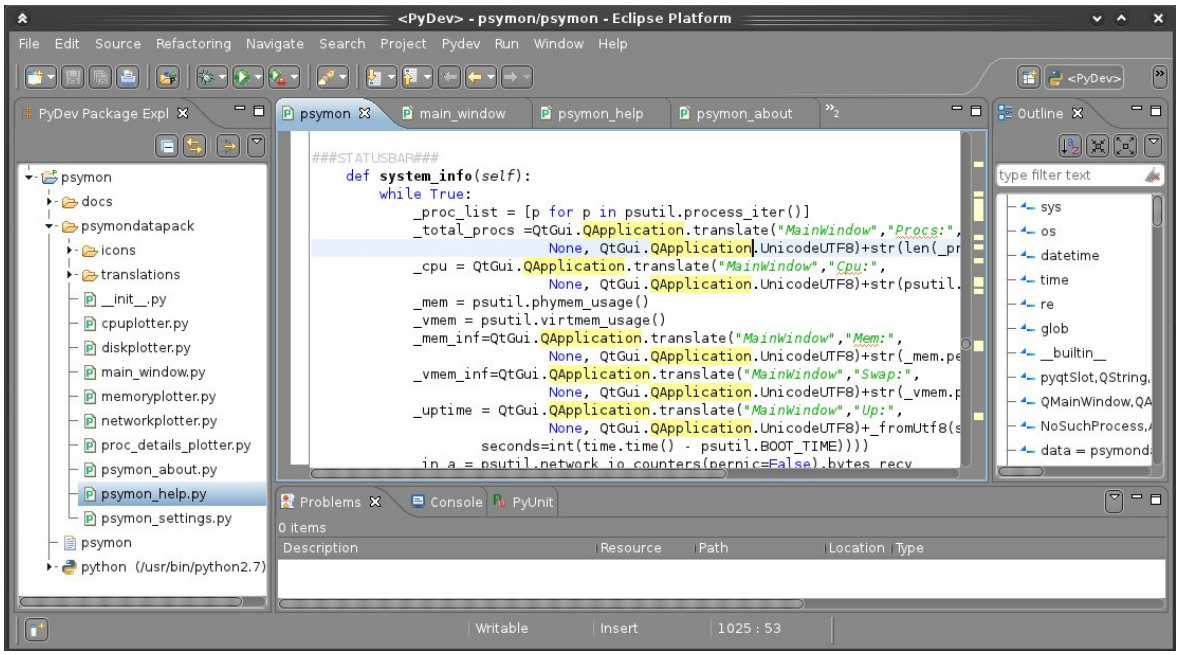
Η αναζήτηση για τις ήδη υλοποιημένες και ολοκληρωμένες ή μη λύσεις είχε ως αποτέλεσμα την εύρεση του αρθρώματος psutil το οποίο διευκόλυνε σε σημαντικότατο βαθμό την ανάπτυξη. Επίσης βασικός οδηγός στην ανάπτυξη υπήρξε η ήδη υπάρχουσα εφαρμογή του KDE, το Ksysguard.

3.1 Τα υλικά και τα εργαλεία της ανάπτυξης

Η ανάπτυξη της εφαρμογής έγινε σε λειτουργικό σύστημα Linux με γραφικό περιβάλλον KDE 4 και συγκεκριμένα στην διανομή Kubuntu. Τα προγράμματα που χρησιμοποιήθηκαν στην ανάπτυξη ήταν το IDE Eclipse με το πρόσθετο PyDev, η εφαρμογή σχεδίασης Qt Designer, η εφαρμογή μεταφράσεων Qt Linguist, ο κειμενογράφος Kate, η εφαρμογή επεξεργασίας εικόνων Gimp, η εφαρμογή σχεδίασης διαγραμμάτων Dia, η γραφική διεπαφή γραμμής εντολών Konsole καθώς και η εφαρμογή Virtualbox στην οποία έγιναν οι δοκιμές για το λειτουργικό σύστημα MS Windows. Τέλος σαν οδηγός όπως προαναφέρθηκε αλλά και ως εφαρμογή ελέγχου χρησιμοποίησης και σωστής λειτουργίας χρησιμοποιήθηκε η

εφαρμογή Ksysguard. Η συγγραφή της παρούσας αναφοράς έγινε στην εφαρμογή Writer της σουίτας εφαρμογών γραφείου LibreOffice.

Τα υλικά που χρησιμοποιήθηκαν στην ανάπτυξη ήταν η γλώσσα προγραμματισμού Python 2.7 μαζί με τις πρότυπες βιβλιοθήκες αρθρωμάτων της και όλα τα βασικά στοιχεία που την απαρτίζουν. Επίσης χρησιμοποιήθηκαν, η Qt και οι συνδέσεις της Python PyQt στην έκδοση 4.7. Το άρθρωμα της Python, Psutil στην έκδοση 0.4 και τέλος η πρόσθετη βιβλιοθήκη για τις ανάγκες ανάπτυξης των γραφημάτων της Qt και PyQt, Qwt και PyQwt αντίστοιχα, στην έκδοση 5.



Εικόνα 3.1: Το IDE Eclipse εν δράσει

Για το πακετάρισμα της εφαρμογής χρησιμοποιήθηκαν τα προγράμματα Ark που είναι εφαρμογή δημιουργίας συμπιεσμένων αρχείων, η εφαρμογή py2exe και InnoSetup την δημιουργία πακέτου εγκατάστασης και εκτελέσιμων αρχείων “exe” για τα λειτουργικά συστήματα MS Windows, η εφαρμογή dpkg-buildpackage για την δημιουργία πακέτου “deb” και τέλος η εφαρμογή alien για την μετατροπή του “deb” πακέτου σε “rpm”. Το πακέτο deb είναι συμβατό με όλες τις διανομές Linux οι οποίες είναι βασισμένες στην διανομή Debian. Το πακέτο rpm αφορά όλες τις διανομές που χρησιμοποιούν τον διαχειριστή πακέτων της διανομής RedHat.

3.2 Η δομή της εφαρμογής

Η δομή των αρχείων της εφαρμογής είναι σχετικά απλή. Ξεκινώντας από τα δημιουργηθέντα αρχεία, υπάρχει ένα κύριο αρχείο κώδικα που περιλαμβάνει τον κύριο κορμό της εφαρμογής καθώς και τον κώδικα αρχικοποίησης και εκκίνησης αυτής. Επίσης υπάρχει και ένα πακέτο python που περιέχει εκτός του αρχείου "__init__.py", άλλα εννέα αρχεία κώδικα που περιέχουν κυρίως τις κλάσεις και τις μεθόδους για τα γραφήματα και τα επιπλέον παράθυρα της εφαρμογής. Μέσα στο πακέτο υπάρχουν και δυο ακόμη φάκελοι. Ο ένας περιέχει τα εικονίδια που χρησιμοποιεί η εφαρμογή και ο άλλος τα αρχεία μετάφρασης.

Ως προς την δομή του κώδικα, σε όλα τα αρχεία στο πάνω μέρος υπάρχει η άδεια χρήσης της εφαρμογής και στην συνέχεια οι εισαγωγές των αρθρωμάτων που χρησιμοποιούνται. Ειδικότερα στο κύριο αρχείο της εφαρμογής αρχικά υπάρχει μια μέθοδος αρχικοποίησης των γενικών μεταβλητών, των εικονιδίων, του αρχείου ρυθμίσεων και της γλώσσας. Στη συνέχεια θα παρουσιαστεί η κύρια κλάση σχεδίασης του παραθύρου μέσα στην οποία υπάρχουν και οι περισσότεροι μέθοδοι της εφαρμογής. Η τοποθέτηση των μεθόδων είναι τέτοια έτσι ώστε να διευκολύνεται η ανάγνωση του αρχείου. Ξεκινώντας από την μέθοδο αρχικοποίησης του παραθύρου "__init__", και συνεχίζοντας στην ομάδα των μεθόδων που υλοποιούν τον μηχανισμό της Qt "signals and slots". Πιο κάτω υπάρχουν οι ομάδες που υλοποιούν τις ρυθμίσεις, τον χειρισμό των φύλλων εργασίας, τα φύλλα εργασίας ξεχωριστά και την μπάρα των γενικών πληροφοριών. Στην συνέχεια υπάρχουνε έξι κλάσεις οι οποίες υλοποιούν να νήματα που χρησιμοποιούνται και τέλος ο κώδικας αρχικοποίησης των αντικειμένων και κλήσης μεθόδων εκκίνησης της εφαρμογής.

3.3 Ανάκτηση πληροφοριών

Η ανάκτηση των πληροφοριών που εμφανίζει η εφαρμογή και που είναι κυρίως μέσα από το σύστημα αρχείων /proc του Linux όπως αναφέρθηκε σε προηγούμενο κεφάλαιο γίνεται με την χρήση των μεθόδων του αρθρώματος psutil.

Για το βασικότερο αλλά και πιο πολύπλοκο κομμάτι της εφαρμογής που είναι ο πίνακας των διεργασιών της εφαρμογής γίνεται κλήση μιας μεθόδου του psutil η οποία επιστρέφει ένα *σωρό* ο οποίος περιέχει όλες τις διεργασίες που τρέχουν εκείνη την στιγμή στο σύστημα, αναφέροντας το όνομα και τον κωδικό αριθμό της κάθε μιας. Η λίστα αυτήν των διεργασιών διαβάζεται συνεχώς από την αρχή ξανά και ξανά και για κάθε διεργασία που περιέχεται γίνεται κλήση των μεθόδων του psutil οι οποίες επιστρέφουν τις πληροφορίες που μας ενδιαφέρουν.

Σε περίπτωση που μια διεργασία έχει πάψει να υπάρχει καλείται μια μέθοδος η οποία την αφαιρεί από τον σωρό. Επίσης σε κάθε τέλος διαβάσματος του σωρού καλείται μέθοδος αναζήτησης για νέες διεργασίες που μόλις έχουν ξεκινήσει και αν υπάρχουν τότε προστίθενται και αυτές στον σωρό.

Για τις ανάγκες ανάκτησης των πληροφοριών που αφορούν το σύστημα και την χρησιμοποίηση των πόρων γίνονται επίσης κλήσεις μεθόδων του psutil οι οποίες μας επιστρέφουν την πληροφορία που μας ενδιαφέρει.

Ιδιαίτερα σημαντικό ρόλο στην εφαρμογή παίζει ο χειρισμός των εξαιρέσεων που παρέχει το psutil αλλά και η python και ο οποίος απαντάται αρκετά συχνά. Οι δυο κύριες εξαιρέσεις είναι η *NoSuchProcess* και η *AccessDenied.* Η πρώτη αφορά τις διεργασίες που έπαψαν να υπάρχουν και η δεύτερη την έλλειψη δικαιωμάτων πρόσβασης από τον χρήστη στις πληροφορίες μιας διεργασίας. Ειδικότερα για την δεύτερη σε περίπτωση που “πεταχτεί” στην προσπάθεια ανάκτησης μια πληροφορίας τότε γίνεται επιστροφή της τιμής μηδέν ή του κενού χαρακτήρα ή της φράσης “AccessDenied”, ανάλογα την περίπτωση.

3.4 Γραφική διεπαφή χρήστη

Η γραφική διεπαφή χρήστη ή GUI όπως συνηθίζεται να λέγεται από τα αρχικά της αγγλικής φράσης Graphical User Interface, σχεδιάστηκε με πρότυπο την εφαρμογή ksysguard και με την χρήση της pyqt.

Αρχικά σχεδιάστηκε ο κύριος κορμός της εφαρμογής με την εφαρμογή Qt Designer και στην συνέχεια το παραγόμενο XML αρχείο μετατράπηκε σε κώδικα python με την εντολή pyuic4. Επίσης μέσω της ίδια διαδικασίας σχεδιάστηκαν και τα επιπλέον παράθυρα “περι...”, “βοήθειας” και “ρυθμίσεων”. Για τις ανάγκες των γραφημάτων χρησιμοποιήθηκαν έτοιμα παραδείγματα τα οποία τροποποιήθηκαν κατάλληλα από την τεκμηρίωση της βιβλιοθήκης του Qwt.

Οι κύριες κλάσεις της Qt που χρησιμοποιήθηκαν είναι η *QApplicatio*n η οποία χρησιμοποιήθηκε για τον έλεγχο της ροής και των βασικών ρυθμίσεων της εφαρμογής,η *QMainWindow* για την δημιουργία του κύριου παραθύρου, η *QThread* η οποία παρέχει τα νήματα που χρησιμοποιήθηκαν, η *QTreeWidget* και *QTreeWidgetitem* για την δημιουργία όλων των πινάκων, η *QLabel* για την δημιουργία των ετικετών, η *QPushButton* και *QComboBox* για την δημιουργία κουμπιών, η *QLineEdit* για την δημιουργία πεδίων εισαγωγής κειμένου και τέλος η *QPixmap* και *QIcon* για την δημιουργία εικονιδίων και εικόνων. Επίσης για την δημιουργία και χειρισμό των γραφημάτων χρησιμοποιήθηκαν οι κλάσεις της Qwt, *QwtPlot*, *QwtPlotMarker*, *QwtScaleDraw*, *QwtPlotItem* και *QwtPlotCurve*.

Στο κύριο κομμάτι της εφαρμογής που είναι ο πίνακας των διεργασιών, όπως αναφέρθηκε στο προηγούμενο υποκεφάλαιο (3.3), μέσω του συνεχούς διαβάσματος του σωρού των διεργασιών προκύπτουν οι πληροφορίες της εκάστοτε διεργασίας. Αυτές οι πληροφορίες συντελούν στην δημιουργία ενός νέου ή στην ανανέωση ενός παλιού αντικειμένου της κλάσης *QTreeWidgetitem* το οποίο δημιουργεί μαζί με τα υπόλοιπα τον πίνακα των διεργασιών ο οποίος είναι ένα αντικείμενο τύπου *QTreeWidget*. Παρόμοια διαδικασία έχουμε και για την δημιουργία των υπόλοιπων πινάκων της εφαρμογής.

Τα νήματα της εφαρμογής χρησιμοποιούνται για να εκτελέσουν ατέρμονες βρόγχους οι οποίοι διαβάζουν συνεχώς τις πληροφορίες που παρέχουν οι κλήσεις των μεθόδων του αρθρώματος psuitl και για να “ζωγραφίσουν” μετέπειτα τις πληροφορίες αυτές μέσω των αντικειμένων των κλάσεων *QTreeWidgetitem* και *QLabel* τα πεδία της εφαρμογής. Τα νήματα εκτελούνται μόνο όταν χρειάζονται. Διαφορετικά τερματίζονται και εκκινούν και πάλι όταν χρειαστούν.

Αναλυτικότερα τα στοιχεία του παραθύρου της εφαρμογής ανανεώνονται από τα νήματα που δημιουργούνε τα αντικείμενα των κλάσεων *TableThread(QThread)* για τον πίνακα διεργασιών, *TreeThread(QThread)* για το δέντρο διεργασιών, *StatusbarThread(QThread)* για την κάτω μπάρα πληροφοριών της εφαρμογής, *TableThreadThread(QThread)* για τον πίνακα των νημάτων μιας διεργασίας, *TableConnectionsThread(QThread)* για των πίνακα συνδέσεων μιας διεργασίας, *TableDisksThread(QThread)* για τον πίνακα των δίσκων και κατατμήσεων του συστήματος.

Για την ανανέωση της σχεδίασης των γραφημάτων αλλά και για την εξόρυξη των συνεχώς νέων πληροφοριών από το σύστημα για τους πόρους χρησιμοποιείται η μέθοδος *timerEvent* της κλάσης του Qwt, *QwtPlot* σε συνδυασμό με τις μεθόδους του αρθρώματος psutil.

3.5 Μεταφράσεις και ρυθμίσεις

Η εφαρμογή διαθέτει υποστήριξη σχετικά εύκολης δημιουργίας και χρήσης μεταφράσεων σε οποιαδήποτε γλώσσα. Οι μεταφράσεις είναι τοποθετημένες στον φάκελο “translations” του python πακέτου της εφαρμογής “psymondatapack”. Οι μεταφράσεις φέρουν την κατάληξη αρχείου “.qm” και δημιουργήθηκαν από την επεξεργασία των αρχείων με κατάληξη “.ts” στο πρόγραμμα που παρέχει η Qt, QtLinguist. Τα αρχεία με κατάληξη “.ts” παράχθηκαν με την εντολή pylupdate4, μέρος επίσης της Qt, με παραμέτρους όλα τα αρχεία πηγαίου κώδικα της

εφαρμογής. Αναλυτικότερα, ο τρόπος λειτουργίας της όλης διαδικασίας δημιουργίας και εφαρμογής της μετάφρασης είναι ο εξής:

1) Σε κάθε σημείο όπου υπάρχει κείμενο ή λέξεις προς μετάφραση χρησιμοποιείται ο κώδικας:

```
QtGui.QApplication.translate(context,"Κείμενο",comment, encoding)
```

2) Η εκτέλεση της εντολής pylupdate4 διάβασε όλο τον κώδικα της εφαρμογής και απομόνωσε τις γραμμές που περιείχαν εντολές της μορφής που παρουσιάστηκαν στο προηγούμενο βήμα και δημιούργησε ένα XML αρχείο με κατάληξη “.ts”.

3) Το αρχείο με κατάληξη “.ts” φορτώθηκε στην εφαρμογή QtLinguist στο οποίο έγινε η χειροκίνητη μετάφραση των κειμένων και των λέξεων σε μια γλώσσα. Έπειτα έγινε η έκδοση ενός νέου αρχείου με κατάληξη “.qm” και περιεχόμενο την μετάφραση σε μια γλώσσα. Για κάθε γλώσσα θα πρέπει να δημιουργηθεί και ένα νέο αρχείο “.qm”. Στην παρούσα εργασία έχουμε προς το παρόν υποστήριξη μόνο της αγγλικής και της ελληνικής γλώσσας.

4) Τα αρχεία με την κατάληξη “.qm” που περιέχουν τις μεταφράσεις της εκάστοτε γλώσσας που θα επιλεχθεί είτε αυτόματα με βάση τις ρυθμίσεις του συστήματος, είτε χειροκίνητα από τον χρήστη φορτώνονται στην αρχή της εκτέλεσης της εφαρμογής με την χρήση της κλάσης *QTranslator*.

Για τις ανάγκες αποθήκευσης των ρυθμίσεων χρησιμοποιήθηκε η κλάση *QSettings* της Qt η οποία προσφέρει την δυνατότητα αυτόματου χειρισμού αποθήκευσης του αρχείου ρυθμίσεων στην κατάλληλη για το εκάστοτε λειτουργικό σύστημα τοποθεσία. Επίσης προσφέρει δυνατότητες ευκολίας δημιουργίας, ανάγνωσης και επεξεργασίας των παραμέτρων του αρχείου ρυθμίσεων.

Επίλογος

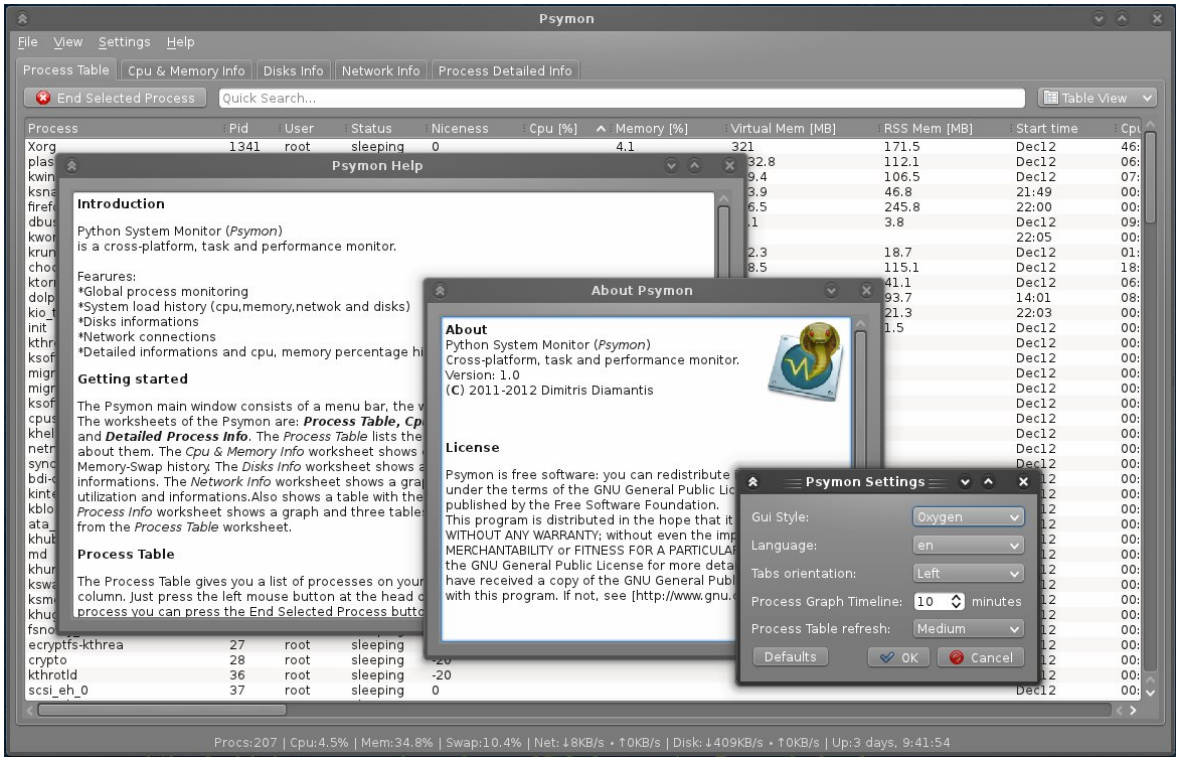
Στο παρόν κεφάλαιο είδαμε τα εργαλεία, τα υλικά καθώς και τον τρόπο που χρησιμοποιήθηκαν για να έχουμε το αποτέλεσμα που είναι η έτοιμη προς χρήση εφαρμογή. Παρακάτω θα παρουσιαστεί η εφαρμογή να δουλεύει στην πράξη.

ΚΕΦΑΛΑΙΟ 4

Psymon, η εφαρμογή

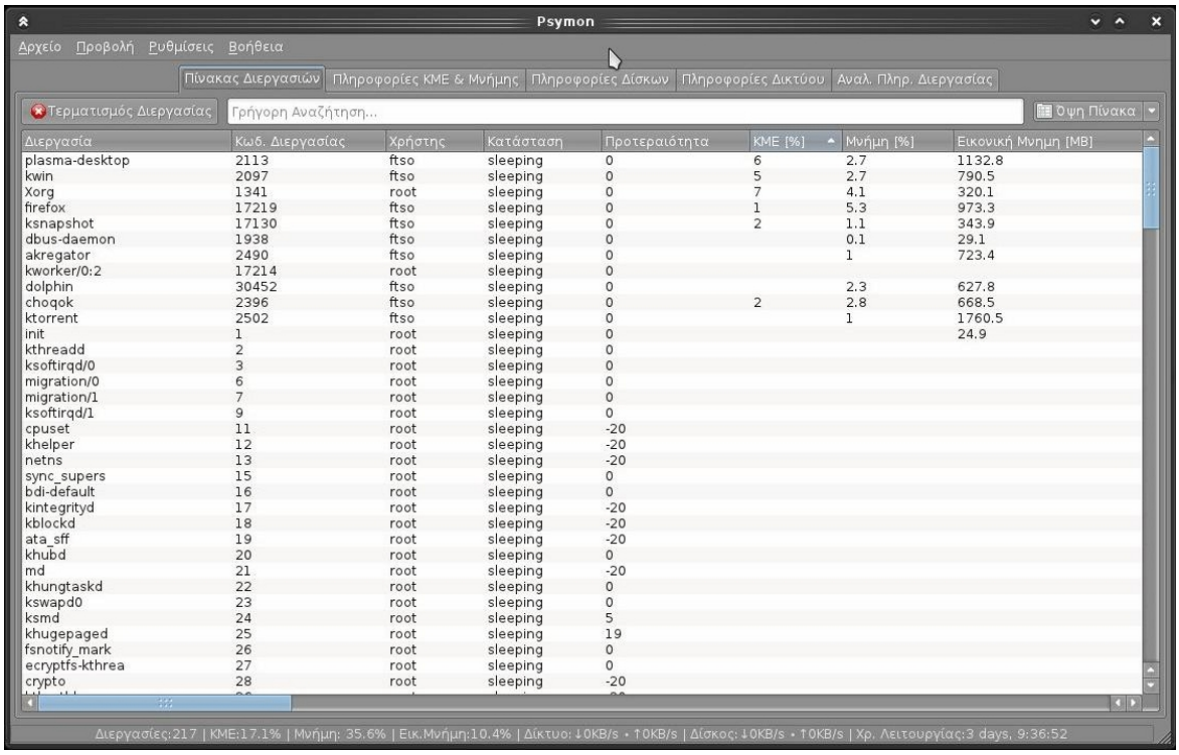
Όπως έχει γίνει αντιληπτό από την εισαγωγή της πτυχιακής εργασίας το όνομα της εφαρμογής είναι ***Psymon*** (προφέρεται ως *Σάιμον*). Το όνομα προέρχεται από τα αρχικά των λέξεων *Python*, *System* και *Monitor*. Μεταφραζόμενο ως οθόνη ελέγχου συστήματος σε γλώσσα προγραμματισμού python.

Το κύριο εικονίδιο του Psymon απεικονίζει μια οθόνη κυματομορφών στην οποία εμφανίζεται μια κυματομορφή ως προέκταση της ουράς ενός φιδιού πύθωνα. Η οθόνη κυματομορφών ως σχέδιο συνηθίζεται να χρησιμοποιείται ως εικονίδιο τέτοιου είδους εφαρμογών. Μια πρώτη γενική οπτική ιδέα της εφαρμογής παρουσιάζεται στις εικόνες *4.1* και *4.2* παρακάτω.



Εικόνα 4.1: Γενική "εικόνα" του Psymon

Όπως φαίνεται στην Εικόνα 4.1 η εφαρμογή αποτελείται από το κύριο παράθυρο το οποίο διαθέτει μια μπάρα μενού, καρτέλες ή αλλιώς φύλλα εργασίας και μια μπάρα πληροφοριών στο κάτω μέρος. Το περιεχόμενο του εκάστοτε φύλλου εργασίας περιγράφεται στον τίτλο που φέρει πάνω του και θα αναλυθεί στα επόμενα υποκεφάλαια. Επίσης στην εικόνα φαίνεται το παράθυρο με την βοήθεια, το παράθυρο με πληροφορίες για την εφαρμογή και τέλος το παράθυρο ρυθμίσεων της εφαρμογής. Στην *Εικόνα 4.2* παρακάτω παρουσιάζεται εφαρμογή με αλλαγμένες ρυθμίσεις εμφάνισης και γλώσσας.



Εικόνα 4.2: Το Psymon με τροποποιημένη εμφάνιση

4.1 Πίνακας διεργασιών

Το φύλλο εργασίας **Πίνακας Διεργασιών** διαθέτει μια περιοχή όπου υπάρχει το κουμπί **Τερματισμός Διεργασίας**, το πεδίο **Γρήγορη Αναζήτηση...** και το κουμπί-λίστα με περιεχόμενα **Όψη Πίνακα** και **Όψη Δέντρου** καθώς και μια δεύτερη

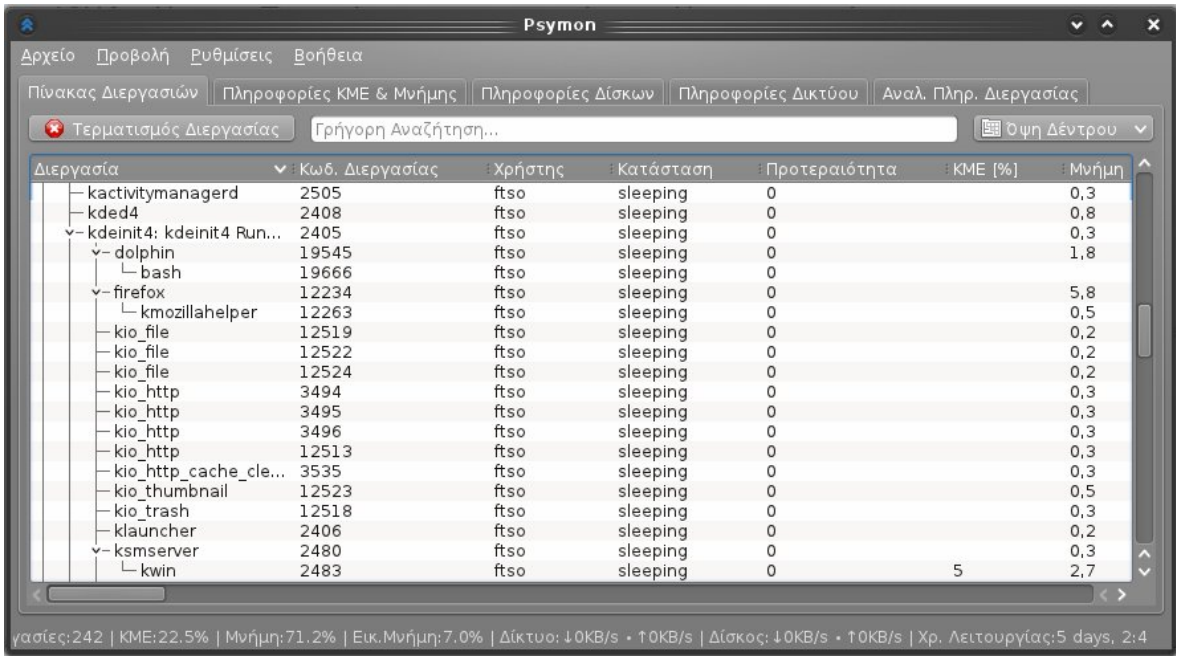
περιοχή όπου υπάρχει ο κυρίως πίνακας των διεργασιών. Ο πίνακας των διεργασιών περιέχει σταθερές κεφαλές που αφορούν την εκάστοτε διεργασία. Αυτές είναι οι εξής:

1) *Διεργασία*, για τα ονόματα.
2) *Κωδ.Διεργασίας*, για τους κωδικούς αριθμούς (pid).
3) *Χρήστης*, για τον χρήστη που την εκκίνησε.
4) *Κατάσταση*, για την τρέχουσα κατάσταση της.
5) *Προτεραιότητα*, για την προτεραιότητα της.
6) *ΚΜΕ [%]*, για το ποσοστό % χρησιμοποίησης της ΚΜΕ από αυτήν.
7) *Μνήμη [%]*, για το ποσοστό % χρησιμοποίησης της Μνήμης από αυτήν.
8) *Εικονική Μνήμη [MB]*, για το ποσοστό σε MB χρησιμοποίησης της εικονικής μνήμης.
9) *RSS Μνήμη [MB]*, για το ποσοστό σε MB χρησιμοποίησης της μνήμης RSS.
10) *Χρόνος Εκκίνησης*, για την χρονική στιγμή που άρχισε να εκτελείται.
11) *Χρόνος ΚΜΕ*, για το χρόνο χρησιμοποίησης της ΚΜΕ.
12) *Νήματα*, για τον αριθμό των εκάστοτε νημάτων της.
13) *I/O αναγνώσεις [bytes]*, για τον αριθμό των I/O αναγνώσεων σε bytes.
14) *I/O εγγραφές [bytes]*, για τον αριθμό των I/O εγγραφών σε bytes.
15) *Γονέας [όνομα,κωδικός]*, για την γονική εργασία που την εκκίνησε.
16) *Κατάλογος Εργασίας*, για τον παρόν κατάλογο εργασίας.
17) *Εντολή*, για την πλήρη εντολή που την εκκίνησε.

Η διάταξη των κεφαλών μπορεί να αλλάξει απλά κάνοντας αριστερό κλικ ποντικιού πάνω τους και σέρνοντας δεξιά ή αριστερά. Επίσης κάνοντας απλά αριστερό κλικ πάνω σε μια κεφαλή ο χρήστης μπορεί να ταξινομήσει τις διεργασίες με βάση τις τιμές που αφορούν την κεφαλή αυτή. Αν επαναλάβει το κλικ στην ίδια κεφαλή τότε μπορεί να αλλάξει την σειρά ταξινόμησης από αύξουσα σε φθίνουσα και το αντίστροφο.

Να σημειώθεί ότι ρυθμός ανανέωσης των τιμών του πίνακα μπορεί να αλλάξει σε πιο γρήγορο ή πιο αργό από το παράθυρο των γενικών ρυθμίσεων της εφαρμογής.

Το κουμπί **Τερματισμός Διεργασίας** μπορεί να φανεί χρήσιμο στην περίπτωση που ο χρήστης θέλει να τερματίσει μια διεργασία. Επιλέγοντας πρώτα μια διεργασία από τον πίνακα και πατώντας το, αποστέλλεται ένα σήμα τερματισμού στην διεργασία (SIGTERM).



Εικόνα 4.3: Psymon με επιλογή Όψη Δέντρου

Το κουμπί-λίστα με περιεχόμενα **Όψη Πίνακα** και **Όψη Δέντρου** μπορεί να χρησιμοποιηθεί για να μετατρέψει τον πίνακα που περιέχει τις διεργασίες σε

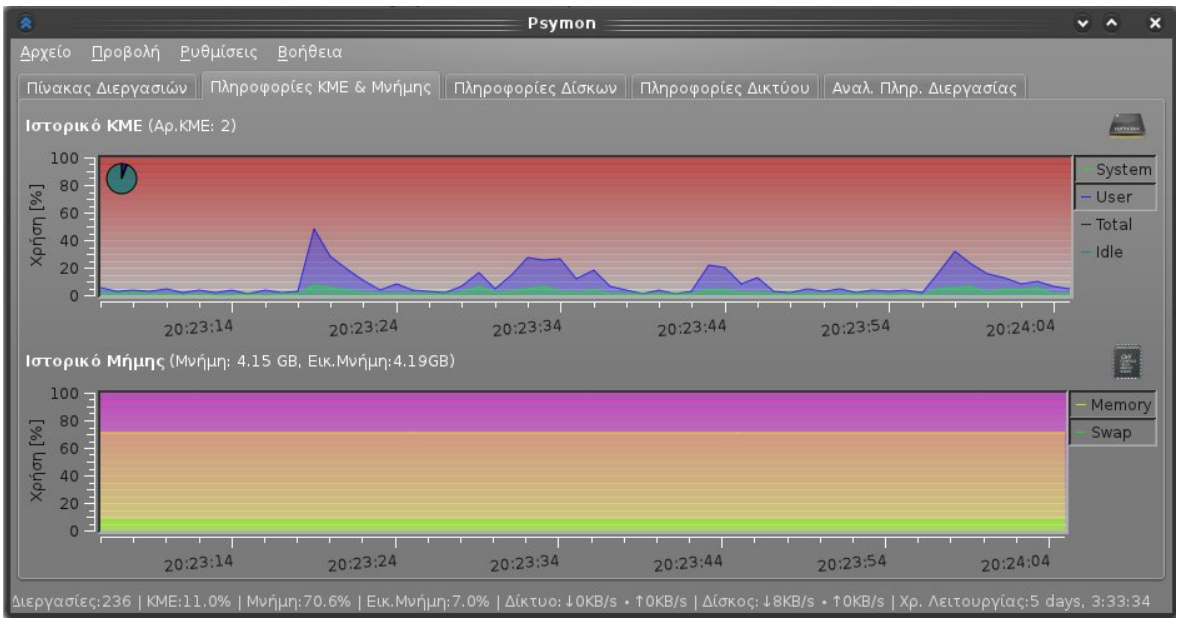
πίνακα ή δέντρο όπως φαίνεται στις *Εικόνες 4.2* και *4.3* αντίστοιχα. Στην περίπτωση του *Δέντρου* τα κλαδιά της εκάστοτε διεργασίας είναι οι διεργασίες οι οποίες εκκίνησε η ίδια. Δηλαδή αυτές για τις οποίες αποτελεί “Γονέας”.

Στο πεδίο **Γρήγορη Αναζήτηση...** αφού τοποθετήσει ο χρήστης τον κέρσορα μέσα κάνοντας απλά αριστερό κλικ, μπορεί να γράψει χαρακτήρες με βάση τους οποίους θα φιλτραριστεί ο πίνακας διεργασιών για να εμφανίσει μόνο τις διεργασίες που τους περιέχουν στο όνομα, στο κωδικό ή στο χρήστη που τις εκίνησε.

4.2 Πληροφορίες ΚΜΕ και Μνήμης

Στο φύλλο εργασίας **Πληροφορίες ΚΜΕ & Μνήμης** εμφανίζονται δυο γραφήματα, το ένα κάτω από το άλλο όπως φαίνεται στην *Εικόνα 4.4*.

Στο πρώτο γράφημα παρουσιάζεται το ιστορικό του ποσοστού % του φόρτου της ΚΜΕ για το σύστημα, το χρήστη και συνολικά. Επίσης παρουσιάζεται και το αντίστροφο του συνολικού. Στα δεξιά του γραφήματος υπάρχουν τα κουμπιά τα οποία μπορούν να χρησιμοποιηθούν για να εμφανίσουν ή να αποκρύψουν τις διαθέσιμες κυματομορφές. Η γραμμή του χρόνου στο άξονα Χ έχει εύρος ένα λεπτό και ανανεώνεται κάθε δευτερόλεπτο. Μέσα στο γράφημα υπάρχει και ένα μικρότερο σε μορφή “πίτας” που μας δείχνει το μέρος από το σύνολο που χρησιμοποιεί το σύστημα και ο χρήστης. Επίσης δίπλα από τον τίτλο, μέσα σε παρενθέσεις δίνεται και ο αριθμός των πυρήνων που διαθέτει η ΚΜΕ του συστήματός μας.



Εικόνα 4.4: Psymon, Πληροφορίες ΚΜΕ & Μνήμης

Το δεύτερο γράφημα έχει την ίδια λειτουργικότητα και χαρακτηριστικά με το πρώτο. Η μόνη διαφορά είναι ότι τα γραφήματα είναι δυο και αφορούν την χρήση % της μνήμης και της εικονικής μνήμης. Επίσης δίπλα απο τον τίτλο εμφανίζεται και εδώ μέσα σε παρενθέσεις το σύνολο των δυο πόρων σε GB.

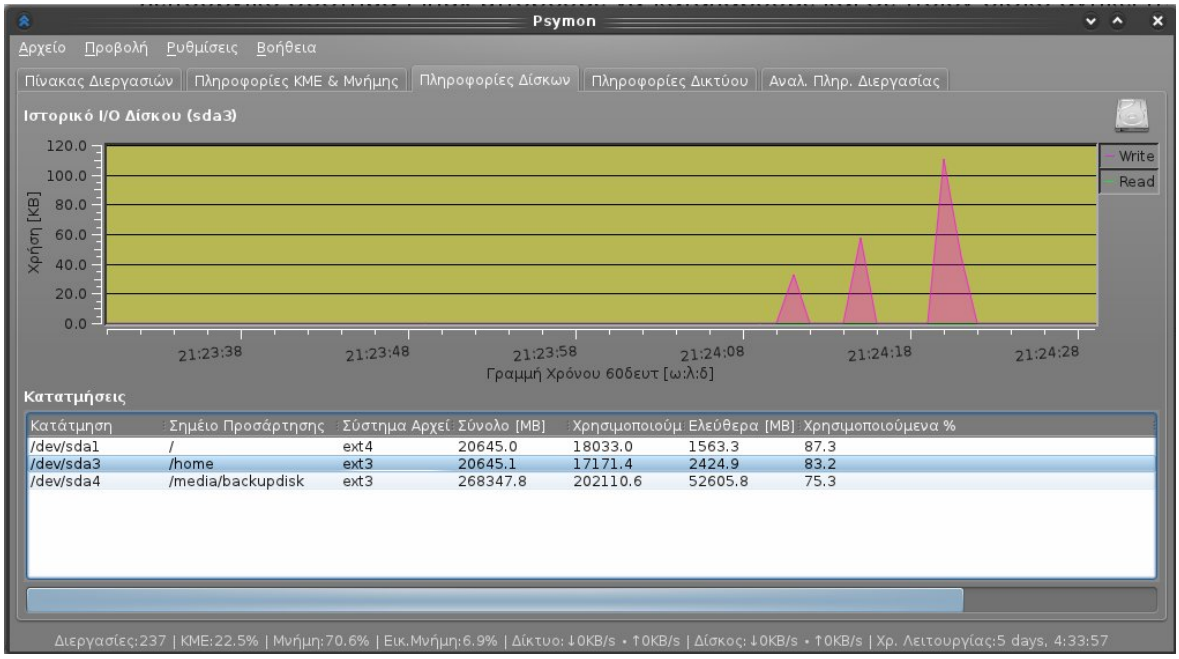
4.3 Πληροφορίες Δίσκων

Το φύλλο εργασίας **Πληροφορίες Δίσκων** αποτελείται από τρία κομμάτια. Το γράφημα, τον πίνακα και μια μπάρα εμφάνισης ποσοστού.

Το γράφημα στην αρχική του μορφή μόλις εμφανιστεί το φύλλο εργασίας δείχνει την χρησιμοποίηση όλων των δίσκων συνολικά. Οι κυματομορφές που διαθέτει είναι η **Write** για τις έγραφες και η **Read** για τις αναγνώσεις. Στον άξονα Χ υπάρχει ο χρόνος ο οποίος έχει εύρος ένα λεπτό και ανανεώνεται κάθε δευτερόλεπτο, ενώ στο άξονα Υ υπάρχει το εύρος των τιμών χρήσης σε KB το οποίο αλλάζει δυναμικά ανάλογα με την μεγαλύτερη τιμή. Στα δεξιά υπάρχουν τα κουμπιά ενεργοποίησης και απενεργοποίησης της κάθε κυματομορφής.

Στο δεύτερο κομμάτι του φύλλου εργασίας και κάτω από γράφημα υπάρχει ο πίνακας των κατατμήσεων του συνόλου των δίσκων του συστήματος μας.

Οι πληροφορίες που παρατίθενται για κάθε κατάτμηση είναι η ίδια η κατάτμηση με την μορφή διαδρομής η οποία περιέχει τον κατάλογο /dev και το όνομα της κατάτμησης, το σημείο προσάρτησης, το σύστημα αρχείων που χρησιμοποιεί, το σύνολο της χωρητικότητας σε MB, τα ελεύθερα MB και τέλος ποσοστό χρησιμοποίησης του χώρου σε MB και %. Από το όνομα της κατάτμησης στο λειτουργικό σύστημα Linux ο χρήστης μπορεί να καταλάβει και σε ποιον δίσκο ανήκει η κάθε κατάτμηση.

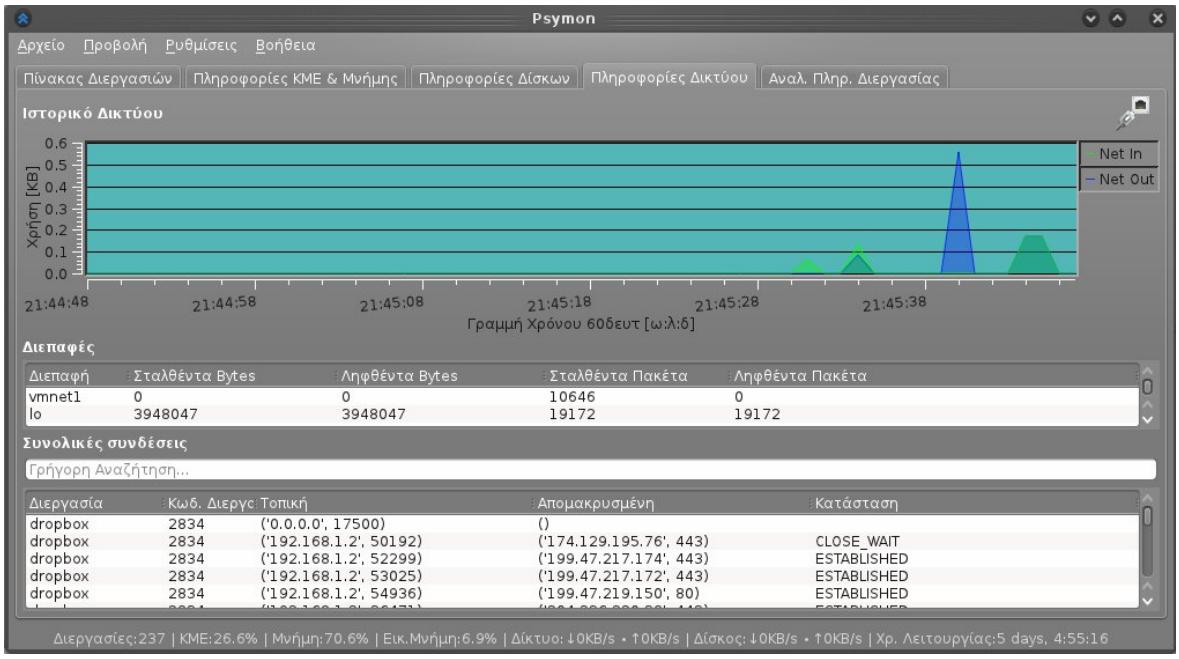


Εικόνα 4.5: Psymon, Πληροφορίες Δίσκων

Όταν ο χρήστης επιλέξει μια κατάτμηση κάνοντας απλά αριστερό κλικ επάνω της μέσα στον πίνακα τότε το γράφημα παύει να εμφανίζει την χρήση όλων των δίσκων και των κατατμήσεων τους αλλά μόνο της κατάτμησης που επιλέξαμε. Επίσης η μπάρα κάτω από τον πίνακα των κατατμήσεων εμφανίζει το ποσοστό χρήσης του χώρου % της κατάτμησης που έχουμε επιλέξει. Αν κάνουμε κλικ επάνω σε μια ήδη επιλεγμένη κατάτμηση τότε αυτήν αποεπιλέγεται.

4.4 Πληροφορίες Δικτύου

Στην καρτέλα **Πληροφορίες Δικτύου** εμφανίζεται κάτι αντίστοιχο με την προηγούμενη καρτέλα τόσο σε σχεδίαση όσο και σε λειτουργικότητα. Η διαφορά εδώ είναι ότι εμφανίζονται διεπαφές δικτύου και όχι κατατμήσεις. Επίσης μια άλλη σημαντική διαφορά είναι ότι λείπει τη μπάρα που υπήρχε στην προηγούμενη καρτέλα κάτω από τον πίνακα των κατατμήσεων, στην θέση της οποίας υπάρχει ένα πεδίο αναζήτησης και ένας ακόμη πίνακας ο οποίος εμφανίζει τις συνολικές συνδέσεις δικτύου του συστήματος. Το πεδίο αναζήτησης δουλεύει πανομοιότυπα με το πεδίο αναζήτησης του φύλλου εργασίας **Πίνακας Διεργασιών**.



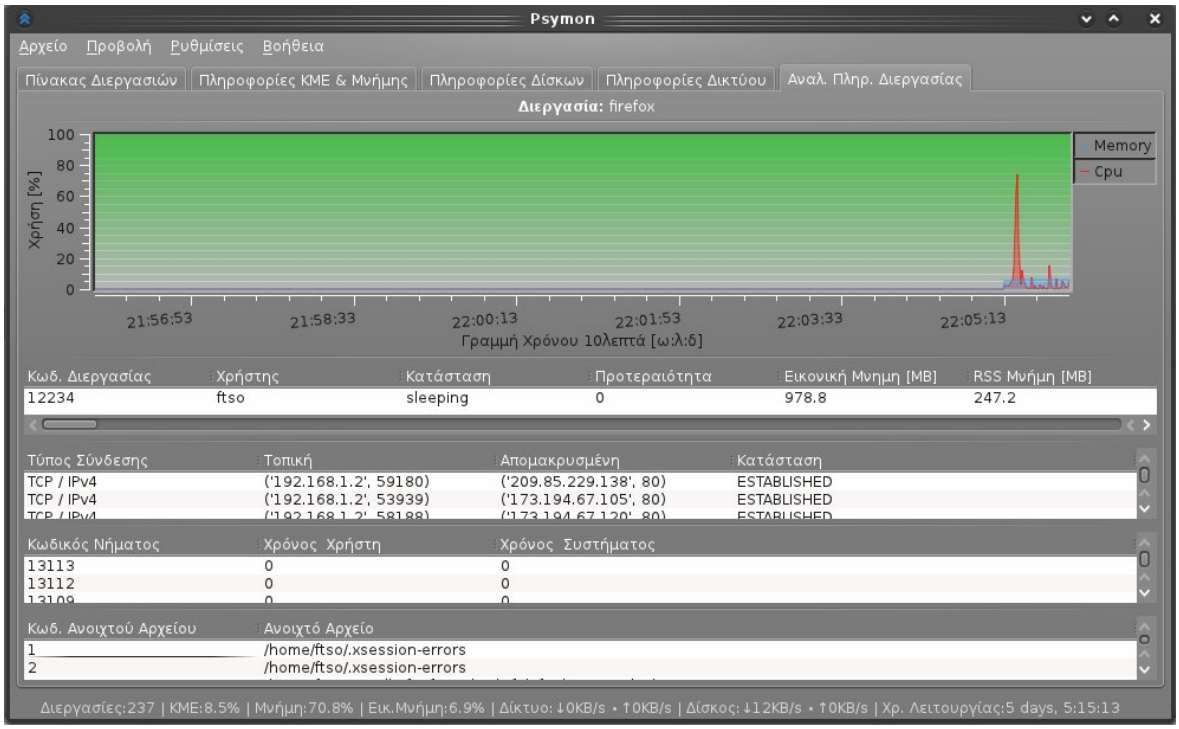
Εικόνα 4.6: Psymon, Πληροφορίες Δικτύου

Οι κεφαλές που διαθέτει ο πίνακας των διεπαφών είναι η *Διεπαφή*, τα *Σταλθέντα Bytes*, τα *Ληφθέντα Bytes*, τα *Σταλθέντα Πακέτα* και τέλος τα *Ληφθέντα Πακέτα*. Στο πίνακα των συνολικών συνδέσεων υπάρχουν οι κεφαλές *Διεργασία* με το όνομα της διεργασίας,τον *Κωδ.Διεργασίας*,την *Τοπική* με την IP και την θύρα της σύνδεσης, την *Απομακρυσμένη* με επίσης την IP και την θύρα της σύνδεσης και τέλος την *Κατάσταση* της σύνδεσης.

4.5 Αναλυτικές πληροφορίες διεργασίας

Το τελευταίο φύλλο εργασίας είναι και το πιο σημαντικό καθώς υλοποιεί το βασικό χαρακτηριστικό της εφαρμογής που είναι οι αναλυτικές πληροφορίες και τα γραφήματα για μια μεμονωμένη κάθε φορά διεργασία.

Για να δει ο χρήστης τις πληροφορίες μιας διεργασίας σε αυτό το φύλλο εργασίας θα πρέπει πρώτα να την έχει επιλέξει από τον πίνακα των διεργασιών του πρώτου φύλλου εργασίας και στην συνέχεια να μεταβεί στο παρόν. Εναλλακτικά μπορεί απλά να κάνει διπλό αριστερό κλικ ποντικιού πάνω στην διεργασία του πίνακα διεργασιών και να μεταφερθεί απευθείας στην τελευταία καρτέλα.



Εικόνα 4.7: Psymon, Αναλ. Πληρ. Διεργασίας

Ξεκινώντας από πάνω προς τα κάτω τα στοιχεία που εμφανίζονται είναι το όνομα της διεργασίας, ένα γράφημα εμφάνισης ποσοστού % χρησιμοποίησης της μνήμης και της ΚΜΕ από την διεργασία, ένα μονόστηλο πίνακα που εμφανίζει ένα σύνολο πληροφοριών για την διεργασία, ένα πίνακα με τις συνδέσεις της διεργασίας, ένα

πίνακα με τα νήματα της διεργασίας και τέλος ένα πίνακα με τα ανοιχτά από την διεργασία αρχεία.

Χαρακτηριστικό του γραφήματος αυτού του φύλλου διεργασίας είναι η δυνατότητα επιλογής του εύρους της γραμμής του χρόνου (άξονας Χ). Η ρύθμιση μπορεί να γίνει μέσω του παραθύρου των ρυθμίσεων της εφαρμογής.

Επίλογος

Το παρόν κεφάλαιο αποτελεί κατα κάποιο τρόπο και το εγχειρίδιο χρήσης της εφαρμογής στην ελληνική γλώσσα. Στην συνέχεια θα ακολουθήσει μια ανάλυση της άδειας χρήσης που χρησιμοποιήθηκε για την εφαρμογή καθώς και το μοντέλο ανάπτυξης που ακολουθεί η εφαρμογή.

ΚΕΦΑΛΑΙΟ 5

Άδεια χρήσης και το μοντέλο ανάπτυξης

Στο παρόν κεφάλαιο θα γίνει παρουσίαση της άδεια χρήσης GNU GPL που χρησιμοποιήθηκε για την εφαρμογή Psymon αλλά και το “ανοιχτό” μοντέλο ανάπτυξης που χρησιμοποιείται στην ανάπτυξη εφαρμογών που φέρουν αυτήν την άδεια.

5.1 Η άδεια χρήσης GNU GPL

Η άδεια χρήσης GNU GPL (General Public License) δημιουργήθηκε αρχικά για τα προγράμματα που έγραφε το Gnu Project. Η άδεια αυτή δίνει στους κατόχους ενός προγράμματος τα ακόλουθα τέσσερα δικαιώματα, που στην κοινότητα του ελεύθερου λογισμικού είναι γνωστά και ως *Τέσσερις Ελευθερίες:*

- να τρέξουν ένα πρόγραμμα για οποιοδήποτε λόγο.
- να μελετήσουν τη λειτουργία ενός προγράμματος και να το τροποποιήσουν
- να διανείμουν αντίγραφα του προγράμματος έτσι ώστε να βοηθήσουν τον πλησίον
- να βελτιώσουν το πρόγραμμα και να προσφέρουν τις βελτιώσεις στο κοινό, έτσι ώστε να ωφεληθεί ολόκληρη η κοινότητα

Προϋποθέσεις για τα παραπάνω είναι ο ανοιχτός κώδικας, δηλαδή ο κώδικας του προγράμματος να είναι γνωστός και προσβάσιμος στον χρήστη. Κάθε παράγωγο και προϊόν ενός GPL προγράμματος, υποχρεούται να κυκλοφορεί κι αυτό υπό την ίδια άδεια.

Η διαφορά μεταξύ GNU και ιδιόκτητου λογισμικού είναι στο ότι το τελευταίο δεν δίνει κανένα δικαίωμα στο χρήστη, πέραν από το δικαίωμα χρήσης και αυτό κατόπιν συμφωνίας με τον ιδιοκτήτη του προγράμματος (με τη μορφή μιας *End User License Agreement*). Κριτικοί του ιδιόκτητου λογισμικού υποστηρίζουν ότι δεν

πρέπει να χρειάζεται άδεια για χρήση και υποστηρίζουν ότι οι περιορισμοί αυτοί και η διάθεση από τους ιδιοκτήτες του λογισμικού μόνο των δυαδικών πακέτων και όχι του πηγαίου κώδικα του προγράμματος απαγορεύουν νόμιμες διαδικασίες όπως η ανάστροφη μηχανική.

Το λογισμικό ανοιχτού κώδικα εκ φύσεως είναι πιο ευάλωτο στην καταπάτηση των δικαιωμάτων των προγραμματιστών από χρήστες ή εταιρείες που χρησιμοποιούν κώδικα χωρίς να σέβονται τις άδειες υπό τις οποίες αυτός δημοσιεύεται. Η GPL αναγκάζει το οποιοδήποτε λογισμικό χρησιμοποιεί ή βασίζεται σε κώδικα που κυκλοφορεί υπό την GPL, να δώσει την δυνατότητα στους χρήστες που το επιθυμούν να δουν τον κώδικα. Παρόλα ταύτα, μερικές υποθέσεις έφτασαν ως τα δικαστήρια όπου και αποδείχτηκε η ασφάλεια και σιγουριά της άδειας αυτής και η πρόθεση του Free Software Foundation να προασπίσει τις τέσσερις ελευθερίες που εγγυάται η άδεια.

5.2 Το "ανοιχτό" μοντέλο ανάπτυξης

Τις τελευταίες δεκαετίες, η κύρια εμπορική δραστηριότητα στο χώρο λογισμικού είχε τη μορφή πώλησης αδειών χρήσης (για περιορισμένο χρονικό διάστημα ή για απεριόριστο χρόνο), ενώ ο πηγαίος κώδικας αποτελούσε μυστικό των κατασκευαστών.

Ωστόσο, η διαθεσιμότητα του πηγαίου κώδικα έχει πολλαπλασιαστικά οφέλη, καθώς οδηγεί στη δημιουργία καλύτερου τελικού προϊόντος, κυρίως μέσω της συμμετοχής περισσοτέρων ατόμων και της συνολικής συνεισφοράς εργασίας που είναι πολλαπλάσια από αυτή που μπορεί να διαθέσει οποιαδήποτε εμπορική επιχείρηση. Προφανές παράδειγμα αποτελεί η μετάφραση και η τοπικοποίηση του λογισμικού σε περιβάλλοντα και χώρες μακριά από την αρχική κατασκευάστρια εταιρεία.

Στην περίπτωση των μεμονωμένων παραγωγών, η διάθεση του πηγαίου κώδικα βοηθάει την αναγνωρισιμότητα και τη φήμη του δημιουργού, που μπορεί να έχει ευεργετικά αποτελέσματα στην επαγγελματική του σταδιοδρομία.

Επίλογος

Η άδεια χρήσης GPL και το “ανοιχτό” μοντέλο ανάπτυξης προτιμούνται από όλο και περισσότερους προγραμματιστές και μικρές ή μεγάλες εταιρίες του χώρου της ανάπτυξης εφαρμογών. Τα παραγόμενα προϊόντα δε αυτού του μοντέλου κερδίζουν συνεχώς νέους φίλους οι οποίοι στην πλειοψηφία του εκτός από απλοί χρήστες γίνονται μέρος της ανάπτυξης του εκάστοτε προϊόντος με διάφορους τρόπους όπως οι μεταφράσεις, η ανάπτυξη κώδικα επέκτασης, η αναφορά σφαλμάτων και η συγγραφή τεκμηριώσεων.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Η ανάπτυξη λογισμικού για το λειτουργικό σύστημα GNU/Linux είναι σχετικά εύκολη διαδικασία αν συγκριθεί με την ανάπτυξη σε άλλες πλατφόρμες. Η ευκολία αυτή οφείλετε κυρίως στην διάθεση του κώδικα των εφαρμογών του αλλά και του ίδιου του λειτουργικού. Επίσης ένα άλλο σημαντικό στοιχείο είναι η δομή των εφαρμογών του οι οποίες στην πλειοψηφία τους μοιάζουν σαν ένα παζλ φτιαγμένο από μικρότερα προγράμματα τα οποία χρησιμοποιούνται και από άλλες εφαρμογές. Έτσι ο χρήστης ή ο προγραμματιστής αντί να συμβιβαστεί με την λύση Α ή την λύση Β η οποία είτε θα ελλείπεται κάποιου χαρακτηριστικού που θα ήθελε είτε θα παρέχει δεκάδες άλλα χαρακτηριστικά τα οποία του είναι άχρηστα κάνοντας την δουλειά του δυσκολότερη, μαζεύει όλα τα μικρά κομμάτια και συνθέτει το δικό του παζλ, δηλαδή μια εφαρμογή στα μέτρα του.

Η γλώσσα προγραμματισμού Python αξίζει προσοχής από κάθε προγραμματιστή. Οι δυνατότητές της και ταυτόχρονα η ευκολία και φιλοσοφία της θα μαγέψουν τον καθένα που θα ασχοληθεί μαζί της. Σε συνδυασμό μάλιστα με την Qt μπορούν να αναπτυχθούν παντός τύπου εφαρμογές με πλειάδα χαρακτηριστικών και καλαίσθητης γραφικής διεπαφής χρήστη.

Το “ανοιχτό” μοντέλο ανάπτυξης λογισμικού και τα προϊόντα του κερδίζουν όλο και περισσότερους οπαδούς. Αυτό οφείλεται τόσο στην ελευθερία χρήσης όσο και στη δωρεάν διάθεση των προϊόντων αυτών. Ο απλός χρήστης ή ο προγραμματιστής μπορεί να γίνει μέρος της ομάδας ανάπτυξης οποιουδήποτε προϊόντος προσφέροντας είτε στο τεχνικό κομμάτι είτε στο λιγότερο ή καθόλου τεχνικό.

ΑΝΑΦΟΡΕΣ

Wikipedia 2011. A free encyclopedia built collaboratively using wiki software. [προσπελάστηκε 23 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://wikipedia.org

Debian gr 2011. To Debian στην Ελλάδα. [προσπελάστηκε 19 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://debian.gr

Telemathea uom. 2011. Ηλεκτρονική βιβλιοθήκη πανεπιστημίου Μακεδονίας. [προσπελάστηκε 18 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://telemathea.uom.gr

Python homepage 2011. Python Programming Language – Official Website. [προσπελάστηκε 20 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://python.org

Swaroopch. A Byte of Python 2011.free book on programming using the Python language [προσπελάστηκε 21 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://www.swaroopch.com/notes/Python

Qt homepage 2011. A cross-platform application and UI framework. [προσπελάστηκε 21 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://qt.nokia.com

Psutil homepage 2011. A cross-platform process and system utilities module for Python [προσπελάστηκε 22 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://code.google.com/p/psutil/

Riverbank computing 2011. PyQt4 [προσπελάστηκε 22 Δεκεμβρίου 2011]. Διαθέσιμη από την ιστοσελίδα: http://www.riverbankcomputing.co.uk/software/pyqt/intro

ΠΑΡΑΡΤΗΜΑΤΑ

Π1 *Αρχεία πηγαίου κώδικα της εφαρμογής*

Εξερεύνηση και ανάγνωση των αρχείων του πηγαίου κώδικα της εφαρμογής μπορεί να γίνει στην παρακάτω διεύθυνση φιλοξενίας έργων λογισμικού ανοιχτού κώδικα της εταιρίας Google:

- URL:
  `http://code.google.com/p/psymon/source/browse/trunk`

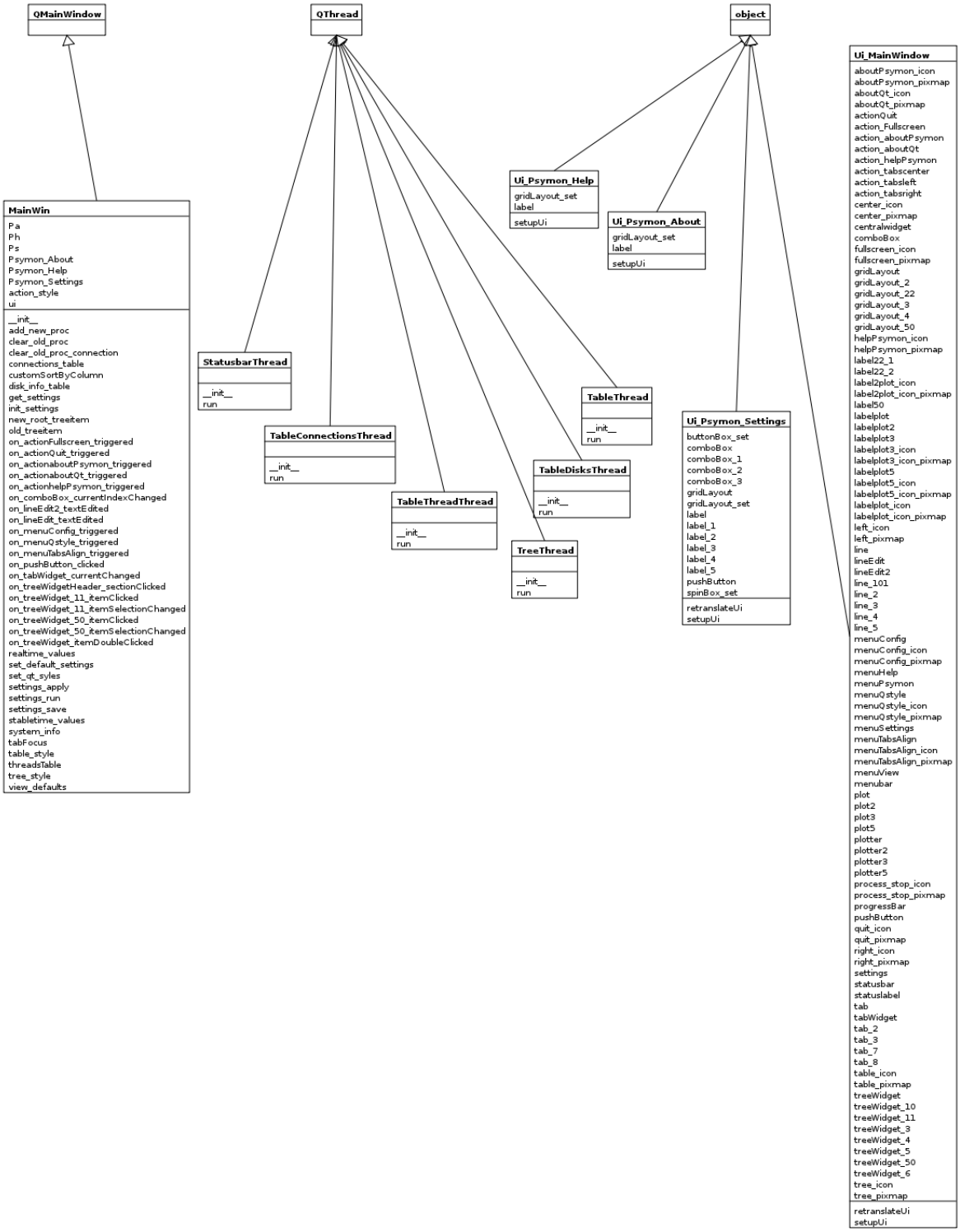
- Sorted URL:
  `http://tinyurl.com/psymon-code`

- Qr Code:

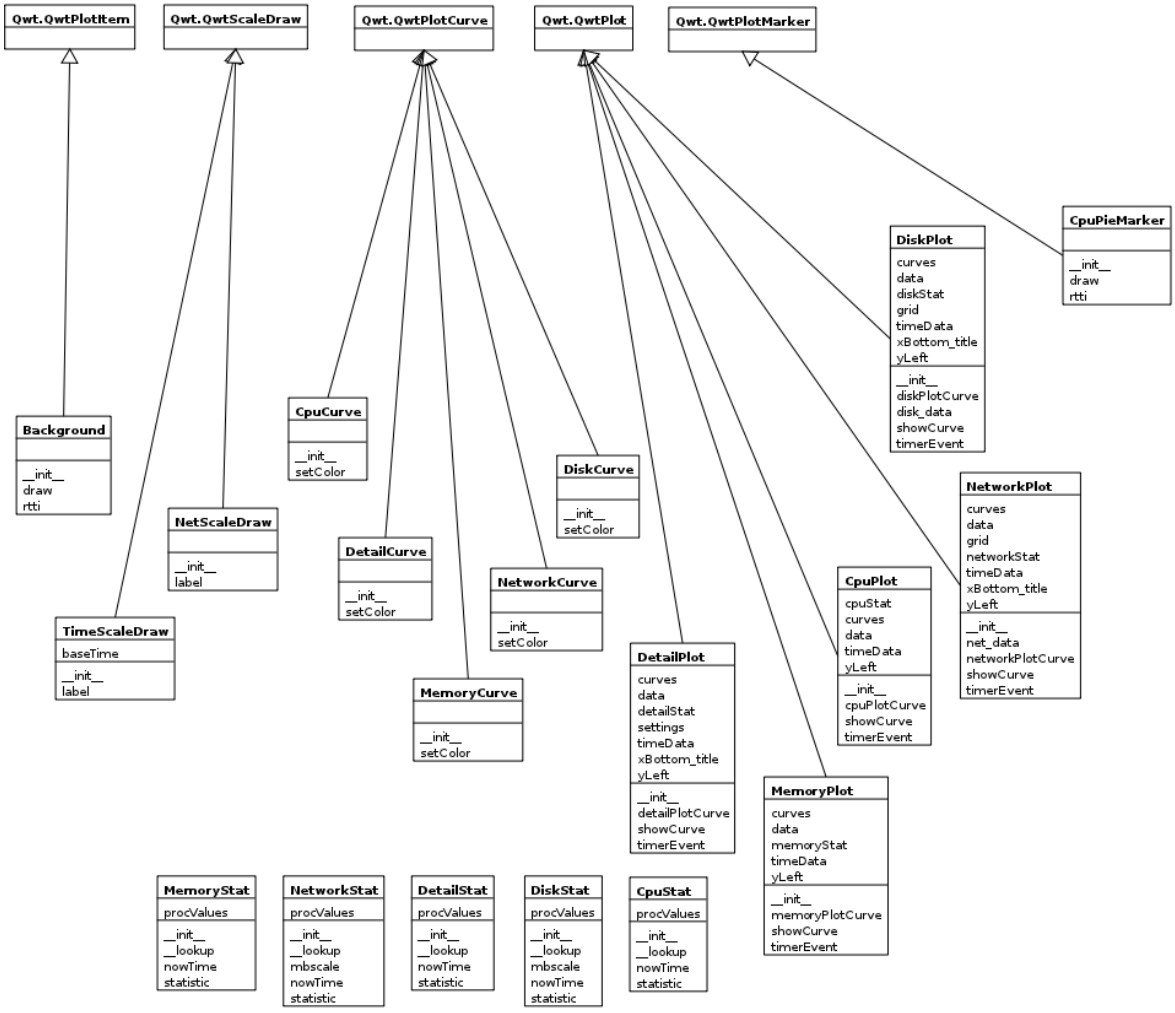
Εναλλακτικά μπορεί να γίνει κατέβασμα αντιγράφου του πηγαίου κώδικα και ανάγνωση τοπικά μέσω της παρακάτω εντολής της εφαρμογής ελέγχου εκδόσεων *Subversion*:

```
svn checkout http://psymon.googlecode.com/svn/trunk/ psymon-read-only
```

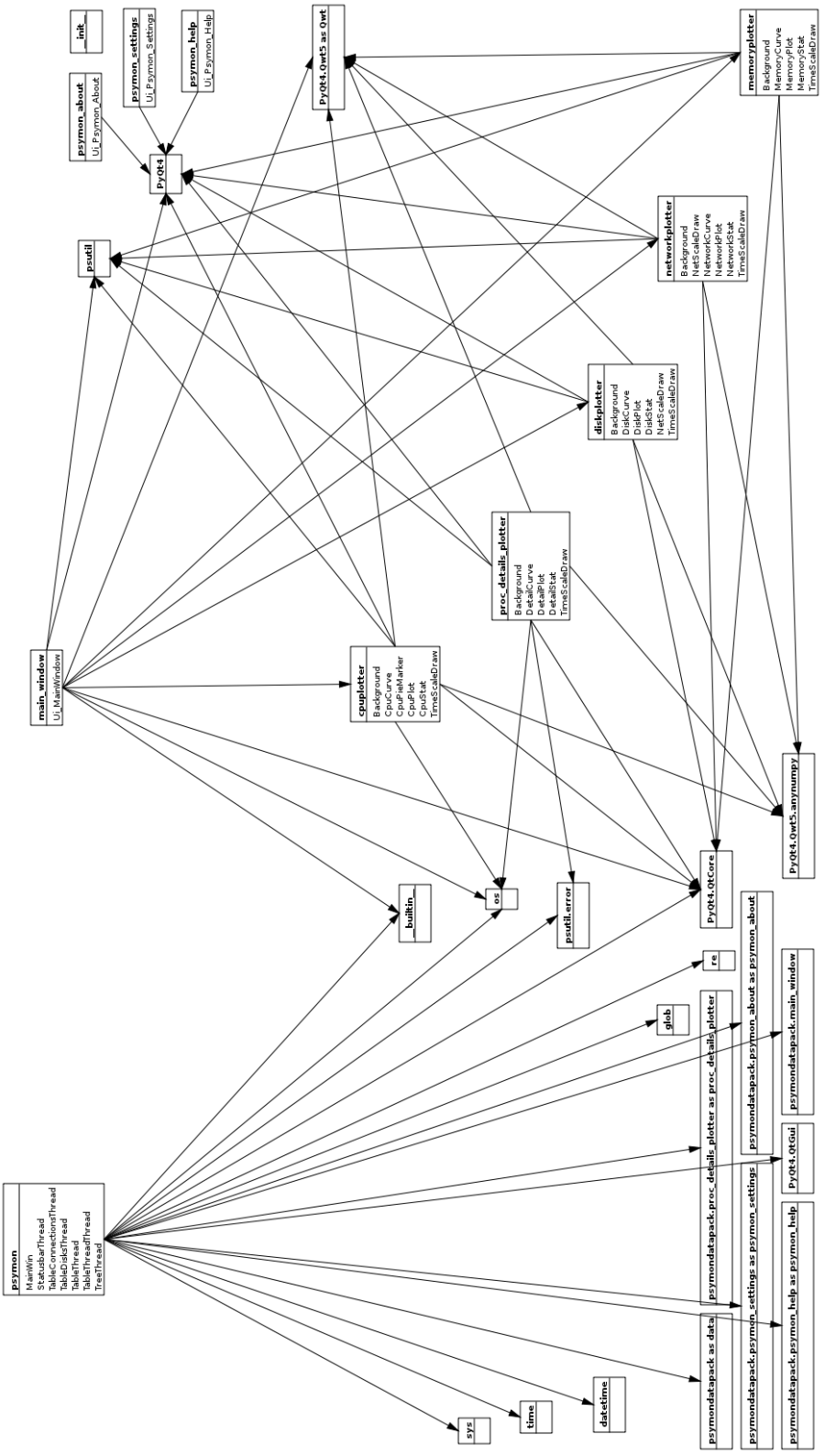
Π2 *Διαγράμματα*



Διάγραμμα κλάσεων (μέρος 1)



Διάγραμμα κλάσεων (μέρος 2)



Διάγραμμα αρθρωμάτων (ανάγνωση στα πλάγια)

Π3 *Ενδεικτικά κομμάτια πηγαίου κώδικα*

Π3.1 Παράδειγμα χρήσης του μηχανισμού της Qt, Signals-Slots

```
@pyqtSlot(QString)
    def on_lineEdit2_textEdited(self,searchtext):
        for list_item in self.ui.treeWidget_10.findItems("*",
                        Qt.MatchWildcard|Qt.MatchRecursive,0):
            if searchtext != None and searchtext in ("$","\\","*","(",")","+"):
                pass
            elif searchtext != None and (
                re.search(str(searchtext),list_item.text(0),flags=re.IGNORECASE)
                or re.search(str(searchtext),list_item.text(1),flags=re.IGNORECASE)
                or re.search(str(searchtext),list_item.text(2),flags=re.IGNORECASE)
                or re.search(str(searchtext),list_item.text(3),flags=re.IGNORECASE)
                or re.search(str(searchtext),list_item.text(4),flags=re.IGNORECASE))
                                                                != None:
                list_item.setHidden(False)
            else:
                list_item.setHidden(True)


def on_menuTabsAlign_triggered(self,_action):
        if _action.text() == "Center":
            self.ui.tabWidget.setStyleSheet("QTabWidget::tab-bar{alignment: center;}")
        elif _action.text() == "Left":
            self.ui.tabWidget.setStyleSheet("")
        elif _action.text() == "Right":
            self.ui.tabWidget.setStyleSheet("QTabWidget::tab-bar{alignment: right;}")
        else:
            self.ui.tabWidget.setStyleSheet("")
```

Π3.2 Μπάρα πληροφοριών της εφαρμογής

```
def system_info(self):
        while True:
            _proc_list = [p for p in psutil.process_iter()]
            _total_procs =QtGui.QApplication.translate("MainWindow","Procs:",
                            None, QtGui.QApplication.UnicodeUTF8)+str(len(_proc_list))
            _cpu = QtGui.QApplication.translate("MainWindow","Cpu:",
                            None,
QtGui.QApplication.UnicodeUTF8)+str(psutil.cpu_percent(interval=1, percpu=False))+"%"
            _mem = psutil.phymem_usage()
            _vmem = psutil.virtmem_usage()
            _mem_inf=QtGui.QApplication.translate("MainWindow","Mem:",
                            None, QtGui.QApplication.UnicodeUTF8)+str(_mem.percent)+"%"
            _vmem_inf=QtGui.QApplication.translate("MainWindow","Swap:",
                            None, QtGui.QApplication.UnicodeUTF8)+str(_vmem.percent)+"%"
            _uptime = QtGui.QApplication.translate("MainWindow","Up:",
                            None,
QtGui.QApplication.UnicodeUTF8)+_fromUtf8(str(datetime.timedelta(
                    seconds=int(time.time() - psutil.BOOT_TIME))))
            _in_a = psutil.network_io_counters(pernic=False).bytes_recv
            _out_a = psutil.network_io_counters(pernic=False).bytes_sent
            _read_a = psutil.disk_io_counters(perdisk=False).read_bytes
            _write_a = psutil.disk_io_counters(perdisk=False).write_bytes
            time.sleep(0.95)
            _in_b = psutil.network_io_counters(pernic=False).bytes_recv
            _out_b = psutil.network_io_counters(pernic=False).bytes_sent
            _read_b = psutil.disk_io_counters(perdisk=False).read_bytes
            _write_b = psutil.disk_io_counters(perdisk=False).write_bytes
            _netdata = QtGui.QApplication.translate("MainWindow","Net:",
                            None, QtGui.QApplication.UnicodeUTF8)+u"\u2193"+str((_in_b -
_in_a)/1000
                    )+u"KB/s \u2219\u2191"+str((_out_b - _out_a)/1000)+"KB/s"

            _diskdata = QtGui.QApplication.translate("MainWindow","Disk:",
                            None, QtGui.QApplication.UnicodeUTF8)+u"\u2193"+str((_write_b -
_write_a
                    )/1000)+u"KB/s \u2219\u2191"+str((_read_b - _read_a)/1000)+"KB/s"

            self.ui.statuslabel.setText(_total_procs+" | "+_cpu+" | "+_mem_inf+" | "+
                            _vmem_inf+" | "+_netdata+" | "+_diskdata+" | "+_uptime)
```

Π3.3 Πίνακας διεργασιών εφαρμογής

```
def customSortByColumn(self,column):
        order = self.ui.treeWidget.header().sortIndicatorOrder()
        time.sleep(0.5)
        self.ui.treeWidget.sortItems(column, order)

def clear_old_proc(self,_proc_list,_proc):

    _proc_list.remove(_proc)
    time.sleep(0.5)
    for list_item in self.ui.treeWidget.findItems("*",
                    Qt.MatchWildcard|Qt.MatchRecursive,0):

        if psutil.pid_exists(int(list_item.text(1))) == False:

            if list_item.parent() == None:
                list_item_index = self.ui.treeWidget.indexOfTopLevelItem(list_item)
                self.ui.treeWidget.takeTopLevelItem(list_item_index)
            else:
                list_item_index = list_item.parent().indexOfChild(list_item)
                list_item.parent().takeChild(list_item_index)

    return _proc_list

def add_new_proc(self,_proc_list):
    _selfprocid = psutil.Process(os.getpid()).pid
    cpids = [p.pid for p in _proc_list]
    for p in psutil.process_iter():
        if p.pid not in cpids:
            if _selfprocid != p.pid:
                _proc_list.append(p); _proc_list.reverse()
    return _proc_list
def new_root_treeitem(self,_name,_pid,_username,_niceness,
                      _parent_pid,_parent_name,_command,_start):
    item_tree = QTreeWidgetItem()
    item_tree.setText(0,str(_name))
    item_tree.setText(1,str(_pid))
    item_tree.setText(2,str(_username))
    item_tree.setData(3,Qt.DisplayRole, "")
    item_tree.setData(4,Qt.DisplayRole, _niceness)
    item_tree.setData(5,Qt.DisplayRole, "")
    item_tree.setData(6,Qt.DisplayRole, "")
    item_tree.setData(7,Qt.DisplayRole, "")
    item_tree.setData(8,Qt.DisplayRole, "")
    item_tree.setData(9,Qt.DisplayRole, _start)
```

```
    item_tree.setData(10,Qt.DisplayRole, "")
    item_tree.setData(11,Qt.DisplayRole, "")
    item_tree.setData(12,Qt.DisplayRole, "")
    item_tree.setData(13,Qt.DisplayRole, "")
    item_tree.setText(14, _parent_name+" , "+ str(_parent_pid))
    item_tree.setText(15,"")
    if str(_command) == "[]":
        item_tree.setText(16,"")
    else:
        item_tree.setText(16,str(_command))
    return item_tree
def old_treeitem(self,old_item,_vsz,_rss,_cputime,_threads,_read_bytes,
            _write_bytes,_memory_percent,_cpu_percent,_status,_working_dir):
    if old_item.text(7) != _vsz and _vsz != 0:
        old_item.setData(7,Qt.DisplayRole, _vsz)
    if old_item.text(8) != _rss and _rss !=0:
        old_item.setData(8,Qt.DisplayRole, _rss)
    if old_item.text(10) != _cputime:
        old_item.setData(10,Qt.DisplayRole, _cputime)
    if old_item.text(11) != _cputime:
        old_item.setData(11,Qt.DisplayRole, _threads)
    if old_item.text(12) != _read_bytes and _read_bytes != 0:
        old_item.setData(12,Qt.DisplayRole, _read_bytes)
    if old_item.text(13) != _write_bytes and _write_bytes != 0:
        old_item.setData(13,Qt.DisplayRole, _write_bytes)
    if _memory_percent != 0 and old_item.text(6) != _memory_percent:
        old_item.setData(6,Qt.DisplayRole, _memory_percent)
    if _cpu_percent != 0 and old_item.text(5) != _cpu_percent:
        old_item.setData(5,Qt.DisplayRole,_cpu_percent)
    if old_item.text(3) != _status:
        old_item.setData(3,Qt.DisplayRole,_status)
    if old_item.text(15) != _working_dir:
        old_item.setText(15,_working_dir)
def realtime_values(self,proc):
    _pid = proc.pid
    _name = proc.name
    try:
        _status = str(proc.status)
    except (AttributeError, AccessDenied, NoSuchProcess):
        _status =""
    try:
        _cpu_percent = round(proc.get_cpu_percent(interval=0.0),0)

            except (AttributeError, AccessDenied, NoSuchProcess):
        _cpu_percent = ""
    else:
```

```
            if _cpu_percent == 0:
                _cpu_percent = ""
        try:
            _memory_percent = round(proc.get_memory_percent(),1)
        except (AttributeError, AccessDenied, NoSuchProcess):
            _memory_percent = ""
        else:
            if _memory_percent == 0:
                _memory_percent = ""
        try:
            _memory = proc.get_memory_info()
        except (AttributeError, AccessDenied, NoSuchProcess):
            _memory = 0
            _rss = ""
            _vsz = ""
        else:
            _rss = round(_memory.rss / 1000000.0,1)
            _vsz = round(_memory.vms / 1000000.0,1)
            if _rss == 0:
                _rss = ""
            if _vsz == 0:
                _vsz = ""
        try:
            _cputime = time.strftime("%M:%S", time.localtime(sum(proc.get_cpu_times())))
        except (AttributeError, AccessDenied, NoSuchProcess):
            _cputime = ""
        try:
            _threads = proc.get_num_threads()
        except (AttributeError, AccessDenied, NoSuchProcess):
            _threads = ""
        try:
            _read_bytes = proc.get_io_counters()[2]
            _write_bytes = proc.get_io_counters()[3]
        except (AttributeError,AccessDenied, NoSuchProcess):
            _read_bytes = ""
            _write_bytes = ""
        try:
            _working_dir = proc.getcwd()
        except(AttributeError,AccessDenied, NoSuchProcess):
            _working_dir = QtGui.QApplication.translate("MainWindow","AccessDenied",
                            None, QtGui.QApplication.UnicodeUTF8)

        return [_status,_cpu_percent,_pid,_memory_percent,
                _rss,_vsz,_cputime,_threads,_read_bytes,_write_bytes,_working_dir]
    def stabletime_values(self,proc,today_day):
        accesdenied = QtGui.QApplication.translate("MainWindow","AccessDenied",
```

```
                    None, QtGui.QApplication.UnicodeUTF8)
try:
    _name = proc.name
except (AttributeError, AccessDenied, NoSuchProcess):
    _name = accesdenied
try:
    _username = proc.username
except (AttributeError, AccessDenied, NoSuchProcess):
    _username = ""
try:
    _command = proc.cmdline
except (AttributeError, AccessDenied, NoSuchProcess):
    _command = ""
else:
    if os.name == 'nt' and '\\' in _username:
        _username = _username.split('\\')[1]
try:
    _parent_name = proc.parent.name
    _parent_pid = proc.parent.pid
except (AttributeError, AccessDenied, NoSuchProcess):
    _parent_name = accesdenied
    _parent_pid = "00000"

try:
    _start = datetime.datetime.fromtimestamp(proc.create_time)
except (AttributeError, AccessDenied, NoSuchProcess):
    _start = ""
else:
    if _start.date() == today_day:
        _start = _fromUtf8(_start.strftime("%H:%M"))
    else:
        _start = _fromUtf8(_start.strftime("%b%d"))
try:
    _niceness = proc.nice
except (AttributeError, AccessDenied, NoSuchProcess):
    _niceness = ""

return [_name,_username,_command,
        _parent_name,_parent_pid,_start,_niceness]
```

```
    def table_style (self):
        global TABLE_INTERVAL
        firstloop = True
        proc_list = [p for p in psutil.process_iter()]
        selfprocid =  psutil.Process(os.getpid()).pid
        while True:
            today_day = datetime.date.today()
            for proc in proc_list:
                try:
                    rtvals = self.realtime_values(proc)
                except (AccessDenied,NoSuchProcess):
                    proc_list = self.clear_old_proc(proc_list,proc)
                else:
                    try:
                        str_pid = str(rtvals[2])
                        old_item=self.ui.treeWidget.findItems(str_pid,Qt.MatchExactly,1)[0]
                    except (IndexError):
                        try:
                            stvals = self.stabletime_values(proc,today_day)
                        except (AccessDenied,NoSuchProcess):
                            proc_list = self.clear_old_proc(proc_list,proc)
                        else:
                            if selfprocid == proc.pid:
                                proc_list = self.clear_old_proc(proc_list,proc)
                            else:
                                item_tree=self.new_root_treeitem(stvals[0],rtvals[2],stvals
[1],
                                                 stvals[6],stvals[4],stvals[3],stvals[2],stv
als[5])
                                self.ui.treeWidget.addTopLevelItem(item_tree)
                    else:
                        self.old_treeitem(old_item,rtvals[5],rtvals[4],rtvals[6],rtvals[7],
                                          rtvals[8],rtvals[9],rtvals[3],rtvals[1],rtvals[0]
,rtvals[10])
            if firstloop == True:
                firstloop = False
            else:
                time.sleep(TABLE_INTERVAL)

            self.ui.treeWidget.sortItems(self.ui.treeWidget.header().sortIndicatorSection()
,
                                         self.ui.treeWidget.header().sortIndicatorOrder())
            proc_list = self.add_new_proc(proc_list)
```

Π3.4 Παράδειγμα παράθυρου διαλόγου, Βοήθεια

```
from PyQt4 import QtCore, QtGui
try:
   _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
   _fromUtf8 = lambda s: s
class Ui_Psymon_Help(object):
   def setupUi(self, Psymon_Help):
     Psymon_Help.setObjectName(_fromUtf8("Psymon_Help"))
     Psymon_Help.resize(680, 450)
     Psymon_Help.setWindowTitle(QtGui.QApplication.translate("Psymon_Help", "Psymon Help",
None, QtGui.QApplication.UnicodeUTF8))
     self.gridLayout_set = QtGui.QGridLayout(Psymon_Help)
     self.gridLayout_set.setObjectName(_fromUtf8("gridLayout_set"))
     self.label = QtGui.QTextEdit(Psymon_Help)
     self.label.setReadOnly(True)
     self.label.setAutoFillBackground(True)
     self.label.setText(QtGui.QApplication.translate("Psymon_Help","<h4>Introduction</h4>Py
thon System Monitor (<i>Psymon</i>)<br> is a cross-platform, task and performance
monitor.<br><br>Fearures:<br>*Global process monitoring<br>*System load history
(cpu,memory,netwok and disks)<br>*Disks informations<br>*Network connections<br>*Detailed
informations and cpu, memory percentage history per process",None,
QtGui.QApplication.UnicodeUTF8))
     self.label.append(QtGui.QApplication.translate("Psymon_Help","<h4>Getting
started</h4>The Psymon main window consists of a menu bar, the work space and a status
bar.<br>The worksheets of the Psymon are: <b><i>Process Table, Cpu & Memory Info, Disks
Info, Network Info</b></i> and <b><i>Detailed Process Info</b></i>. The <i>Process
Table</i> lists the running processes with many informations about them. The <i>Cpu &
Memory Info</i> worksheet shows graphs of system utilization: Cpu history and Memory-Swap
history. The <i>Disks Info</i> worksheet shows a graph and a table of disks utilization and
informations. The <i>Network Info</i> worksheet shows a graph and a table of network
interfaces utilization and informations.Also shows a table with the global network
connections.The <i>Detailed Process Info</i> worksheet shows a graph and three tables with
informations of one selected process from the <i>Process Table</i> worksheet.",None,
QtGui.QApplication.UnicodeUTF8))
```

```
        self.label.append(QtGui.QApplication.translate("Psymon_Help","<h4>Process
Table</h4>The Process Table gives you a list of processes on your system. The list can be
sorted by each column. Just press the left mouse button at the head of the column. If you
have selected one process you can press the End Process button to terminate it. If this
applications still have unsaved data this data will be lost. So use this button with care.
The <i>Quick Search...</i> filter which processes are shown by the text given here. The
text can be a partial string match of the Name, Pid or Username of the process. The
compobox in the right of the <i>Quick Search...</i> can used to change the process table in
<i>Table View</i> or <i>Tree View</i>. Also if you perform a double left click on a process
you moved to the <i>Detailed Process Info</i> worksheet where you can see detailed
informations about this process.",None, QtGui.QApplication.UnicodeUTF8))
        self.label.append(QtGui.QApplication.translate("Psymon_Help","<h4>Graphs</h4>The
graphs of the Psymon on the axis Y shows percentage or KiloBytes (KB) and on the axis X the
time. On the right side of every graph-box there are buttons that can be used to show or
hide graphs.",None, QtGui.QApplication.UnicodeUTF8))
        self.label.append(QtGui.QApplication.translate("Psymon_Help","<br><i>Note:</i> The
multiples of a byte in Psymon are decimals and not binaries. For example, a KiloByte is
1000 bytes and not 1024 bytes, which is a KibiByte.",None, QtGui.QApplication.UnicodeUTF8))
        self.gridLayout_set.addWidget(self.label, 0, 0, 0, 0)
```

Π3.5 Παράδειγμα δημιουργίας γραφήματος, ΚΜΕ

```
import psutil
import os
from PyQt4 import Qt,QtCore,QtGui
import PyQt4.Qwt5 as Qwt
from PyQt4.Qwt5.anynumpy import *
from PyQt4.QtCore import QString

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    _fromUtf8 = lambda s: s

class CpuStat:
    User = 0
    Nice = 1
    System = 2
    Idle = 3
    counter = 0
    def __init__(self):
        self.procValues = self.__lookup()

    def statistic(self):
        values = self.__lookup()
        userDelta = 0.0
        for i in [CpuStat.User, CpuStat.Nice]:
            userDelta += (values[i] - self.procValues[i])
        systemDelta = values[CpuStat.System] - self.procValues[CpuStat.System]
        totalDelta = 0.0
        for i in range(len(self.procValues)):
            totalDelta += (values[i] - self.procValues[i])
        self.procValues = values
        return 100.0*userDelta/totalDelta, 100.0*systemDelta/totalDelta

    def nowTime(self):
        result = Qt.QTime(0, 0)
        return result

    def __lookup(self):
        mycputimes=psutil.cpu_times(percpu=False)
        if os.name == "nt":
            tmp = [mycputimes.user,0.0,mycputimes.system,mycputimes.idle,0.0,0.0,0.0]
        else:
```

```
            tmp =
[mycputimes.user,mycputimes.nice,mycputimes.system,mycputimes.idle,mycputimes.iowait,mycput
imes.irq,mycputimes.softirq]
        return tmp

class CpuPieMarker(Qwt.QwtPlotMarker):
    def __init__(self, *args):
        Qwt.QwtPlotMarker.__init__(self, *args)
        self.setZ(1000.0)
        self.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased, True)

    def rtti(self):
        return Qwt.QwtPlotItem.Rtti_PlotUserItem

    def draw(self, painter, xMap, yMap, rect):
        margin = 5
        pieRect = Qt.QRect()
        pieRect.setX(rect.x() + margin)
        pieRect.setY(rect.y() + margin)
        pieRect.setHeight(yMap.transform(80.0))
        pieRect.setWidth(pieRect.height())

        angle = 3*5760/4
        for key in ["User", "System", "Idle"]:
            curve = self.plot().cpuPlotCurve(key)
            if curve.dataSize():
                value = int(5760*curve.y(0)/100.0)
                painter.save()
                painter.setBrush(Qt.QBrush(curve.pen().color(),
                                           Qt.Qt.SolidPattern))
                painter.drawPie(pieRect, -angle, -value)
                painter.restore()
                angle += value

class TimeScaleDraw(Qwt.QwtScaleDraw):
    def __init__(self, baseTime, *args):
        Qwt.QwtScaleDraw.__init__(self, *args)
        baseTime = baseTime.currentTime()
        self.baseTime = baseTime.addSecs(-59)

    def label(self, value):
        nowTime = self.baseTime.addSecs(int(value))
        return Qwt.QwtText(nowTime.toString())

class Background(Qwt.QwtPlotItem):
    def __init__(self):
```

```
        Qwt.QwtPlotItem.__init__(self)
        self.setZ(0.0)
    def rtti(self):
        return Qwt.QwtPlotItem.Rtti_PlotUserItem
    def draw(self, painter, xMap, yMap, rect):
        c = Qt.QColor(Qt.Qt.red)
        r = Qt.QRect(rect)

        for i in range(100, 0, -5):
            r.setBottom(yMap.transform(i - 5))
            r.setTop(yMap.transform(i))
            c.setAlpha(100)
            painter.fillRect(r, c)
            c = c.light(105)

class CpuCurve(Qwt.QwtPlotCurve):
    def __init__(self, *args):
        Qwt.QwtPlotCurve.__init__(self, *args)
        self.setRenderHint(Qwt.QwtPlotItem.RenderAntialiased)

    def setColor(self, color):
        c = Qt.QColor(color)
        c.setAlpha(180)
        self.setPen(c)
        c.setAlpha(80)
        self.setBrush(c)

HISTORY = 60
class CpuPlot(Qwt.QwtPlot):
    def __init__(self, *args):
        Qwt.QwtPlot.__init__(self, *args)
        self.curves = {}
        self.data = {}
        self.timeData = 1.0 * arange(HISTORY-1, -1, -1)
        self.cpuStat = CpuStat()
        self.setAutoReplot(False)

        self.plotLayout().setAlignCanvasToScales(True)
        legend = Qwt.QwtLegend()
        legend.setItemMode(Qwt.QwtLegend.CheckableItem)
        self.insertLegend(legend, Qwt.QwtPlot.RightLegend)
        self.setAxisScaleDraw(
            Qwt.QwtPlot.xBottom, TimeScaleDraw(self.cpuStat.nowTime()))
        self.setAxisScale(Qwt.QwtPlot.xBottom, 0, HISTORY)
        self.setAxisLabelRotation(Qwt.QwtPlot.xBottom, -05.0)
        self.setAxisLabelAlignment(
```

```
    Qwt.QwtPlot.xBottom, Qt.Qt.AlignLeft | Qt.Qt.AlignBottom)
self.yLeft = Qwt.QwtText(QtGui.QApplication.translate("MainWindow","Usage [%]",
                  None, QtGui.QApplication.UnicodeUTF8))
self.setAxisTitle(Qwt.QwtPlot.yLeft, self.yLeft)
self.setAxisScale(Qwt.QwtPlot.yLeft, 0, 100)
self.setMinimumHeight(130)
background = Background()
background.attach(self)
pie = CpuPieMarker()
pie.attach(self)
curve = CpuCurve('System')
curve.setColor(Qt.Qt.green)
curve.attach(self)
self.curves['System'] = curve
self.data['System'] = zeros(HISTORY, Float)
curve = CpuCurve('User')
curve.setColor(Qt.Qt.blue)
curve.setZ(curve.z() - 1.0)
curve.attach(self)
self.curves['User'] = curve
self.data['User'] = zeros(HISTORY, Float)
curve = CpuCurve('Total')
curve.setColor(Qt.Qt.black)
curve.setZ(curve.z() - 2.0)
curve.attach(self)
self.curves['Total'] = curve
self.data['Total'] = zeros(HISTORY, Float)
curve = CpuCurve('Idle')
curve.setColor(Qt.Qt.darkCyan)
curve.setZ(curve.z() - 3.0)
curve.attach(self)
self.curves['Idle'] = curve
self.data['Idle'] = zeros(HISTORY, Float)
self.showCurve(self.curves['System'], True)
self.showCurve(self.curves['User'], True)
self.showCurve(self.curves['Total'], False)
self.showCurve(self.curves['Idle'], False)
self.startTimer(1000)
self.connect(self,
             Qt.SIGNAL('legendChecked(QwtPlotItem*, bool)'),
             self.showCurve)
self.replot()
```

```
def timerEvent(self, e):
    for data in self.data.values():
        data[1:] = data[0:-1]
    self.data["User"][0], self.data["System"][0] = self.cpuStat.statistic()
    self.data["Total"][0] = self.data["User"][0] + self.data["System"][0]
    self.data["Idle"][0] = 100.0 - self.data["Total"][0]
    self.timeData += 1.0
    self.setAxisScale(
        Qwt.QwtPlot.xBottom, self.timeData[-1], self.timeData[0])
    for key in self.curves.keys():
        self.curves[key].setData(self.timeData, self.data[key])
    self.replot()
def showCurve(self, item, on):
    item.setVisible(on)
    widget = self.legend().find(item)
    if isinstance(widget, Qwt.QwtLegendItem):
        widget.setChecked(on)
    self.replot()

def cpuPlotCurve(self, key):
    return self.curves[key]
```

ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Για την χρήση της εφαρμογής χρειάζεται πρώτα η εγκατάσταση της στο σύστημά μας χρησιμοποιώντας το κατάλληλο αρχείο εγκατάστασης για το λειτουργικό μας σύστημα. Τα αρχεία εγκατάστασης της τελευταίας έκδοσης της εφαρμογής είναι διαθέσιμα στην παρακάτω ηλεκτρονική διεύθυνση:

`http://code.google.com/p/psymon/downloads/list`

Η εκκίνηση της εφαρμογής μπορεί να γίνει από το κύριο μενού των εφαρμογών του λειτουργικού μας συστήματος. Για λεπτομέρειες χρήσης της εφαρμογής ανατρέξτε στο *Κεφάλαιο 4* της πτυχιακής εργασίας.

ΙΣΤΟΣΕΛΙΔΑ ΕΦΑΡΜΟΓΗΣ

http://code.google.com/p/psymon/