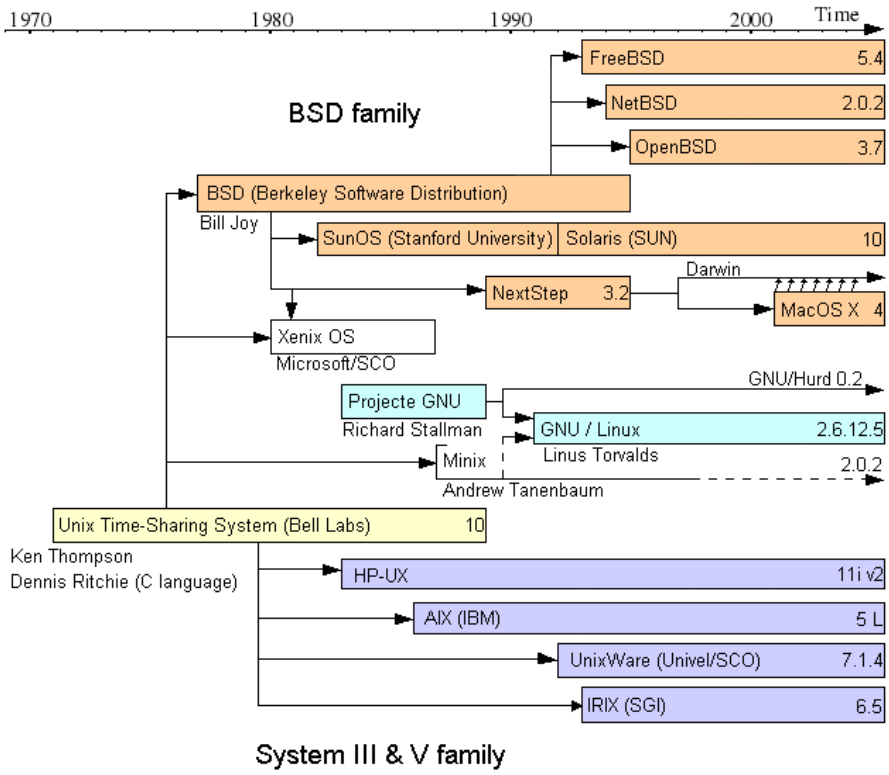
[ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ]



[27.8.2007]

Κούτρας Ιωάννης AM: 4043
Κουτσανδριάς Παναγιώτης AM: 4088

Επιβλέπων Καθηγητής: Μηνάς Δασυγένης

[Κατασκευάστε ένα διαλειτουργικό σύστημα client-server (2 εφαρμογές). Η μία εφαρμογή θα εκτελείται σε λειτουργικό σύστημα Linux ενώ η άλλη σε Windows (σε άλλο PC). Η εφαρμογή client θα συνδέεται στο server και θα στέλνει ένα TEXT μήνυμα το οποίο θα παίρνει η διεργασία server και θα το προσαρτά (append) σε ένα αρχείο. Θα χρησιμοποιείται το πρωτόκολλο TCP.]

Πίνακας Περιεχομένων

ΕΙΣΑΓΩΓΗ

Για την παρούσα εργασία επιλέχτηκε η γλώσσα προγραμματισμού Java. Οι λόγοι για τους οποίους έγινε αυτή η επιλογή αναλύονται συνοπτικά παρακάτω.

Τα χαρακτηριστικά της Java

Ένα από τα βασικά πλεονεκτήματα της Java έναντι των περισσότερων άλλων γλωσσών είναι η ανεξαρτησία του λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε *Java* τρέχουνε ακριβώς το ίδιο σε Windows, Linux, Unix και Macintosh (σύντομα θα τρέχουν και σε Playstation καθώς και σε άλλες παιχνιδομηχανές) χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) αλλά και λειτουργικού συστήματος (Windows, Unix, Linux, Unix, MacOS). Ο λόγος είναι ότι κάθε κεντρική μονάδα επεξεργασίας μπορεί και «καταλαβαίνει» διαφορετικό *assembly* κώδικα. Ο συναρμολογούμενος (*assembly*) κώδικας που τρέχει σε Windows είναι διαφορετικός από αυτόν που τρέχει σε ένα Macintosh. Η λύση δόθηκε με την ανάπτυξη της *Εικονικής Μηχανής* (*Virtual Machine* ή VM ή EM στα ελληνικά).

Γενικά για την Java

Η εικονική μηχανή της Java

Αφού γραφτεί κάποιο πρόγραμμα σε Java τότε γίνεται compile μέσω του Java compiler (javac), εκείνος με την σειρά του δίνει ένα αριθμό από .class αρχεία (=bytecode). Το bytecode είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττιστεί. Όταν προσπαθήσουμε λοιπόν να εκτελέσουμε την εφαρμογή μας το Java Virtual Machine που πρέπει να είναι εγκατεστημένο στο μηχάνημά μας, θα αναλάβει να διαβάσει τα .class αρχεία και να τα μεταφράσει σε γλώσσα και εντολές μηχανής (assembly) που υποστηρίζει το λειτουργικό μας και ο επεξεργαστής μας, έτσι ώστε να εκτελεστεί (να σημειώσουμε εδώ ότι αυτό συμβαίνει με την παραδοσιακή Εικονική Μηχανή (Virtual Machine) . Πιο σύγχρονες εφαρμογές της Εικονικής Μηχανής μπορούν και μεταγλωττίζουν τμήματα bytecode απ' ευθείας σε ιθαγενή κώδικα (native code), χρησιμοποιώντας πολλαπλά νήματα επεξεργασίας με αποτέλεσμα να βελτιώνεται η ταχύτητα. Χωρίς αυτό δε θα ήταν δυνατή η εκτέλεση λογισμικού γραμμένου σε Java. Πρέπει να πούμε ότι το Virtual Machine είναι λογισμικό platform specific δηλαδή για κάθε είδος λειτουργικού και ανάλογης τεχνολογίας επεξεργαστή, υπάρχει διαφορετική έκδοση. Έτσι υπάρχουν διαθέσιμες εκδόσεις του για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδομηχανές κλπ!

Οτιδήποτε θέλει να κάνει ο προγραμματιστής (ή ο χρήστης) γίνεται μέσω της εικονικής μηχανής. Αυτό βοηθάει στο να υπάρχει μεγαλύτερη ασφάλεια στο σύστημα γιατί η εικονική μηχανή είναι υπεύθυνη για την επικοινωνία χρήστη - υπολογιστή. Ο προγραμματιστής δεν μπορεί να γράψει κώδικα ο οποίος θα έχει καταστροφικά αποτελέσματα για τον υπολογιστή γιατί η εικονική μηχανή θα τον ανιχνεύσει και δε θα επιτρέψει να εκτελεστεί. Από την άλλη μεριά ούτε ο χρήστης μπορεί να κατεβάσει «κακό» κώδικα από το δίκτυο και να τον εκτελέσει. Αυτό είναι ιδιαίτερα χρήσιμο για μεγάλα κατανεμημένα συστήματα όπου πολλοί χρήστες χρησιμοποιούν το ίδιο πρόγραμμα συγχρόνως.

Ο συλλέκτης απορριμμάτων (Garbage Collector)

Ακόμα μία ιδέα που βρίσκεται πίσω από τη *Java* είναι η ύπαρξη του συλλέκτη απορριμμάτων (*Garbage Collector*). Συλλογή απορριμμάτων είναι μία κοινή ονομασία που χρησιμοποιείται στον τομέα της πληροφορικής για να δηλώσει την ελευθέρωση τμημάτων μνήμης από δεδομένα που δε χρειάζονται και δε χρησιμοποιούνται άλλο. Αυτή η απελευθέρωση μνήμης στη Java είναι αυτόματη και γίνεται μέσω του συλλέκτη απορριμμάτων. Υπεύθυνη για αυτό είναι και πάλι η εικονική μηχανή η οποία μόλις «καταλάβει» ότι ο σωρός (heap) της μνήμης (στη Java η συντριπτική πλειοψηφία των αντικειμένων αποθηκεύονται στο σωρό σε αντίθεση με τη C++ όπου αποθηκεύονται κυρίως στη στοίβα - stack) κοντεύει να γεμίσει ενεργοποιεί το συλλέκτη απορριμμάτων. Έτσι ο προγραμματιστής δε χρειάζεται να ανησυχεί για το πότε και αν θα ελευθερώσει ένα συγκεκριμένο τμήμα της μνήμης, ούτε και για δείκτες (pointers) που αναφέρονται σε άδειο χώρο μνήμης. Αυτό είναι ιδιαίτερα σημαντικό αν σκεφτούμε ότι ένα μεγάλο ποσοστό κατάρρευσης των προγραμμάτων οφείλονται σε λανθασμένο χειρισμό της μνήμης.

Επιδόσεις

Παρόλο που η εικονική μηχανή προσφέρει όλα αυτά (και όχι μόνο) τα πλεονεκτήματα, η *Java* είναι πιο αργή σε σχέση με άλλες προγραμματιστικές γλώσσες υψηλού επιπέδου (high-level) όπως η C και η C++. Έχει αποδειχτεί ότι η C++ μπορεί να είναι αρκετές φορές γρηγορότερη από τη Java. Ευτυχώς γίνονται φιλότιμες προσπάθειες από τη Sun για τη βελτιστοποίηση της εικονικής μηχανής, ενώ και διάφορες άλλες πραγματοποιήσεις της εικονικής μηχανής υπάρχουν από διάφορες άλλες εταιρίες (όπως IBM) οι οποίες μπορεί σε κάποια σημεία να προσφέρουν καλύτερα και σε κάποια άλλα χειρότερα αποτελέσματα. Επιπλέον με τον ερχομό των JIT (Just In Time) compilers, οι οποίοι μετατρέπουν το bytecode απ' ευθείας σε γλώσσα μηχανής, η διαφορά ταχύτητας από τη C++ έχει μικρύνει κατά πολύ. Οι τελευταίες εκδόσεις του Java Compiler με την χρήση της τεχνολογίας Hot Spot καταφέρνει να παρέχει αξιόλογες επιδόσεις που πλησιάζουν ή και ξεπερνούν σε μερικές περιπτώσεις native κώδικα.

Υποδοχές (Sockets)

Η *υποδοχή* είναι ένα σημείο τέλους της σύνδεσης(endpoint), με δυνατότητα διπλής κατεύθυνσης επικοινωνίας μεταξύ δύο προγραμμάτων που τρέχουν στο δίκτυο. Μια υποδοχή είναι συνδεδεμένη σε ένα port έτσι ώστε το επίπεδο TCP να μπορεί να προσδιορίσει την εφαρμογή για την οποία το στοιχείο προορίζεται να αποσταλεί.

Κανονικά, ένας εξυπηρετητής τρέχει σε έναν συγκεκριμένο υπολογιστή και έχει μια υποδοχή που είναι συνδεδεμένη σε έναν συγκεκριμένο αριθμό θύρας. Ο εξυπηρετητής περιμένει, αναμένοντας στην υποδοχή έναν πελάτη για να υποβάλει ένα αίτημα σύνδεσης.

Στην πλευρά του client: Ο client ξέρει το hostname του υπολογιστή στην οποία ο εξυπηρετητής τρέχει και τον αριθμό της θύρας στον οποίο ο εξυπηρετητής αναμένει Έτσι, ο client κάνει μια αίτηση για σύνδεση. Ο client πρέπει επίσης να προσδιοριστεί στον εξυπηρετητή και έτσι δεσμεύει έναν αριθμό θύρας που θα χρησιμοποιήσει κατά τη διάρκεια αυτής της σύνδεσης. Αυτό ορίζεται συνήθως από το σύστημα.

Εάν όλα πηγαίνουν καλά, ο εξυπηρετητής δέχεται τη σύνδεση. Αφού γίνει η αποδοχή της σύνδεσης, ο εξυπηρετητής παίρνει μια νέα υποδοχή και την συνδέει στην ίδια τοπική θύρα και επίσης θέτει στο σημείο τέλους της σύνδεσης(endpoint) του τη διεύθυνση και τη θύρα του client. Χρειάζεται επίσης και μια νέα υποδοχή έτσι ώστε να μπορεί να συνεχίσει να ακούει στην αρχική υποδοχή για τα αιτήματα σύνδεσης ενώ παράλληλα να ανταποκρίνεται στις ανάγκες του client.

Από την πλευρά του client, εάν η σύνδεση γίνει αποδεκτή, μια υποδοχή δημιουργείται επιτυχώς και ο client μπορεί να χρησιμοποιήσει την υποδοχή για να επικοινωνήσει με τον εξυπηρετητή.

Ο πελάτης και ο κεντρικός υπολογιστής μπορούν τώρα να επικοινωνήσουν με την εγγραφή και την ανάγνωση δεδομένων από τις υποδοχές τους.

Ένα τέλους της σύνδεσης(endpoint) είναι ένας συνδυασμός μιας διεύθυνσης IP και μίας θύρας. Κάθε σύνδεση TCP μπορεί να προσδιοριστεί πλήρως από δύο endpoints της. Με αυτόν τον τρόπο μπορούν να υπάρξουν πολλαπλές συνδέσεις μεταξύ ενός client και ενός εξυπηρετητή.

Το πακέτο `java.net` στην πλατφόρμα της Java παρέχει μια κλάση, την Socket , η οποία εφαρμόζει μια διπλής κατεύθυνσης σύνδεση μεταξύ του προγράμματος της Java και ενός άλλου προγράμματος στο δίκτυο. Η κλάση Socket συνδέεται πάνω σε μια εξαρτώμενη από την εκάστοτε πλατφόρμα εφαρμογή, που κρύβει τις λεπτομέρειες οποιουδήποτε ιδιαίτερου συστήματος από το πρόγραμμα Java. Με τη χρήση της κλάσης `java.net.Socket` αντί της χρήσης εγγενή κώδικα(native code), τα προγράμματα της Java μπορούν να επικοινωνήσουν μέσω δικτύου ανεξάρτητα από το είδος της πλατφόρμας που χρησιμοποιείται κάθε φορά.

Επιπλέον, το πακέτο java.net περιλαμβάνει την κλάση ServerSocket, η οποία δημιουργεί μια υποδοχή που χρησιμοποιείται από τον εξυπηρετητή κατά την αναμονή και την αποδοχή των συνδέσεων με τους clients.

Στη συνέχεια ακολουθεί ένα απλό παράδειγμα που επεξηγεί πώς ένα πρόγραμμα μπορεί να δημιουργήσει μια σύνδεση σε έναν εξυπηρετητή χρησιμοποιώντας την κλάση Socket και έπειτα, πώς ο client μπορεί να στείλει τα δεδομένα και να λάβει τα δεδομένα από τον εξυπηρετητή μέσω της υποδοχής.

Το πρόγραμμα στο παράδειγμα δημιουργεί έναν client, με όνομα `EchoClient`, ο οποίος συνδέεται με έναν echo εξυπηρετητή. Ο εξυπηρετητής τύπου echo λαμβάνει απλά τα στοιχεία από τον client του και τα επιστρέφει πίσω. Ο εξυπηρετητής τύπου echo είναι μια γνωστή υπηρεσία όπου οι clients απικοινωνούν μέσω της θύρας 7.

`Ο client EchoClient` δημιουργεί μια υποδοχή με αυτόν τον τρόπο έτσι ώστε να δημιουργείται μια σύνδεση με τον εξυπηρετητή τύπου echo. Διαβάζει την εισαγωγή δεδομένων από το χρήστη στο τυποποιημένο σύστημα εισαγωγής, και διαβιβάζει έπειτα εκείνο το κείμενο στον εξυπηρετητή τύπου echo με την εγγραφή του κειμένου στην υποδοχή. Ο εξυπηρετητή τύπου echo επιστρέφει την είσοδο μέσω της υποδοχής στον client. Το πρόγραμμα του client διαβάζει και προβάλει τα δεδομένα που επέστρεψαν σε αυτόν μέσω του εξυπηρετητή τύπου echo.

```
import java.io.*;
import java.net.*;

public class EchoClient {
```

```
   public static void main(String[] args) throws IOException {

      Socket echoSocket = null;
      PrintWriter out = null;
      BufferedReader in = null;

      try {
         echoSocket = new Socket("taranis", 7);
         out = new PrintWriter(echoSocket.getOutputStream(), true);
         in = new BufferedReader(new InputStreamReader(
                              echoSocket.getInputStream()));
      } catch (UnknownHostException e) {
         System.err.println("Don't know about host: taranis.");
         System.exit(1);
      } catch (IOException e) {
         System.err.println("Couldn't get I/O for "
                           + "the connection to: taranis.");
         System.exit(1);
      }

            BufferedReader stdIn = new BufferedReader(
                        new InputStreamReader(System.in));
            String userInput;

            while ((userInput = stdIn.readLine()) != null) {
               out.println(userInput);
               System.out.println("echo: " + in.readLine());
            }

            out.close();
            in.close();
            stdIn.close();
            echoSocket.close();
   }
}
```

Ας σημειωθεί ότι ο EchoClient κάνει ανάγνωση και εγγραφή δεδομένων μέσω της υποδοχής, με αποτέλεσμα την αποστολή και την λήψη δεδομένων.

Σχολιάζοντας το ανωτέρω παράδειγμα, οι τρείς δηλώσεις στο block try της μεθόδου main είναι κρίσιμες για την έναρξη της επικοινωνίας. Οι επόμενες τρείς γραμμές δημιουργούν την σύνδεση μέσω της υποδοχής μεταξύ του client και του εξυπηρετητή, καθώς και έναν PrintWriter και έναν BufferedReader στην υποδοχή.

```
echoSocket = new Socket("taranis", 7);
out = new PrintWriter(echoSocket.getOutputStream(), true);
in = new BufferedReader(new InputStreamReader(
                     echoSocket.getInputStream()));
```

Η πρώτη δήλωση δημιουργεί ένα αντικείμενο τύπου υποδοχής με όνομα echoSocket. Ο constructor Socket που χρησιμοποιείται απαιτεί το όνομα και την θύρα του υπολογιστή στον οποίο επιθυμούμε να συνδεθούμε. Η δεύτερη δήλωση λαμβάνει τα δεδομένα από την υποδοχή, ανοίγει έναν PrintWriter. Η τρίτη δήλωση λαμβάνει την είσοδο δεδομένων από την υποδοχή και ανοίγει έναν BufferedReader. Για την αποστολή των δεδομένων από την υποδοχή στον εξυπηρετητή, απλά ο

EchoClient γράφει στον PrintWriter. Όμοια, για την ανάγνωση των δεδομένων αρκεί ο EchoClient να αναγνώσει τα δεδομένα από τον BufferedReader.

Στη συνέχεια ακολουθεί η ανάγνωση των δεδομένων και η αποστολή τους πίσω στον EchoClient. Όταν η redline επιστρέψει, εκτυπώνεται στον EchoClient η πληροφορία των δεδομένων. Ο βρόχος συνεχίζει έως ότου ο χρήστης εισάγει έναν χαρακτήρα τερματισμού. Έτσι ο EchoClient διαβάζει την είσοδο από τον χρήστη, την αποστέλλει στον εξυπηρετητή τύπου echo, την απάντηση από τον εξυπηρετητή, και την εκτυπώνει. Αυτό συνεχίζεται έως ότου ο χρήστης εισάγει έναν χαρακτήρα τερματισμού. Τέλος κλείνουν όλες οι υποδοχές και τερματίζονται οι PrintWriter και BufferedReader.

Νήματα (Threads)

Ένα νήμα είναι μία ελαφριά διεργασία (lightweight process) που χαρακτηρίζεται ως ένα αυτοδύναμο περιβάλλον εκτέλεσης (self-contained execution environment). Κάθε διεργασία περιέχει ένα ή περισσότερα νήματα εκτέλεσης (threads of execution). Όλα τα νήματα μίας διεργασίας μοιράζονται από κοινού τους πόρους της διεργασίας, μοιράζονται δηλαδή πρωτίστως ένα κοινό χώρο στη μνήμη. Όταν θέλουμε να υλοποιήσουμε μία εφαρμογή η οποία να μπορεί να εκτελεί ταυτόχρονα περισσότερες από μία εργασίες είναι απαραίτητο να χρησιμοποιήσουμε νήματα. Κάτι τέτοιο είναι ιδιαίτερα αναγκαίο σε διαδραστικές εφαρμογές (interactive applications) όπου χρειάζεται συνεχής αλληλεπίδραση με το χρήστη καθώς η εφαρμογή εκτελεί χρονοβόρες εργασίες στο παρασκήνιο. Σε αντίθεση με τις διεργασίες, τα νήματα μπορούν να δημιουργηθούν με μικρότερη επιβάρυνση (overhead) και συνεπώς μπορούν να υποστηρίξουν με πιο αποδοτικό τρόπο την ταυτόχρονη διεκπεραίωση αυτών των εργασιών. Λόγω του ότι ενεργούν πάνω σε ένα κοινό χώρο στη μνήμη, τα νήματα μπορούν να επικοινωνούν μεταξύ τους πιο γρήγορα από ότι οι διεργασίες, ωστόσο αυτό εγκυμονεί και κάποιους κινδύνους του οποίους θα εξετάσουμε στη συνέχεια.

Η κάθε διεργασία αποτελείται από πολλά νήματα τα οποία εκτελούνται κατά τη διάρκεια των χρονομεριδίων που παραχωρούνται στη διεργασία οποία ανήκουν. Σε μία εφαρμογή, όλες οι διεργασίες χρόνο-προγραμματίζονται από το λειτουργικό σύστηνα Όλα τα σύγχρονα λειτουργικά συστήματα χρησιμοποιούν χρονομερισμό – επίσης γνωστό ως υπέρτερη πολυεργεία (preemptive multitasking). Το ΛΣ μπορεί ανά πάσα στιγμή να θέσει σε αναμονή κάποια διεργασία και να παραχωρήσει τον επεξεργαστή σε κάποια άλλη διεργασία. Σε μία πολυνηματική εφαρμογή, ο προγραμματιστής είναι υπεύθυνος για τον χρονοπρογραμματισμό των νημάτων μίας διεργασίας. Οι προσεγγίσεις που μπορούν να ακολουθηθούν είναι είτε η συνεργατική νηματοποίηση (cooperative threading), όπου το κάθε νήμα παραχωρεί τον επεξεργαστή στα νήματα οικειοθελώς, είτε η υπέρτερη νηματοποίηση (preemptive threading) όπου περιοδικά το ΛΣ αφαιρεί τον επεξεργαστή από ένα νήμα και τον παραχωρεί σε κάποιο άλλο (χρονομερισμός). Όταν κάποιο νήμα βρίσκεται σε αναμονή, μπορεί να ανασταλεί η εκτέλεσή του μέχρις ότου καταφθάσει το γεγονός που αναμένει. Με αυτό τον τρόπο αυξάνεται η απόδοση της πολυνηματικής εφαρμογής μας αφού κάποιο άλλο νήμα που δε βρίσκεται σε αναμονή μπορεί να εκτελεστεί στη θέση του προηγούμενου.

Swing

Το Swing είναι ένα σετ εργαλείων για GUI σε Java. Περιέχει GUI widgets, καθώς και text-boxes, κουμπιά, split-panes, και πίνακες. Τα widgets σε Swing προσφέρουν πιο ολοκληρωμένα components για GUI από αυτά που προσέφερε το σετ εργαλείων AWT. Από την στιγμή που είναι γραμμένα σε Java, τρέχουν όμοια σε όλες τις πλατφόρμες, γεγονός που δεν συνέβαινε με το AWT, το οποίο εξαρτιόταν άμεσα από το παραθυρικό σύστημα της εκάστοτε πλατφόρμας. Το Swing υποστηρίζει pluggable γραφικά εμφάνισης, χωρίς να χρησιμοποιεί τις εγγενείς πηγές της πλατφόρμας, αλλά με χρήση εξομοίωσης αυτών. Αυτό σημαίνει ότι μπορούμε να έχουμε οποιοδήποτε υποστηριζόμενο σετ γραφικών εμφάνισης σε οποιαδήποτε πλατφόρμα. Το μειονέκτημα των components με μικρές απαιτήσεις για μνήμη είναι η μικρότερη ταχύτητα εκτέλεσης. Το πλεονέκτημα είναι η καθολική συμπεριφορά σε όλες τις πλατφόρμες.

Σύνοψη

Η επιλογή της Java έγινε αρχικά για τη δυνατότητα χρήση του ίδιου κώδικα τόσο σε περιβάλλον Windows, όπου θα πρέπει να λειτουργεί ο server, όσο και σε περιβάλλον Linux, όπου θα πρέπει να λειτουργεί ο client.

Παράλληλα όμως, ένα πλήθος χαρακτηριστικών και βιβλιοθηκών της Java μάς επέτρεψαν να αξιοποιήσουμε κώδικα που ήδη υπάρχει και να δημιουργήσουμε αποδοτικά προγράμματα.

Για παράδειγμα, οι κλάσεις ServerSocket και Socket κάλυψαν όλες τις ανάγκες των προγραμμάτων στο κομμάτι της δικτυακής επικοινωνίας, χωρίς να χρειαστεί να επικοινωνήσουμε άμεσα με το λειτουργικό σύστημα.

Πολύ σημαντική ήταν και η συνεισφορά των ρευμάτων δεδομένων (data streams), χάρη στα οποία γινόταν η συλλογή μηνυμάτων εισόδου/εξόδου των sockets, καθώς και η προσάρτηση μηνυμάτων των clients σε ένα αρχείο του συστήματος.

Τέλος, τα νήματα στην Java βοήθησαν εξίσου στα προγράμματα, ειδικά στο πρόγραμμα του server, όπου κάθε client έχει το δικό του, ανεξάρτητο νήμα μέσω του οποίου ο server δέχεται τα μηνύματα του client.

Ακόμη και το παραθυρικό περιβάλλον (Graphical User Interface) έχει το δικό του νήμα. Με αυτόν τον τρόπο, οι εφαρμογές δεν «κολλούσαν» κάθε φορά που περίμεναν δεδομένα από τα sockets.

ΤΟ ΠΡΟΓΡΑΜΜΑ TCPSERVER

Στην κλάση TCPServer βρίσκονται όλα τα στοιχεία που συνθέτουν τον server της εργασίας. Πρόκειται κυρίως για ένα JFrame με τα κατάλληλα components και κάποιες ενέργειες, τα οποία συνιστούν το GUI, και τέλος δύο κλάσεις threads, την TCPServerListenerThread και TCPServerThread. Η πρώτη αναλαμβάνει να δημιουργήσει ένα listener, ο οποίος σε συγκεκριμένο port του server θα ακούει για καινούριες συνδέσεις και η δεύτερη καλείται κάθε φορά που ένας νέος client συνδέεται στο server.

Περισσότερες λεπτομέρειες του κώδικα δίνονται παρακάτω:

Start Listening

Μόλις πατηθεί το κουμπί «Start Listening», εκτός από κάποιες μικρο-αλλαγές στο GUI, ξεκινάει η δημιουργία του thread του listener. Η διαδικασία αυτή είναι λίγο παραπάνω σύνθετη από την τυπική δημιουργία και εκτέλεση ενός thread, εξαιτίας της υλοποίησης ServerSocket από τον κώδικα. Έτσι, εκτός από τον constructor της κλάσης και τη μέθοδο start(), καλείται και η μέθοδος initialize() που δημιουργεί το ServerSocket. Παρακάτω φαίνονται αναλυτικά οι σχετικές μέθοδοι:

```
TCPServerListenerThread(int port) {
   this.port = port;
   clientid = 1;
}

public void initialize() throws IOException {
   serverSocket = new ServerSocket(port);
}

public void run() {
   try {
      while(!isInterrupted()) {
         new TCPServerThread(serverSocket.accept(), clientid).start();
         clientid++;
      }
   } catch (IOException e) {
      // accept() failed or interrupted...
   }
}
```

Η μεταβλητή clientid είναι ένας μετρητής για τους clients που συνδέονται ή συνδέθηκαν στον server κατά τη διάρκεια του listening.

Θα μπορούσε κανείς να προτείνει την ενσωμάτωση των εντολών της initialize() μέσα στον constructor της κλάσης, αλλά τότε θα ήμασταν αναγκασμένοι να δηλώσουμε ότι ο constructor μπορεί να πετάξει εξαίρεση (exception) πράγμα που κάνει πολύ πιο σύνθετο τον constructor.

Τέλος, στη μέθοδο run() βλέπουμε τις εντολές που εκτελούνται στο επιπλέον thread. Όπως φαίνεται, λοιπόν, μέσα σε ένα σχεδόν ατέρμονο βρόχο (η boolean μεταβλητή isInterrupted() του thread γίνεται true μόνο όταν σταματήσουμε χειροκίνητα το thread – μέσω του Stop Listening) περιμένει ο server νέες συνδέσεις.

Η εξαίρεση είναι απαραίτητη, καθώς κατά την εκτέλεση του interrupt του thread είναι σχεδόν σίγουρο ότι το thread θα περιμένει νέα σύνδεση (η μέθοδος accept() του ServerSocket μπλοκάρει τη συνέχεια του κώδικα).

Stop Listening

Η λειτουργία του κουμπιού «Stop Listening» είναι αντίστοιχη του Start Listening. Καλώντας τη μέθοδο stoplistening() του listener thread εκτελείται ο παρακάτω κώδικας:

```
public void stoplistening() {
    interrupt();
    try {
        serverSocket.close();
    } catch (IOException e) {
        String message = "Could not stop listening on port: " + port;
        JOptionPane.showMessageDialog(new JFrame(), message, "Dialog",
                JOptionPane.ERROR_MESSAGE);
    }
```

Δηλαδή, πρώτα τερματίζεται το thread για να μην υπάρχουν νέες συνδέσεις και έπειτα γίνεται προσπάθεια να κλείσει το ServerSocket.

TCPServerThread

Το TCPServerThread είναι ένα πιο τυπικό thread σε σύγκριση με αυτό του listener:

```
class TCPServerThread extends Thread {
    private Socket socket = null;
    private int id;
    public TCPServerThread(Socket socket, int id) {
        super("TCPServerThread");
        this.socket = socket;
        this.id = id;
    }
```

```
public void run() {
    try {
        PrintWriter out = new PrintWriter(socket.getOutputStream(),
                true);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(
                socket.getInputStream()));
        String inputLine;
        Runnable newClientMsg = new Runnable() {
            public void run() {
                tConsole.append("\nClient " + id +
                        " has been connected.");
            }
        };
        SwingUtilities.invokeLater(newClientMsg);
        while ((inputLine = in.readLine()) != null) {
            if (inputLine.equals("_DISCONNECT")) {
                final String dismsg = "\nClient " + id +
                        " disconnected";
                Runnable doWorkRunnable1 = new Runnable() {
                    public void run() {
                        tConsole.append(dismsg);
                    }
                };
                SwingUtilities.invokeLater(doWorkRunnable1);
                try {
                    filewriter.newLine();
                    filewriter.write(dismsg);
                    filewriter.flush();
                } catch (IOException e) {
                    String message = "Could not append to file";
```

```
                    JOptionPane.showMessageDialog(new JFrame(), message,
                            "Dialog", JOptionPane.ERROR_MESSAGE);
                }
                break;
            }
            final String output = "\nClient " + id + ": " + inputLine;
            Runnable doWorkRunnable = new Runnable() {
                public void run() {
                    tConsole.append(output);
                }
            };
            SwingUtilities.invokeLater(doWorkRunnable);
            try {
                filewriter.newLine();
                filewriter.write(output);
                filewriter.flush();
            } catch (IOException e) {
                String message = "Could not append to file";
                JOptionPane.showMessageDialog(new JFrame(), message,
                        "Dialog", JOptionPane.ERROR_MESSAGE);
            }
        }
        out.close();
        in.close();
        socket.close();
    } catch (IOException e) {
        // Possibly a connection reset
    }
  }
}
```

Όπως φαίνεται, το μεγαλύτερο μέρος του κώδικα, αφιερώνεται στις εντολές κατά την εκτέλεση του thread και όχι κατά τη δημιουργία του.

Οι εντολές που κυριαρχούν είναι αυτές που ασχολούνται με data streams (είτε από τα sockets, είτε για την εγγραφή του αρχείου καταγραφής), πάντα μαζί με τις κατάλληλες εξαιρέσεις και τα μηνύματα λάθους.

Μόλις εδραιωθεί η σύνδεση ξεκινάει ένας ατέρμονος βρόχος στο νήμα, όπου το νήμα περιμένει τον client να του στείλει κάτι. Ο βρόχος αυτός σταματάει είτε μέσω interrupt του νήματος, είτε μέσω της αποστολής του μηνύματος «_DISCONNECT» από τον client.

JFileChooser

Τέλος, αξίζει να αναφέρουμε τον JFileChooser, με τον οποίο μπορεί ο χρήστης να αλλάξει το αρχείο καταγραφής. Πατώντας το κουμπί «Append to file:» παρουσιάζεται ένας διάλογος με τον οποίο μπορεί κανείς να επιλέξει σε ποιο αρχείο θα καταγράφεται η κονσόλα.

```
private void bAppendActionPerformed(java.awt.event.ActionEvent evt)

    JFileChooser fc = new JFileChooser();

    fc.showOpenDialog(this);

    File file = fc.getSelectedFile();

    tFilename.setText(file.getAbsolutePath());

}
```

Ο παραπάνω κώδικας είναι πολύ απλός: Με σκοπό τη λιτότητα του προγράμματος, δεν γίνεται ιδιαίτερος έλεγχος σφαλμάτων στο σημείο αυτό, οπότε θα πρέπει να προσέξει κανείς ότι το αρχείο που θα επιλέξει είναι προσβάσιμο για το λογαριασμό του. Διαφορετικά, ενδεχομένως η εκκίνηση του listener να μην είναι επιτυχής.

ΤΟ ΠΡΟΓΡΑΜΜΑ TCPCLIENT

Το πρόγραμμα TCPClient είναι απλούστερο του προγράμματος για server, αλλά στηρίζεται στην ίδια φιλοσοφία: Τη σύνδεση με τον server και την επικοινωνία τις αναλαμβάνει ένα επιπλέον thread, το TCPClientThread.

Για λόγους συντομίας, παρακάτω θα παρουσιαστεί μόνο η αρχικοποίηση του TCPClientThread και οι μέθοδοι αποστολής μηνύματος και αποσύνδεσης από τον server, ενώ οι μέθοδοι και οι εντολές που αφορούν το GUI της εφαρμογής θα παραληφθούν.

TCPClientThread.initialize()

```
public void initialize() throws UnknownHostException, IOException {
        socket = new Socket(tAddr.getText(), Integer.parseInt(
              tPort.getText()));
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(
              socket.getInputStream()));
        connected = true;
        Runnable SrvConnMsg = new Runnable() {
           public void run() {
              tConsole.append("\nConnected to server.");
           }
        };
        SwingUtilities.invokeLater(SrvConnMsg);
    }
```

Παρόμοια με την εκκίνηση του TCPServerThread, ένα νέο Socket δημιουργείται κάθε φορά που δημιουργείται αυτό το thread. Έπειτα, καλούνται τα απαραίτητα αντικείμενα με τα οποία χειριζόμαστε τις ροές δεδομένων στο socket.

Αποστολή Μηνύματος και Αποσύνδεση

Η αποστολή μηνύματος και η αποσύνδεση από τον server είναι αρκετά απλές, τώρα που έχουν δημιουργηθεί τα αντικείμενα χειρισμού:

```
    public void sendTxt(String text) {
        out.println(text);
    }

    public void disconnect() throws IOException {
        out.println("_DISCONNECT");
        socket.close();
    }
```

Ουσιαστικά, με μια απλή εκτύπωση στο κατάλληλο PrintWriter, αποστέλλεται το επιθυμητό μήνυμα (για αποσύνδεση αρκεί να σταλθεί το μήνυμα «_DISCONNECT»).

Για τη γενικότερη αποστολή μηνύματος, ο χρήστης θα πρέπει να γράψει το μήνυμά του στο κατάλληλο πεδίο και κατόπιν να πατήσει το πλήκτρο Enter ή το κουμπί Send.

Η δεύτερη μέθοδος είναι αναμενόμενη, οπότε ας ρίξουμε μια ματιά μόνο στην πρώτη:

```
private void tTextKeyPressed(java.awt.event.KeyEvent evt) {
    if(evt.getKeyCode() == KeyEvent.VK_ENTER && bSendMsg.isEnabled()) {
        String msg = tText.getText();
        if(msg != "") {
            tcpclientthread.sendTxt(msg);
            tConsole.append("\n" + msg);
            tText.setText("");
        }
    }
}
```

Δηλαδή ένα KeyEvent ελέγχεται κάθε φορά και αν δείχνει ότι το πλήκτρο Enter έχει πατηθεί και επιτρέπεται η αποστολή μηνύματος, τότε το μήνυμα αποστέλλεται και το πεδίο καθαρίζει.

ΠΡΟΓΡΑΜΜΑ GUICHOOSER

Για τις ανάγκες της άσκησης δημιουργήθηκε ένα ακόμη μικρό πρόγραμμα, το GuiChooser.

Με αυτό το πρόγραμμα ρυθμίζεται η εμφάνιση των παραθύρων και δημιουργούνται όσοι καινούριοι servers και clients επιθυμεί ο χρήστης.

Εμφάνιση των παραθύρων (Look and Feel)

Η βιβλιοθήκη Swing που χρησιμοποιήθηκε έχει το δικό της σχεδιασμό παραθύρων. Προκειμένου όμως τα παράθυρα της να ταιριάζουν με την αισθητική του κάθε λειτουργικού συστήματος, δίνεται η δυνατότητα αλλαγής της εμφάνισης των παραθύρων:

```
try {
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
} catch (UnsupportedLookAndFeelException e) {
    // handle exception
} catch (ClassNotFoundException e) {
    // handle exception
} catch (InstantiationException e) {
```

```
        // handle exception
    } catch (IllegalAccessException e) {
        // handle exception
    }
```

Με την παραπάνω εντολή μέσα στο try-catch block, τα παράθυρα αποκτούν εμφάνιση παραθύρων όμοια με αυτή των εγγενών εφαρμογών σε κάθε λειτουργικό σύστημα.

Εφόσον οι εφαρμογές Server και Client καλούνται από το πρόγραμμα αυτό, τότε και τα παράθυρά τους θα έχουν το ίδιο στυλ με το αρχικό παράθυρο.

Κλήση παραθύρων Server και Client

Η κλήση των παραθύρων Server και Client γίνεται στο κατάλληλο event, όταν πατηθεί το κατάλληλο κουμπί:

```
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new TCPServer().setVisible(true);
    }
```

Προκειμένου να είναι σε ανεξάρτητο thread από το τρέχον παράθυρο, δηλαδή, το αντικείμενο δημιουργείται μέσα σε καινούριο Runnable (κάτι αντίστοιχο του Thread) και δουλεύει σε ανεξάρτητο νήμα.

Αντίστοιχα καλείται και καινούριο παράθυρο Client.

Έξοδος από Server και Client

Αν κανείς προσπαθήσει να κλείσει κάποιο παράθυρο πατώντας το κουμπί κλεισίματος παραθύρου (συνήθως πάνω αριστερά σε ΛΣ Windows), θα διαπιστώσει πως όλες οι εφαρμογές θα κλείσουν!

Αυτό συμβαίνει επειδή με το κουμπί αυτό, δίνεται γενικό σήμα στην εφαρμογή να κλείσει, και εφόσον όλες οι εφαρμογές είναι ουσιαστικά νήματα μίας, όλα θα κλείσουν.

Η Java συνιστά σε τέτοιες περιπτώσεις την εντολή dispose(). Με την εντολή αυτή μπορεί κανείς να αποδεσμεύσει τους πόρους που χρησιμοποιεί ενός παράθυρο πλήρως. Το μόνο που θα πρέπει να γίνει προηγουμένως είναι το παράθυρο αυτό να σταματήσει να είναι ορατό (visible), πράγμα που μπορεί να γίνει εύκολα και αξιόπιστα (είναι ασφαλής η μέθοδος ακόμη και σε περιπτώσεις πολυνηματικής εφαρμογής).

Στα προγράμματα server και client, λοιπόν, θα μπορούσαμε να τροποποιούσαμε το windowClosing event των παραθύρων και να προσθέταμε εκεί πέρα τις κατάλληλες εντολές. Προκειμένου όμως να

φαίνεται η διαφορετική λειτουργία κλεισίματος, επιλέχθηκε η δημιουργία κουμπιών «Quit» σε κάθε εφαρμογή.

Παρακάτω παρατίθεται ο κώδικας εξόδου στο Server:

```
private void bQuitActionPerformed(java.awt.event.ActionEvent evt) {
    if(filewriter != null)
        try {
            filewriter.close();
        } catch (IOException ex) {
            String message = "Could not close the file: " +
                    tFilename.getText();
            JOptionPane.showMessageDialog(new JFrame(), message, "Dialog",
                    JOptionPane.ERROR_MESSAGE);
        }
    this.setVisible(false);
    this.dispose();
}
```

Δηλαδή, εκτός από τις εντολές setVisible(false) και dispose() φροντίζουμε να κλείσουμε τυχόν εναπομείναντες ροές δεδομένων.

Αντίστοιχες ενέργειες γίνονται και στην περίπτωση του κουμπιού «Quit» στον client.

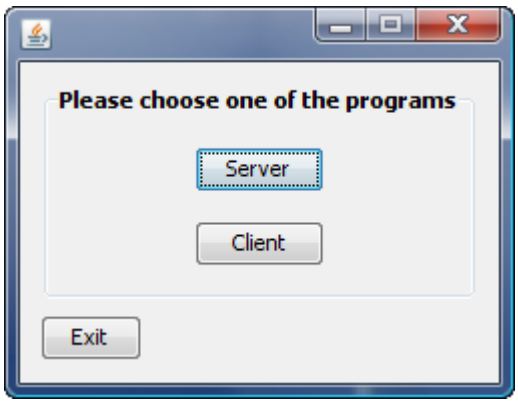
ΠΑΡΑΡΤΗΜΑ - ΕΙΚΟΝΕΣ

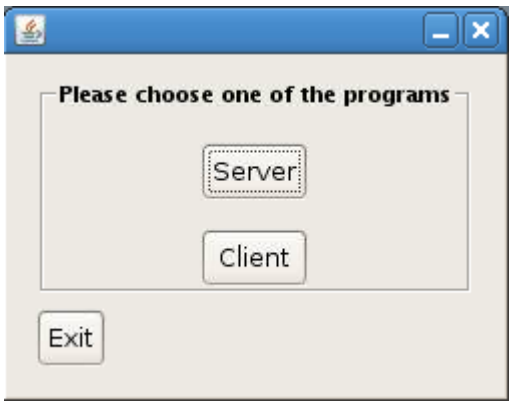
Στο παράρτημα αυτό παρουσιάζονται εικόνες των εφαρμογών σε Windows και Linux, όπως ζητείται στην εργασία. Το λειτουργικό, ωστόσο, δεν αποτελεί εμπόδιο, καθώς και server και client φάνηκαν να λειτουργούν και στα δύο αυτά λειτουργικά συστήματα..

Όλα τα προγράμματα περιέχονται μαζί με τον κώδικά τους στο αρχείο TCPServerClient.jar, ενώ για να τρέξουν απαιτείται να βρίσκεται το αρχείο swing-layout-1.0.jar μέσα στο φάκελο lib της ίδιας διαδρομής με αυτή του TCPServerClient.jar.

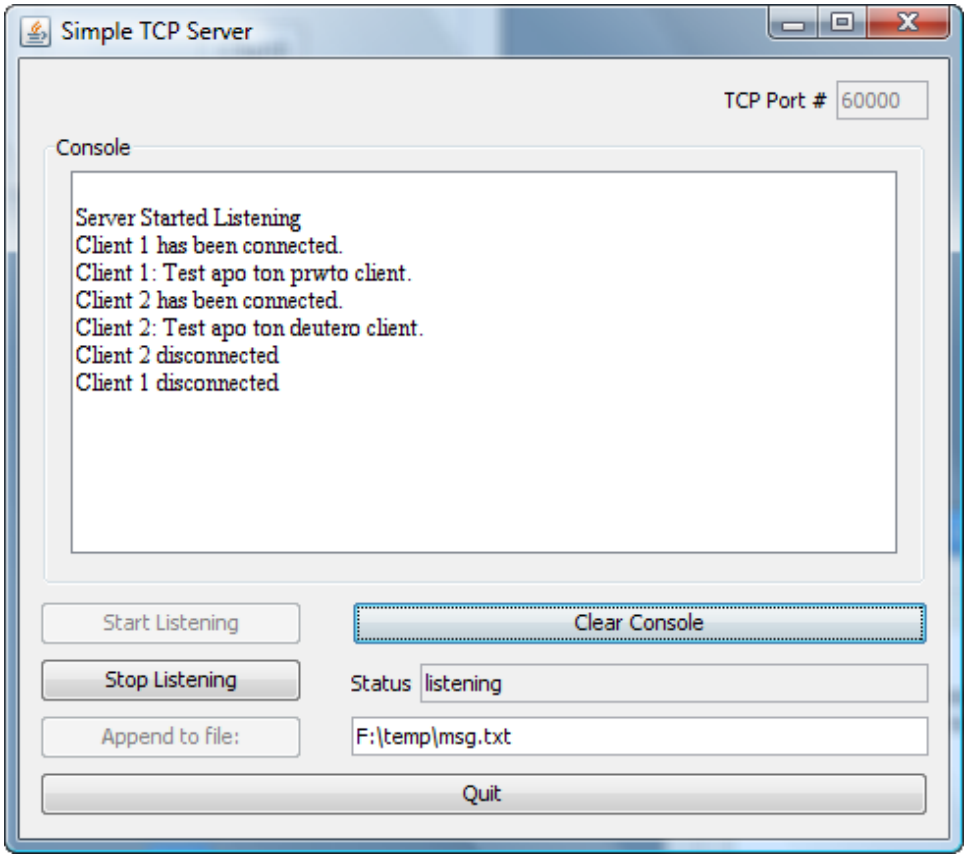
Τα προγράμματα δοκιμάστηκαν σε Java Runtime Environment 1.6.0, οπότε και αυτό προτείνεται, ενώ ο κώδικας αναπτύχθηκε στο περιβάλλον NetBeans.

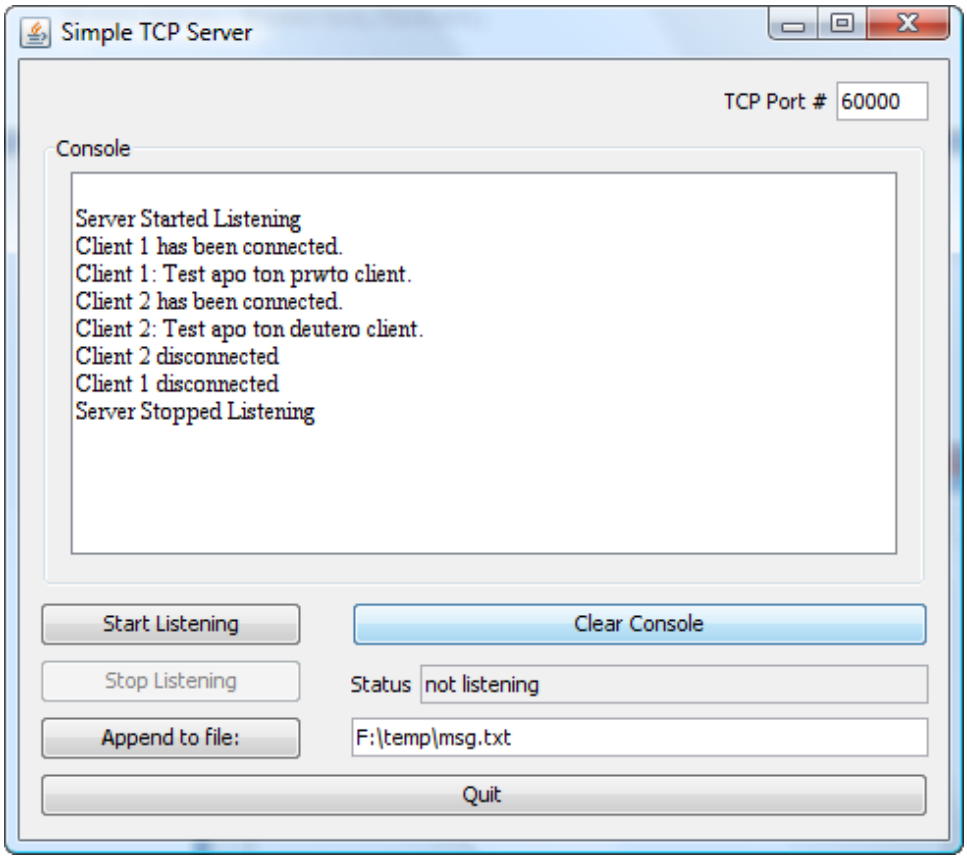
GUIChooser (Windows και Linux)



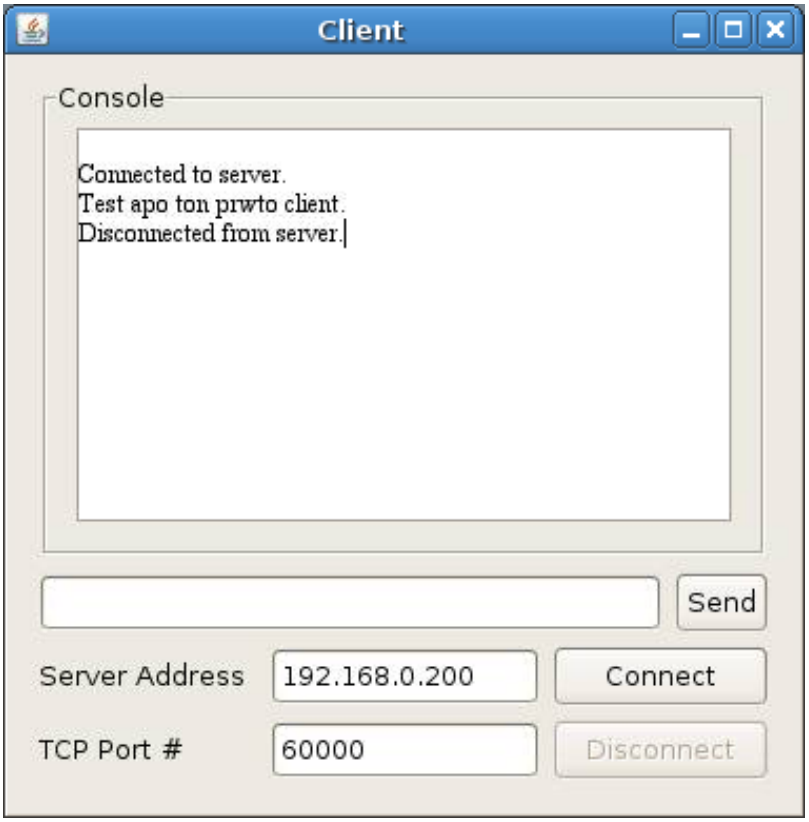


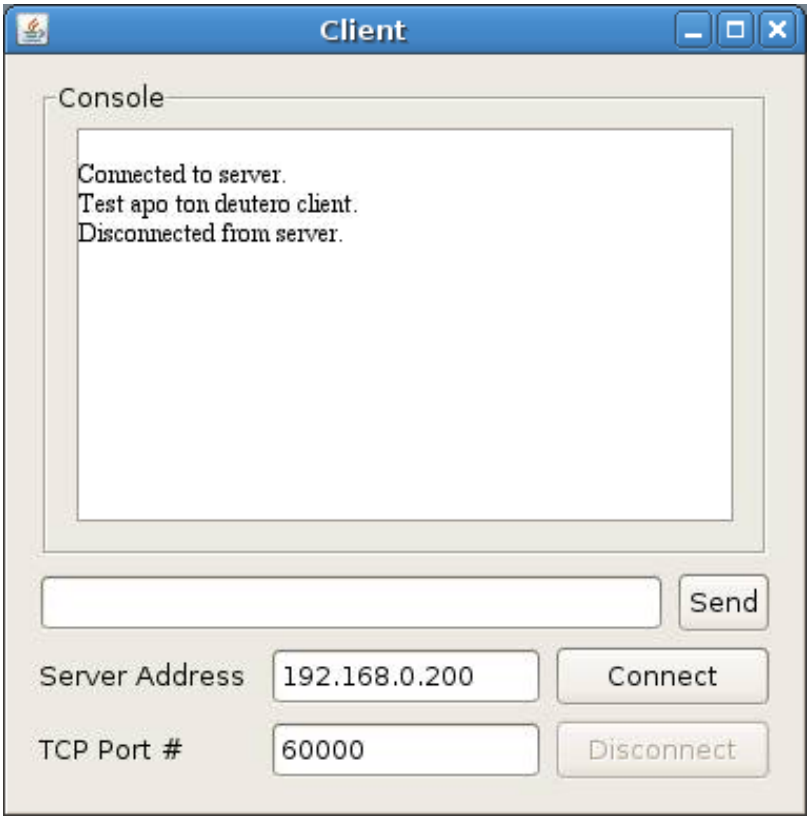
TCPServer (Windows)





TCPClient (Linux)





Αρχείο Καταγραφής των Συνδέσεων και των Μηνυμάτων στον Server