Προγραμματισμός Δικτύων – Ε-01

3η Διάλεξη

Διδάσκων: Νίκος Ντάρμος

<ntarmos@cs.uoi.gr>
[http://www.cs.uoi.gr/~ntarmos/Courses/NetworkProgramming/]

Τμήμα Πληροφορικής
Πανεπιστήμιο Ιωαννίνων

Στο σημερινό μάθημα...

- Η μετάβαση στο IPv6.
- Το πρωτόκολλο DNS.
- Εισαγωγή στα BSD sockets.

Η μετάβαση στο IPv6

Η ανάγκη για το IPv6

- Το IPv4 χρησιμοποιεί διευθύνσεις των 32 bits.
  - $2^{32} \approx 4$ δισεκατομμύρια διευθύνσεις!
  - Οι διευθύνσεις ελέγχονται από την IANA.
  - Classless Inter-Domain Routing μετά το 1993.
- **Τεράστιοι πίνακες δρομολόγησης στους routers.**
  - $\approx$ 80,000 εγγραφές.
- **Ως πότε θα έχουμε διευθύνσεις;**
  - Το 2003 η πρόβλεψη ήταν ως το 2023.
  - Το 2005 η πρόβλεψη ήταν ως το 2009 ή 2010.
  - Σήμερα η πρόβλεψη είναι ως το 2011 το αργότερο!
- Προσωρινή λύση **(;)**: **Network Address Translation (NAT)**.

Η ανάγκη για το IPv6

Network Address Translation

- Υπολογιστές εντός «εταιρικών» δικτύων παίρνουν διευθύνσεις από την τάξη των «ιδιωτικών δικτύων».
- Δρομολογητές στα άκρα των δικτύων μεταφράζουν (NAT) τις (εκατοντάδες) εσωτερικές διευθύνσεις σε λίγες εξωτερικές (μη ιδιωτικές) διευθύνσεις.
- Δυνατότητα για ιεραρχία από τέτοια εσωτερικά δίκτυα.

Προβλήματα

- Καταρρίπτει την συνδεσιμότητα από άκρο σε άκρο.
- Ακατάλληλο για ορισμένα πρωτόκολλα.
- Ακατάλληλο για περιπτώσεις δικτύων με «ειδικές απαιτήσεις».
- Δυσκολία στη διαχείριση και τη συντήρηση.

IPv6: Η απάντηση

Ιστορία

- Η ανάπτυξή του ξεκίνησε το 1992 (RFC 1550).
- Πρώτη ολοκληρωμένη προσπάθεια προδιαγραφών το 1996 (RFC 2460).
- Η υποστήριξη από τα διάφορα Λειτουργικά Συστήματα έρχεται σταδιακά:

  1997: AIX, Tru64, OpenVMS.
  2000: FreeBSD, NetBSD, OpenBSD.
  2001: Cisco IOS, HP-UX.
  2002: Windows XP & Server 2003.
  2003: DragonflyBSD, Mac OS X.
  2005: Linux.
  2007: Windows Vista.

- Τελευταία προσθήκη στα στάνταρντ το ... Μάρτιο του 2009 (RFC 5454).
- Διεισδυτικότητα σήμερα: $\leq 1\%$ (πηγή: Google).

IPv6: Η απάντηση

Προσφέρει:

- Διευθύνσεις 128 bits, με (τουλάχιστον) 65,536 υποδίκτυα (/48) ανά site.
  - Αντίστοιχο του ενός **class A** δικτύου ανά site.
  - **340,282,366,920,938,463,463,374,607,431,768,211,456** δυνατές διευθύνσεις...
    - ... σε σύγκριση με τις **4,294,967,296** δυνατές διευθύνσεις του IPv4...
    - $\Rightarrow$ $\approx$15 τρις. «IPv4 Internets» ανά cm^2, συμπεριλαμβανομένων και των ωκεανών....
    - $\Rightarrow$ $\approx 5 * 10^{28}$ διευθύνσεις ανά άνθρωπο στη Γη
  - $\Rightarrow$ Το NAT θεωρείται πλέον ξεπερασμένο.
- Μεθόδους αυτόματης ρυθμίσεις παραμέτρων δικτύου
  - $\Rightarrow$ Το DHCP θεωρείται πλέον ξεπερασμένο.
- Καλύτερη ασφάλεια, με το IPsec να είναι υποχρεωτικό.

IPv6: Η απάντηση

Προσφέρει:

- Εξ αρχής υποστήριξη για συνάθροιση διευθύνσεων.
  - Πίνακες δρομολόγησης το πολύ 8,192 εγγραφών.
- Υποχρεωτική υποστήριξη για multicast.
- Καλύτερη υποστήριξη για κινητούς χρήστες.
  - Anycast επικοινωνία και scoped διευθύνσεις.
- Απλουστευμένη/βελτιστοποιημένη μορφή πακέτων.
  - $\Rightarrow$ Γρηγορότερη επεξεργασία των πακέτων στους δρομολογητές και μεγαλύτερη επεκτασιμότητα.
- Μεθόδους ομαλής μετάβασης από το IPv4.
  - Dual stack, IPv6-over-IPv4 (6to4) & Teredo tunneling, relays.

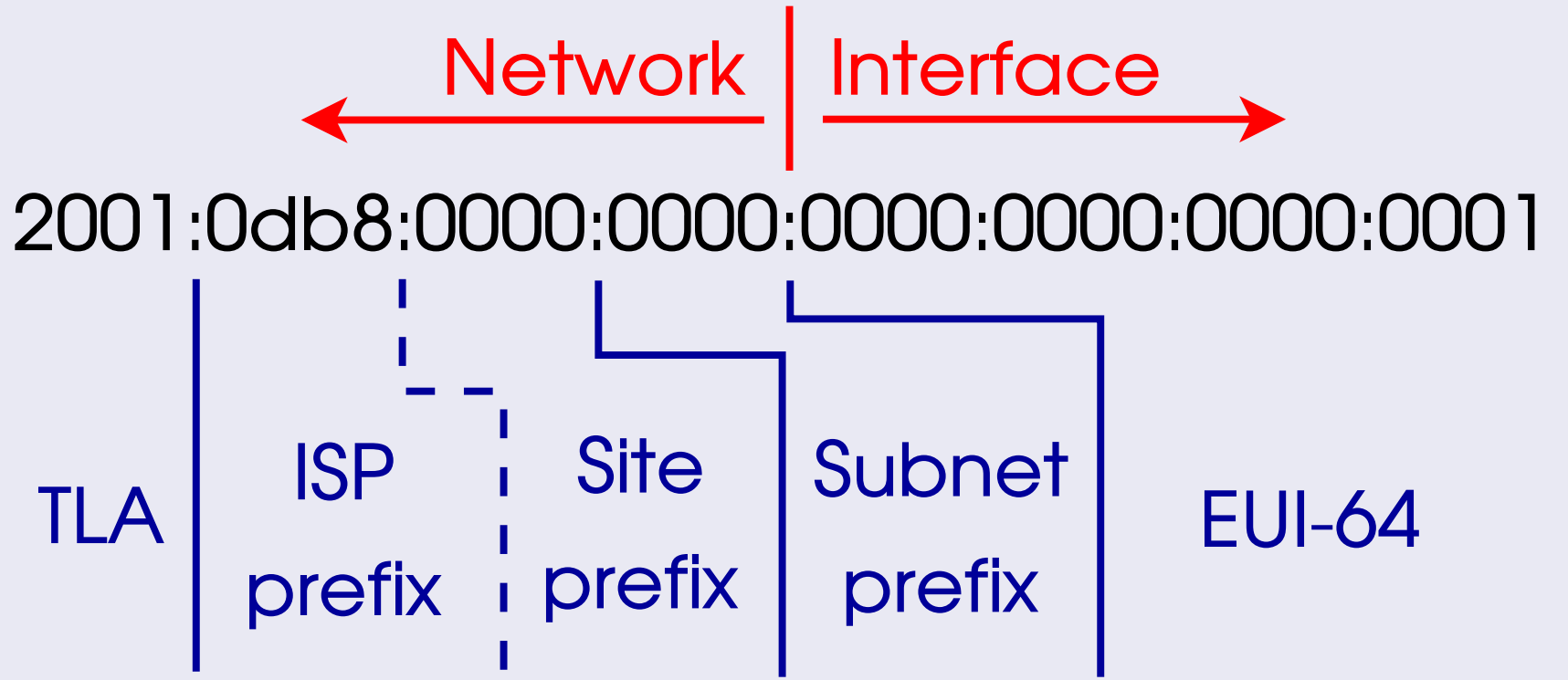
Διευθύνσεις

Μορφή διευθύνσεων

- Βασική μορφή: τετράδες δεκαεξαδικών ψηφίων, χωρισμένων από «:».
  - 2001:0db8:0000:0000:0000:0000:0000:0001 (RFC 3849).
  - 2001:4860:a003:0000:0000:0000:0000:0068 (ipv6.google.com).
- Κάθε μηδενικό πρόθεμα ανά τετράδα μπορεί να αγνοηθεί.
  - 2001:db8:0:0:0:0:0:1.
  - 2001:4860:a003:0:0:0:0:68.
- Διαδοχικές μηδενικές τετράδες μπορούν να αντικατασταθούν με «::».
  - 2001:db8::1.
  - 2001:4860:a003::68.
  - Το «::» μπορεί να χρησιμοποιηθεί μόνο μία φορά ανά διεύθυνση.
- Προσθέτουμε στο τέλος το μήκος του προθέματος δρομολόγησης.
  - 2001:db8::/32
  - 2001:4860::/32.
- Εξ ορισμού 64 bits για διεύθυνση διεπαφής (interface) και 64 bits για διεύθυνση δικτύου.

Διευθύνσεις

Μορφή διευθύνσεων



Διευθύνσεις

Ειδικές κατηγορίες διευθύνσεων

- ::/128 – «μη ρυθμισμένη σύνδεση».
- ::1/128 – loopback διεύθυνση.
- fe80::/10 – link-local διευθύνσεις.
- fc00::/7 – «μοναδικές τοπικές διευθύνσεις».
- ff00::/8 – διευθύνσεις για multicast.
- ff02::1:ffxx:xxxx – διευθύνσεις για την εύρεση γειτόνων.
- ::ffff:0:0/96 – διευθύνσεις μεταφρασμένες από το IPv4.
- 2000::/3 – διευθύνσεις για unicast.
- 2001::/32 – διευθύνσεις Teredo tunnels.
- 2001:10::/28 – ORCHID (RFC 4843).
- 2001:db8::/32 – διευθύνσεις για παραδείγματα.
- 2002::/16 – διευθύνσεις για 6to4 tunnels.

Αυτόματη ρύθμιση διευθύνσεων

Στο IPv4:

- 48-bit MAC διευθύνσεις.
- ARP/RARP $\rightarrow$ BOOTP $\rightarrow$ DHCP.
- DHCP:
  1. Ο κόμβος στέλνει μία αίτηση με broadcast.
     - Εσωκλείει και την MAC διεύθυνσή του.
     - Broadcast $\Rightarrow$ απαιτεί υποστήριξη από τους routers.
  2. Ο εξυπηρετητής DHCP απαντά απευθείας στον κόμβο με την νέα του IPv4 διεύθυνση.
     - Απαιτείται η ύπαρξη εξυπηρετητή DHCP.

Στο IPv6:

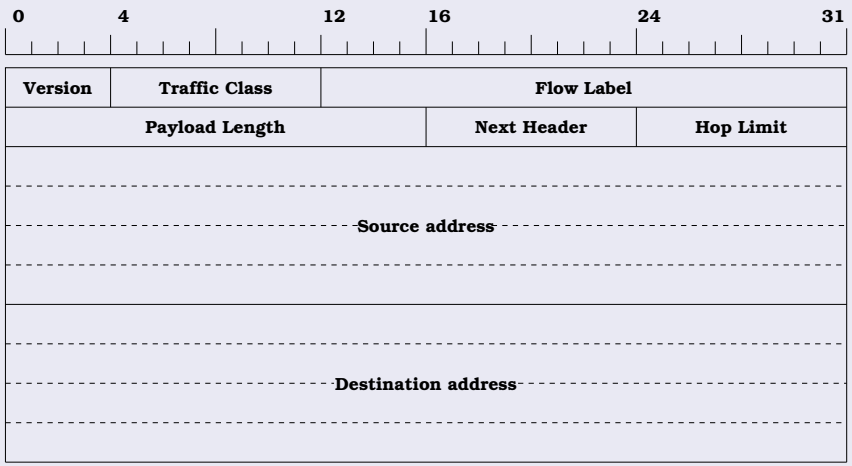
- 64-bit διευθύνσεις διεπαφής.
  - Μπορεί να βασίζονται στην MAC διεύθυνση της διεπαφής (EUI-64 – RFC 3596).
  - Π.χ.: 00 : a0 : c9 : a6 : 64 : 4d $\rightarrow$ 02a0 : c9**ff** : **fe**a6 : 644d.

  1. Neighbor Discovery Protocol (NDP -- RFC 4861).
  2. Ο κόμβος στέλνει αίτηση στην «multicast» link-local διεύθυνση του δικτύου.
  3. Ο router του δικτύου απαντά με ένα σύνολο κατάλληλων ρυθμίσεων.

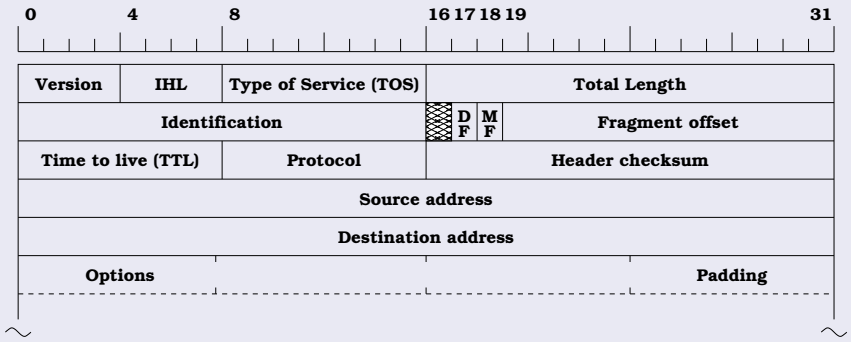
- Δυνατότητα για χειροκίνητη ρύθμιση αλλά και για DHCPv6.

Το PDU του IPv6

Βασική κεφαλίδα πακέτου IPv6

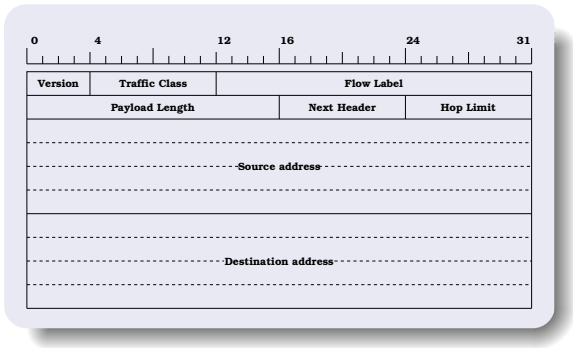


Κεφαλίδα πακέτου IPv4



Η βασική κεφαλίδα του IPv6

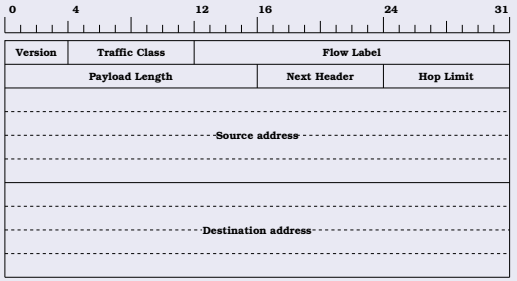
- Version: Καθορίζει την έκδοση του IP στην οποία ανήκει το πακέτο (6 = 0110).
- Traffic class: Ορίζει την προτεραιότητα του πακέτου.
- Flow label: Ορίζει μία «ροή πακέτων» τα οποία πρέπει όλα να έχουν την ίδια αντιμετώπιση.
- Payload length: Το μέγεθος του πακέτου **εκτός από την βασική κεφαλίδα**.
- Next header: Ο τύπος των δεδομένων που ακολουθούν την βασική κεφαλίδα.
- Hop limit: Όριο αλμάτων για το πακέτο (αντίστοιχο του πεδίου TTL του IPv4).
- Source/Destination address: Η 128-bit διεύθυνση του αποστολέα/παραλήπτη του πακέτου.



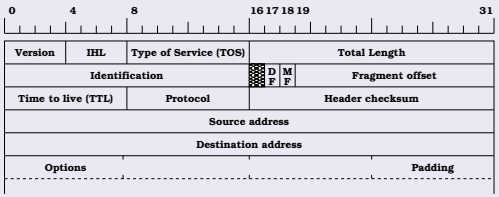
Σύγκριση με την κεφαλίδα του IPv4

- Δεν υπάρχουν τα πεδία:
  - IHL: Η βασική κεφαλίδα έχει **σταθερό μέγεθος** (40 bytes).
  - Don't Fragment (DF): Τα πακέτα του IPv6 **δεν κατακερματίζονται στην πορεία παρά μόνο στα άκρα της σύνδεσης**.
  - More Fragments (MF), Identification, Fragment Offset: Τα τμήματα του πακέτου ταυτοποιούνται με την κεφαλίδα κατακερματισμού.
  - Header checksum: Το IPv6 **δεν χρησιμοποιεί έλεγχο για την κεφαλίδα**. Τα ανώτερα πρωτόκολλα πρέπει υποχρεωτικά να παρέχουν τον έλεγχο αυτό!
  - Options: Οι οποιεσδήποτε επεκτάσεις του βασικού πρωτοκόλλου ορίζονται με πρόσθετες κεφαλίδες επέκτασης
- Έχουν αλλαχθεί τα πεδία:
  - Total length: Έγινε payload length και μετράει μόνο το μέγεθος των δεδομένων μετά την βασική κεφαλίδα.
  - Protocol: Έγινε next header για να μπορεί να εκφράσει και κεφαλίδες επέκτασης εκτός από μόνο είδος SDU.
  - TTL: Έγινε hop limit.

IPv6



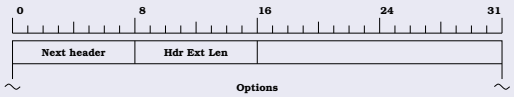
IPv4



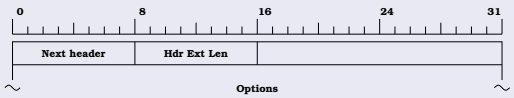
Οι κεφαλίδες επέκτασης του IPv6

- Το πεδίο next header επιτρέπει τη δημιουργία αλυσίδων κεφαλίδων.
- Ορίζονται οι εξής κεφαλίδες επέκτασης (extension headers):
  - Επιλογών ανά άλμα (Hop-by-hop options header).
  - Επιλογών παραλήπτη (Destination options header).
  - Δρομολόγησης (Routing header).
  - Κατακερματισμού (Fragmentation header).
  - Υποστήριξης IPsec:
    - Αυθεντικοποίησης (Authentication Header).
    - Ασφαλείας ενθυλάκωσης δεδομένων (Encapsulating Security Payload header).
- Οι κεφαλίδες επέκτασης (εκτός από το hop-by-hop header) αφορούν μόνο τον τελικό παραλήπτη.
- Προτεινόμενη σειρά κεφαλίδων:
  - IPv6 header → Hop-by-hop options header → Destination options header → Routing header → Fragmentation header → Authentication header → Encapsulating Security Payload header → Destination options header → SDU.

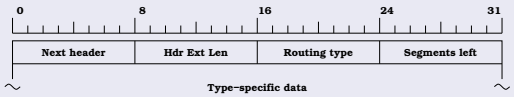
Hop-by-hop options header



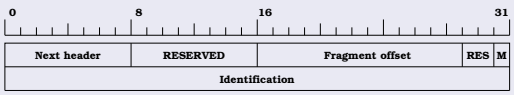
Destination options header



Routing header



Fragmentation header



Μέγεθος πακέτου

Κατακερματισμός

- Το IPv6 δεν υποστηρίζει κατακερματισμό/επανασύθεση στα ενδιάμεσα άλματα παρά μόνο στα άκρα της σύνδεσης.

⇒ **Τότε τι γίνεται όταν πρέπει να περάσουμε από δίκτυα με διαφορετικό MTU;**

- **Λύση:**
  - **Path MTU Discovery** στην αρχή της επικοινωνίας και χρήση της Κεφαλίδας Κατακερματισμού.
  - Χρήση του **ελάχιστου MTU** που προβλέπει το IPv6 (1280 bytes).

Jumbograms

- Η βασική κεφαλίδα έχει πεδίο μεγέθους 16 bits ⇒ μέγιστο payload: 65,535 Bytes.

⇒ **Κι αν θέλω να μεταδώσω (πολύ) μεγαλύτερα πακέτα;**

- **Λύση:**
  - **Jumbograms** (RFC 2675).
    - Υλοποιείται με χρήση του hop-by-hop option header.
    - Επιτρέπει πακέτα μεγέθους $2^{32} - 1 \approx$ **4 Gbytes**.

Πρωτόκολλα ανωτέρων επιπέδων

- Όσα πρωτόκολλα λαμβάνουν υπόψιν την IP διεύθυνση αποστολέα/παραλήπτη πρέπει να αλλαχθούν.
- ICMPv6: Σχεδόν ίδιο με το ICMPv4.
- UDP, TCP: Όπως και στο IPv4 (αλλάζει μόνο το checksum της ψευδοκεφαλίδας).
- Πρωτόκολλα πάνω από τα TCP/UDP: Καμία αλλαγή.
  - HTTP, FTP, SMTP, ... δουλεύουν ακριβώς όπως και στο IPv4.
  - Μόνο αν χρησιμοποιούμε απευθείας διευθύνσεις IP στο πρωτόκολλο θα χρειαστεί κάποια αλλαγή
    - FTP: PORT, PASV → EPRT, EPSV (RFC 2428).
    - HTTP: http://<IPv4 address>:<port number> → http://[<IPv6 address>] :<port number> (RFC 2732).

Domain Name System

Η ανάγκη για το DNS

- Οι διευθύνσεις IP δεν είναι σχεδιασμένες για ανθρώπους.
  - Ποιος θυμάται την 2η διεύθυνση στα παραδείγματα για το IPv6;
  - Ποιος θυμάται την IP διεύθυνση του zeus;
- Λύση:
  - Συμβολικά ονόματα που μεταφράζονται σε διευθύνσεις.

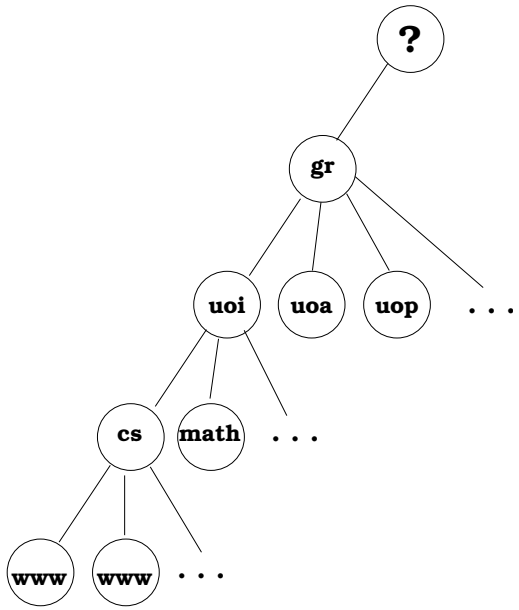
  $\Rightarrow$ **Domain Name System**.

Γενικά

- Πρωτόκολλο επιπέδου εφαρμογών.
- Χρησιμοποιεί το UDP και κατά σύμβαση την θύρα 53.
- Ιεραρχική δομή ονομάτων και χώρου διευθύνσεων.
  - Top-Level Domains, domains, subdomains, ...

  1. Root nameservers: Μεταφράζουν το `.` μετά το όνομα.
     - 13 root nameservers: `[A-M].ROOT-SERVERS.NET.`
  2. TLD nameservers: Διαχειρίζονται και μεταφράζουν το τελευταίο τμήμα του ονόματος.
     - Π.χ. ανά χώρα (`.gr`) ή είδος διεύθυνσης (`.com`, `.net`, `.org` κτλ.)
  3. Authoritative nameservers: Διαχειρίζονται τα ονόματα ανά πάροχο (ISPs) ή άλλη οργανωτική δομή.
  4. Local nameservers: Διαχειρίζονται τα ονόματα ανά οργανισμό.

- `dig +trace www.cs.uoi.gr`
- Το πολύ 63 χαρακτήρες ανά επίπεδο, 255 χαρακτήρες στο συνολικό συμβολικό όνομα (RFC 1034).



Εγγραφές

- Κάθε εγγραφή αποτελείται από τα εξής πεδία (RFC 1035):
  - NAME: Το «όνομα» της εγγραφής.
  - TYPE: Ορίζει το είδος της εγγραφής.
  - CLASS: Ορίζει την τάξη της εγγραφής (για το Διαδίκτυο = IN).
  - TTL: Ορίζει την διάρκεια ισχύος της εγγραφής αυτής (caching)..
  - RDLENGTH: Μέγεθος του πεδίου RDATA.
  - RDATA: Ανάλογα με την τιμή του πεδίου TYPE, η αντιστοιχία στο «όνομα».
- Παραδείγματα τύπων εγγραφών:
  - SOA: συμβολίζει την αρχή μίας «ζώνης εξουσίας».
  - NS: αντιστοιχία ονόματος $\leftrightarrow$ υπεύθυνου nameserver.
  - A: αντιστοιχία ονόματος $\leftrightarrow$ διεύθυνσης IPv4.
  - AAAA: αντιστοιχία ονόματος $\leftrightarrow$ διεύθυνσης IPv6.
  - PTR: αντιστοιχία διεύθυνσης IPv4/IPv6 $\leftrightarrow$ ονόματος.
  - CNAME: αντιστοιχία alias $\leftrightarrow$ κανονικού ονόματος.
  - MX: αντιστοιχία «ονόματος δικτύου» $\leftrightarrow$ mailserver.
- Δυνατότητες:
  - Πολλά ονόματα ανά κόμβο (aliases).
  - Πολλοί κόμβοι ανά όνομα (DNS Round Robin).
  - Πολλούς εξυπηρετητές DNS, SMTP ανά domain.
  - Ειδικού τύπου εγγραφές (TXT, HINFO, MINFO κτλ.)

Μετάφραση ονομάτων

- Η μετάφραση των ονομάτων γίνεται σε βήματα.
  1. Η εφαρμογή ελέγχει την τοπική της cache ονομάτων (αν έχει).
  2. Η εφαρμογή ρωτά το λειτουργικό σύστημα. Αν το λειτουργικό έχει την απάντηση στην cache του, απαντά.
  3. Το λειτουργικό σύστημα ρωτά τον τοπικό nameserver. Αν αυτός έχει την απάντηση στην cache του, απαντά.
  4. Αν ο nameserver που ρωτήσαμε στο προηγούμενο βήμα επιτρέπει αναδρομικά ερωτήματα, περιμένουμε την απάντηση.
  5. Αλλιώς, στέλνουμε το ερώτημα σε έναν nameserver για το υψηλότερο επίπεδο που δεν έχουμε ακόμα μεταφράσει και επαναλαμβάνουμε τη διαδικασία για διαδοχικά χαμηλότερα επίπεδα στο συμβολικό όνομα.
     - Μπορεί να πάρουμε ως απάντηση είτε απευθείας την αντιστοιχία ονόματος $\leftrightarrow$ IP διεύθυνσης, είτε έναν «δείκτη» για να συνεχίσουμε την αναζήτηση.
     - Σε κάθε περίπτωση, ψάχνουμε τον αντίστοιχο authoritative nameserver.

BSD sockets

Γενικά

- Υπάρχει μόνο ένας τρόπος δικτυακής επικοινωνίας διεργασιών: **μέσω αποστολής και λήψης μηνυμάτων**.
  - Όλα τα είδη επικοινωνίας βασίζονται σε μηνύματα.
- Υπάρχει μόνο ένας τρόπος αποστολής και λήψης μηνυμάτων: **χρησιμοποιώντας sockets**.
  - Όλες οι άλλες μορφές επικοινωνίας βασίζονται σε sockets.
- Επιτρέπουν την επικοινωνία διεργασιών που εκτελούνται είτε στον ίδιο κόμβο είτε σε απομακρυσμένους κόμβους του δικτύου.
- Βασικές λειτουργίες:
  - Διευθυνσιοδότηση: σε ποιόν κόμβο θα πάνε τα δεδομένα;
  - Πολυπλεξία: σε ποια διεργασία θα παραδοθούν τα δεδομένα, όταν φτάσουν στον κόμβο;
- Βασική πληροφορία του socket: `<IP address + port number>` (ή filename για επικοινωνία τοπικά σε κάθε κόμβο).
- Κάθε σύνδεση έχει δύο sockets στα άκρα της.
- Ουσιαστικά υλοποιούν το επίπεδο συνεδρίας του μοντέλου OSI.

BSD sockets

- Έχουν προδιαγραφεί πολλές διεπαφές προγραμματισμού εφαρμογών (APIs) για προγραμματισμό με sockets: BSD sockets, XTI, TLI, OpenTransport, STREAMS, WinSock, ...
- Τα **BSD sockets** όμως επικράτησαν.

- Τα BSD sockets πρωτοεμφανίστηκαν στο 4.2BSD (4.1c για την ακρίβεια) το 1982 και «καθιερώθηκαν» με το 4.3BSD το 1986.
- Υποστηρίζονται τα πρωτόκολλα UDP, TCP και SCTP, καθώς και πρωτόκολλα χαμηλότερων επιπέδων (IPv4, IPv6, ICMP, IGMP, ARP, ...)

Βασικές λειτουργίες

- Το API των BSD sockets προσφέρει κλήσεις συστήματος για:
  - Μετατροπή δεδομένων από/προς μορφή αναπαράστασης δικτύου (network/host byte order).
    - `htonl(3)`, `htons(3)`, `ntohl(3)`, `ntohs(3)`.
  - Μετάφραση ονομάτων και διευθύνσεων.
    - `getpeername(2)`, `getsockname(2)`, `gethostbyname(3)`, `gethostent(3)`, `getprotoent(3)`, `getaddrinfo(3)`, `getnameinfo(3)`, `inet_pton(3)`, `inet_ntop(3)`, ...
  - Καθορισμό των άκρων επικοινωνίας.
    - `socket(2)`, `socketpair(2)`, `bind(2)`, `listen(2)`.
  - Αποστολή/αποδοχή αίτησης σύνδεσης.
    - `connect(2)`, `accept(2)`.
  - Αποστολή και λήψη δεδομένων.
    - `read(2)`, `recv(2)`, `recvfrom(2)`, `recvmsg(2)`, `write(2)`, `send(2)`, `sendto(2)`, `sendmsg(2)`.
  - Τερματισμό σύνδεσης.
    - `shutdown(2)`, `close(2)`.
  - Διαχείριση παραμέτρων και σφαλμάτων.
    - `setsockopt(2)`, `getsockopt(2)`, `gai_strerror(3)`.

Network vs System Byte Ordering

Το πρόβλημα του endianess

- Έστω η συμβολοσειρά «ΔΓΒΑ» ⇒ Πώς αποθηκεύεται/μεταδίδεται;
- Δύο απαντήσεις:
  - Big-endian ⇒ «ΔΓΒΑ».
  - Little-endian ⇒ «ΑΒΓΔ».
  - Καθορίζεται από την αρχιτεκτονική του επεξεργαστή ή/και το λειτουργικό σύστημα.
    - x86/x86_64: little-endian, sun4u/powerpc: big-endian.

test-endianess.c

```
#include <unistd.h>
#include <inttypes.h>
int main()
{
   uint32_t i = 0x33323130; // "3210"
   write(1, &i, sizeof(uint32_t));
   return 0;

}
```

zeus (big-endian)

```
zeus:~> ./test-endianess
3210
```

ace (little-endian)

```
ace:~> ./test-endianess
0123
```

Network vs System Byte Ordering

Μετατροπή endianess

- Στα IP δίκτυα τα δεδομένα μεταφέρονται σε big-endian σειρά (network byte ordering).
- Όλα τα πεδία των δομών του API **είναι σε network byte ordering!**

⇒ Κι αν ο υπολογιστής μου έχει διαφορετικό endianess;

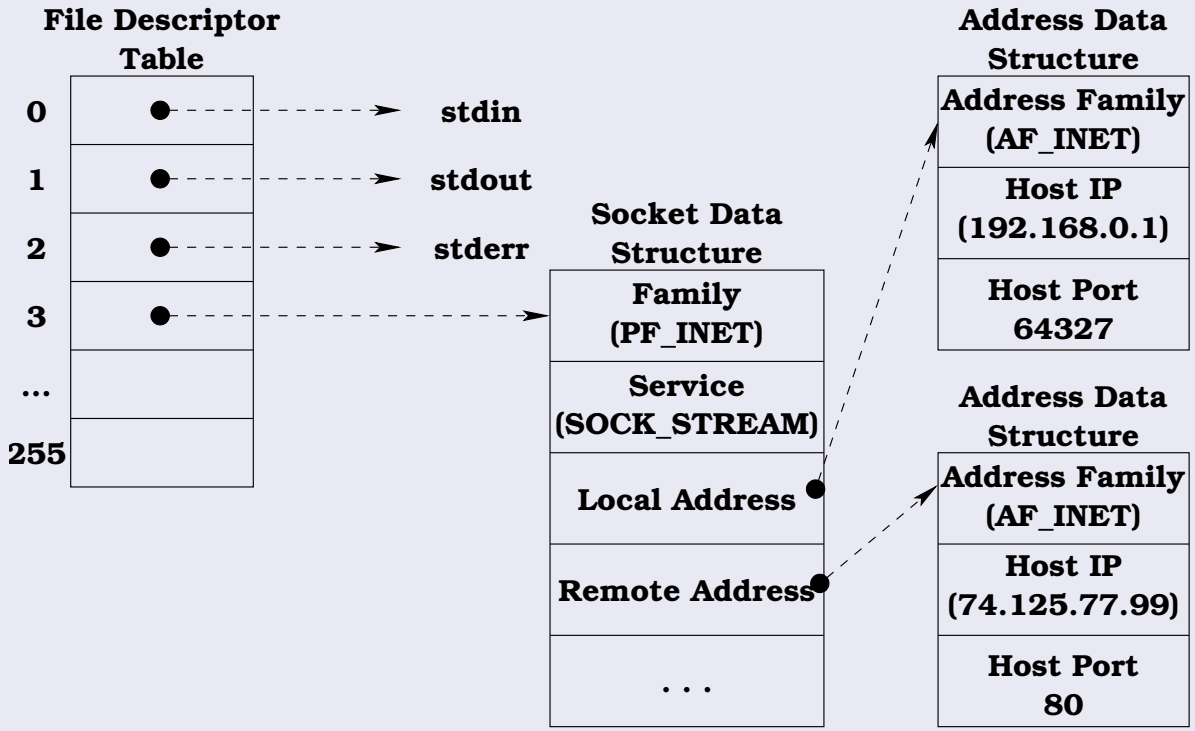
- `htons(3)`, `htonl(3)`, `ntohs(3)`, `ntohl(3)`
  (`h`: host, `n`: network, `s`: 16-bit int, `l`: 32-bit int).

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

«Στο UNIX όλα είναι αρχεία...»

Δομή ενός socket

- Ένα BSD socket είναι μία δομή δεδομένων σε kernel space.



- Το σύστημα το χειρίζεται σαν **αρχείο**.

Οι δομές δεδομένων `struct sockaddr*`

struct sockaddr

```
struct sockaddr {
   sa_family_t     sa_family;
   char            sa_data[14];
};
```

struct sockaddr_un

```
struct sockaddr_un {
   unsigned char   sun_len;
   sa_family_t     sun_family;
   char            sun_path[104];
};
```

struct sockaddr_in

```
struct sockaddr_in {
   unsigned char   sin_len;
   sa_family_t     sin_family;
   in_port_t       sin_port;
   struct in_addr  sin_addr;
   char            sin_zero[8];
};
```

struct sockaddr_in6

```
struct sockaddr_in6 {
   unsigned char   sin6_len;
   sa_family_t     sin6_family;
   in_port_t       sin6_port;
   uint32_t        sin6_flowinfo;
   struct in6_addr sin6_addr;
   uint32_t        sin6_scope_id;
};
```

struct in_addr

```
struct in_addr {
   in_addr_t s_addr;
};
```

struct in6_addr

```
struct in6_addr {
   uint8_t __u6_addr8[16];
};

#define s6_addr __u6_addr8
```

Οι δομές δεδομένων `struct sockaddr*`

Πεδία των δομών `struct sockaddr*`

- `{sa,sun,sin,sin6}_len`: πραγματικό μέγεθος της δομής σε bytes.
- `{sa,sun,sin,sin6}_family`: «οικογένεια» πρωτοκόλλων που περιγράφει η δομή.
  - `AF_LOCAL`: η διεύθυνση είναι ένα όνομα αρχείου (unix-domain).
  - `AF_INET`: διεύθυνση IPv4.
  - `AF_INET6`: διεύθυνση IPv6.
  - ...
- `sin_port/sin6_port`: ο αριθμός θύρας του άκρου επικοινωνίας.
- `sin_addr/sin6_addr`: η IP διεύθυνση του άκρου επικοινωνίας.
- `sin6_flowinfo`: καθορίζει την κλάση και τη ροή της σύνδεσης αυτής.
- `sin6_scope_id`: καθορίζει το scope της διεύθυνσης (link-local, site-local, ...)

Οι δομές δεδομένων `struct sockaddr*`

- Για λόγους μεταφερσιμότητας χρησιμοποιούνται οι τύποι δεδομένων: `sa_family_t` (=`uint32_t`), `in_addr_t` (=`uint32_t`) και `in_port_t` (=`uint16_t`).
- Επίσης, για τους ίδιους λόγους, ορίζεται και η δομή δεδομένων `struct sockaddr_storage` η οποία είναι αρκετά μεγάλη για να «χωρά» οποιαδήποτε από τις προαναφερθείσες δομές `struct sockaddr*`.

`struct sockaddr_storage`

```
struct sockaddr_storage {
   unsigned char   ss_len;
   sa_family_t     ss_family;
   char            __ss_pad1[_SS_PADSIZE];
   __int64_t       __ss_align;
   char            __ss_pad2[_SS_PADSIZE];
};
```

Μετατροπή διευθύνσεων

- Ακόμα και οι προγραμματιστές έχουν προβλήματα μνήμης...
- Τα συμβολικά ονόματα είναι σίγουρα προτιμότερα από τις διευθύνσεις IP...
- Οι ASCII αναπαραστάσεις είναι σίγουρα προτιμότερες από να γράφουμε απευθείας σε binary...

$\Rightarrow$ Χρειαζόμαστε συναρτήσεις μετατροπής!

Μετατροπή διευθύνσεων

Δυαδική και ASCII μορφή

- Υπάρχουν πολλές συναρτήσεις μετατροπής ανάμεσα σε αυτές τις μορφές διευθύνσεων.
  - `int inet_aton(const char *cp, struct in_addr *pin);`
  - `in_addr_t inet_addr(const char *cp);`
  - `char * inet_ntoa(struct in_addr in);`
  - `...`
- Δεν υποστηρίζουν IPv6 διευθύνσεις!
- Λύση: `inet_ntop(3)` και `inet_pton(3)`.

Μετατροπή διευθύνσεων

Από δυαδική σε ASCII μορφή

```
const char *inet_ntop(int af, const void *src, char *dst,
socklen_t size);
```

- `af`: οικογένεια διεύθυνσης (`AF_INET`, `AF_INET6`).
- `src`: δείκτης σε αντίστοιχη δομή διεύθυνσης (`struct in_addr`, `struct in6_addr`) σε network byte order.
- `dst`: δείκτης σε πίνακα χαρακτήρων όπου θα αποθηκευτεί το αποτέλεσμα.
- `size`: μέγεθος του πίνακα `dst` (μέγιστο `INET_ADDRSTRLEN` ή `INET6_ADDRSTRLEN`).
- Επιστρέφει δείκτη στον πίνακα του αποτελέσματος σε επιτυχία ή `NULL` για αποτυχία.

Από ASCII σε δυαδική μορφή

```
int inet_pton(int af, const char *src, void *dst);
```

- `af`: όπως παραπάνω.
- `src/dst`: όπως το `dst/src` στην `inet_ntop` αντίστοιχα.
- Επιστρέφει 1 αν η μετατροπή πέτυχε, 0 αν η διεύθυνση στο `src` δεν ισχύει, -1 σε περίπτωση άλλου λάθους.

Μετατροπή διευθύνσεων

Συμβολικά ονόματα και διευθύνσεις IP

- Υπάρχουν πολλές συναρτήσεις μετατροπής ανάμεσα σε αυτές τις μορφές διευθύνσεων/ονομάτων θυρών.
  - `struct hostent * gethostbyname(const char *name);`
  - `struct hostent * gethostbyname2(const char *name);`
  - `struct hostent * gethostbyaddr(const void *addr, socklen_t len, int type);`
  - `struct servent * getservbyname(const char *name, const char *proto);`
  - `struct servent * getservbyport(int port, const char *proto);`
  - `...`

```
struct  hostent {
   char *h_name;        /* name of host */
   char **h_aliases;    /* alias list */
   int  h_addrtype;     /* address type */
   int  h_length;       /* address length */
   char **h_addr_list; /* addr. from DNS */
};
#define h_addr  h_addr_list[0]
```

```
struct  servent {
   char *s_name;      /* name of service */
   char **s_aliases; /* alias list */
   int  s_port;       /* service port */
   char *s_proto;     /* service protocol */
};
```

Μετατροπή διευθύνσεων

Από hostname σε διεύθυνση IP και κάτι παραπάνω

- Οι περισσότερες από αυτές τις συναρτήσεις δεν υποστηρίζουν IPv6 διευθύνσεις.
- Ακόμα και αυτές που υποστηρίζουν (`inet_pton(3)`, `inet_ntop(3)`, `gethostbyname2(3)`) παίρνουν ως όρισμα το address family...
- Λύση:
  - `getaddrinfo(3)`: μετατροπή από διευθύνσεις (συμβολικές ή αριθμητικές) και υπηρεσίες (ονομαστικά ή με αριθμό θύρας) σε κατάλληλες δομές δεδομένων.
  - `getnameinfo(3)`: μετατροπή από δομές δεδομένων σε συμβολική/αριθμητική διεύθυνση και ονομαστική/αριθμητική υπηρεσία.

Μετατροπή διευθύνσεων

getaddrinfo(3)

```
int getaddrinfo(const char *hostname, const char *servname,
                const struct addrinfo *hints,
                struct addrinfo **res);
void freeaddrinfo(struct addrinfo *ai);
```

- `hostname`: Συμβολικό όνομα κόμβου ή IP διεύθυνση σε ASCII αναπαράσταση (ή `NULL`).
- `servname`: Συμβολικό όνομα υπηρεσίας (`services(5)`) ή αριθμός θύρας σε ASCII αναπαράσταση (ή `NULL`).
- `hints`: δείκτης σε δομή τύπου `struct addrinfo` (ή `NULL`) – καθορίζει επιλογές του χρήστη όσον αφορά το είδος της σύνδεσης.
- `res`: δείκτης σε συνδεδεμένη λίστα με τα αποτελέσματα της μετατροπής.
- Επιστρέφει 0 για επιτυχία, κωδικού λάθους (`gai_strerror(3)`) για αποτυχία.
- Η `freeadrinfo(3)` απελευθερώνει τον χώρο που δέσμευσε η `getaddrinfo(3)` για το αποτέλεσμα (`res`).

Μετατροπή διευθύνσεων

Η δομή δεδομένων `struct addrinfo`

```
struct addrinfo {
   int ai_flags;              /* input flags */
   int ai_family;             /* PF_xxx or PF_UNSPEC*/
   int ai_socktype;           /* SOCK_xxx or 0 */
   int ai_protocol;           /* IPPROTO_xxx or 0 */
   socklen_t ai_addrlen;      /* length of socket-address */
   struct sockaddr *ai_addr; /* socket-address for socket */
   char *ai_canonname;        /* canonical name */
   struct addrinfo *ai_next; /* pointer to next in list */
};
```

- `ai_flags`:
  - `AI_PASSIVE`: η προς μετάφραση διεύθυνση θα χρησιμοποιηθεί ως «server». Αν το `hostname` είναι `NULL` τότε η IP θα τεθεί στην τιμή `INADDR_ANY` ή `INADDR6_ANY_INIT`.
  - `AI_CANONNAME`: επέστρεψε το κανονικό συμβολικό όνομα του κόμβου.
  - `AI_NUMERICHOST`: το `hostname` έχει αριθμητική διεύθυνση IP.
  - `AI_ADDRCONFIG`: επέστρεψε μόνο διευθύνσεις για πρωτόκολλα για τα οποία το τρέχον σύστημα έχει διεύθυνση.
  - `AI_NUMERICSERV`: το `servname` έχει αριθμό θύρας.
- Τα `ai_family`, `ai_socktype` και `ai_protocol` χρησιμοποιούνται στην `socket(2)`.

Μετατροπή διευθύνσεων

getnameinfo(3)

```
int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen,
                int flags);
```

- `sa, salen`: δείκτης σε μία από τις δομές `sockaddr` και το αντίστοιχο μέγεθός της.
- `host`: δείκτης σε χώρο μνήμης μεγέθους `hostlen` (μέγιστο `NI_MAXHOST`) στον οποίο θα γραφεί η διεύθυνση/το όνομα του κόμβου.
- `serv`: δείκτης σε χώρο μνήμης μεγέθους `servlen` (μέγιστο `NI_MAXSERV`) στον οποίο θα γραφεί το συμβολικό όνομα της υπηρεσίας. Αν το `serv` είναι `NULL` ή το `servlen` 0, τίποτα δε θα γραφεί εκεί.
- `flags`:
  - `NI_NOFQDN`: ζητείται μόνο το «τοπικό» κομμάτι του ονόματος του κόμβου (χωρίς το domain).
  - `NI_NUMERICHOST`: ζητείται η διεύθυνση IP του κόμβου.
  - `NI_NAMEREQD`: ζητείται συμβολικό όνομα. Αν αυτή η τιμή έχει τεθεί και δεν μπορεί να βρεθεί συμβολικό όνομα, επιστρέφεται σφάλμα. Στην αντίθετη περίπτωση, επιστρέφεται η διεύθυνση IP.
  - `NI_NUMERICSERV`: ζητείται ο αριθμός θύρας της υπηρεσίας (σε ASCII αναπαράσταση).
  - `NI_DGRAM`: ζητείται υπηρεσία πρωτοκόλλου UDP (για ειδικές περιπτώσεις που στην ίδια θύρα υπάρχει άλλη υπηρεσία για TCP και UDP).
- Επιστρέφει 0 για επιτυχία, κωδικού λάθους (`gai_strerror(3)`) για αποτυχία.

«Άνοιγμα» ενός socket

`socket(2)`

```
int socket(int domain, int type, int protocol);
```

- Δημιουργεί το ένα άκρο της επικοινωνίας.
- `domain`: Καθορίζει το «μέσο» επικοινωνίας.
  - `PF_LOCAL/PF_UNIX`: Τοπική επικοινωνία (unix-domain).
  - `PF_INET`: Επικοινωνία πάνω από IPv4.
  - `PF_INET6`: Επικοινωνία πάνω από IPv6.
  - `...`
  - `PF_LINK`: Επικοινωνία πάνω από το επίπεδο συνδέσμου.
  - `PF_ROUTE`: Επικοινωνία με το μηχανισμό δρομολόγησης.
- `type`: καθορίζει το είδος της επικοινωνίας.
  - `SOCK_STREAM`: Συνδεσμοστραφής επικοινωνία (π.χ. TCP, SCTP).
  - `SOCK_DGRAM`: Ασυνδεσμική επικοινωνία με datagrams (π.χ. UDP).
  - `SOCK_SEQPACKET`: Ασυνδεσμική επικοινωνία ακολουθίας πακέτων (π.χ. SCTP).
  - `SOCK_RAW`: «Επικοινωνία» απευθείας με το κατώτερο επίπεδο.
- `protocol`: καθορίζει ένα συγκεκριμένο πρωτόκολλο προς χρήση με το socket.
  - default: 0.
  - `IPPROTO_IP, IPPROTO_TCP, IPPROTO_UDP, ...`
- Επιστρέφει: τον socket descriptor ($\geq$0) ή -1 για σφάλμα.

«Άνοιγμα» ενός socket

`socketpair(2)`

```
int socketpair(int domain, int type, int protocol, int *sv);
```

- Δημιουργεί ένας ζεύγος συνδεδεμένων sockets
- Οι παράμετροι είναι όπως και στην `socket(2)`.
- `sv`: Πίνακας δύο θέσεων στον οποίο επιστρέφονται οι socket descriptors.
- Επιστρέφει: 0 για επιτυχία ή -1 για σφάλμα.

Προετοιμασία ενός socket για εισερχόμενες συνδέσεις

`bind(2)`

```
int bind(int s, const struct sockaddr *addr, socklen_t
addrlen);
```

- Χρησιμοποιείται στην πλευρά του εξυπηρετητή.
- Αντιστοιχίζει ένα socket με μία διεύθυνση και μία θύρα.
- `s`: Ο socket descriptor.
- `addr`: Δείκτης σε μία δομή τύπου `struct sockaddr`.
- `addrlen`: Το μέγεθος της δομής στην οποία δείχνει το `addr`. Χρησιμοποιείται ο τύπος `socklen_t` (=32-bit ακέραιος) για μεταφερσιμότητα.
- Επιστρέφει: 0 για επιτυχία ή -1 για σφάλμα.

Προετοιμασία ενός socket για εισερχόμενες συνδέσεις

`listen(2)`

```
int listen(int s, int backlog);
```

- Δηλώνει την «επιθυμία» της διεργασίας να δεχθεί συνδέσεις και καθορίζει ένα όριο στον αριθμό των εκκρεμών αιτήσεων σύνδεσης που θα υποστηρίζονται.
- Έχει νόημα μόνο για το server κομμάτι συνδεσμοστραφούς άκρου επικοινωνίας.
- `s`: Ο socket descriptor.
- `backlog`: Το μέγεθος της «ουράς» εκκρεμών αιτήσεων (στην πραγματικότητα η ουρά έχει 1.5× αυτό το μέγεθος).
- Επιστρέφει: 0 για επιτυχία ή -1 για σφάλμα.
- Το μέγεθος της ουράς μπορεί να αλλαχθεί με διαδοχικές κλήσεις για το ίδιο socket descriptor.

Αποστολή αιτήματος σύνδεσης

`connect(2)`

```
int connect(int s, const struct sockaddr *name,
socklen_t namelen);
```

- Χρησιμοποιείται και για συνδεσμοστραφή αλλά και για ασυνδεσμική επικοινωνία.
  - Συνδεσμοστραφής: αποστέλλει ένα αίτημα σύνδεσης στην απομακρυσμένη διεργασία (το στάδιο SYN του 3-way handshake) και περιμένει την απάντησή της (ACK) για να στείλει και την επιβεβαίωση της σύνδεσης (SYN+ACK).
  - Ασυνδεσμική: ορίζει απλά σε ποια διεύθυνση θα στέλνονται τα datagrams και από ποια διεύθυνση θα περιμένουμε απαντήσεις.
- `s`: ο socket descriptor από τον οποίο θα αποσταλεί το αίτημα σύνδεσης.
- `name`: δείκτης σε δομή τύπου `struct sockaddr` στην οποία περιέχεται πληροφορία (διεύθυνση + θύρα / όνομα αρχείου) για τον παραλήπτη του αιτήματος σύνδεσης.
- `namelen`: το μέγεθος σε bytes του `name`.
- Επιστρέφει 0 για επιτυχία, -1 για αποτυχία.

Αποδοχή αιτήματος σύνδεσης

`accept(2)`

```
int accept(int s, struct sockaddr *addr, socklen_t *
addrlen);
```

- Χρησιμοποιείται στην πλευρά του εξυπηρετητή συνδεσμοστραφούς επικοινωνίας.
- Υλοποιεί το στάδιο αποστολής επιβεβαίωσης (ACK) του 3-way handshake.
- `s`: ο socket descriptor στον οποίο θα λαμβάνονται οι εισερχόμενες συνδέσεις.
- `addr`: `NULL` ή δείκτης σε δομή τύπου `struct sockaddr` στην οποία θα αποθηκευθεί πληροφορία για τον αποστολέα του αιτήματος σύνδεσης.
- `addrlen`: `NULL` ή δείκτης σε ακέραιο στον οποίο αρχικά περιέχεται το μέγεθος σε bytes του `addr` και στον οποίο θα αποθηκευθεί το νέο μέγεθός του όταν επιστρέψει η συνάρτηση.
- Η κλήση αυτή (συνήθως) μπλοκάρει τον καλούντα έως ότου υπάρξει εισερχόμενη αίτηση σύνδεσης. Τότε δημιουργεί ένα νέο socket το οποίο και συνδέει με το αντίστοιχο της απομακρυσμένης διεργασίας.
- Επιστρέφει τον socket descriptor του νέου socket ή -1 για αποτυχία.

Τερματισμός σύνδεσης

close(2)

```
int close(int d);
```

- Χρησιμοποιείται και για συνδεσμοστραφή αλλά και για ασυνδεσμική επικοινωνία.
  - Και στις δύο περιπτώσεις: αποδεσμεύει την δομή που κρατούσε ο kernel για το socket.
  - Συνδεσμοστραφής: Αν η διεργασία που έκανε την κλήση είναι και η τελευταία που είχε αναφορά στο socket αυτό, τότε η κλήση αυτή αποστέλλει ή απορρίπτει όσα πακέτα εκκρεμούν και τερματίζει την σύνδεση (αντιστοιχεί στο τελικό στάδιο (FIN/FIN+ACK) της επικοινωνίας).
- `d`: ο socket descriptor.
- Επιστρέφει 0 για επιτυχία, -1 για αποτυχία.

Τερματισμός σύνδεσης

shutdown(2)

```
int shutdown(int s, int how);
```

- Η κλήση αυτή απενεργοποιεί την αποστολή και λήψη δεδομένων από ένα socket.
- Επιτρέπει την μετάβαση του socket σε κατάσταση half-closed (δεν υποστηρίζεται στο SCTP).
- `s`: ο socket descriptor.
- `how`: καθορίζει τον τύπο της λειτουργίας αυτής.
  - `SHUT_RD`: απενεργοποιείται η λήψη δεδομένων.
  - `SHUT_WR`: απενεργοποιείται η αποστολή δεδομένων.
  - `SHUT_RDWR`: απενεργοποιείται και η λήψη και η αποστολή δεδομένων.
- Η ακριβής λειτουργία και η τιμή επιστροφής στις περιπτώσεις `SHUT_WR` και επομένως και `SHUT_RDWR` καθορίζεται από το πρωτόκολλο του socket.
  - TCP και συνδεσμοστραφές SCTP: αποστέλλονται όλα τα εκκρεμόντα δεδομένα, αναμένεται επιβεβαίωση και κατόπιν αποστέλλεται το FIN. Επιστρέφει 0.
  - UDP: δεν αποστέλλει κάποιο μήνυμα ICMP. Επιστρέφει 0.
  - Ασυνδεσμικό SCTP: δεν υποστηρίζεται αυτή η λειτουργία. Επιστρέφει -1.
- Στις υπόλοιπες περιπτώσεις, επιστρέφει 0 για επιτυχία, -1 για αποτυχία.

Αποστολή & λήψη δεδομένων

Ροή δεδομένων

- Καθώς τα sockets είναι «αρχεία», μπορούμε να χρησιμοποιήσουμε τις ίδιες συναρτήσεις για Ε/Ε (`read(2)`, `write(2)` κτλ.):

```
ssize_t read(int d, void *buf, size_t nbytes);
ssize_t write(int d, void *buf, size_t nbytes);
```

  - `d`: ο socket descriptor (από την `connect(2)` ή την `accept(2)`).
  - `buf`: δείκτης σε χώρο μνήμης από όπου θα διαβασθούν τα προς αποστολή δεδομένα ή θα γραφούν τα ληφθέντα δεδομένα.
  - `nbytes`: το μέγεθος του `buf` σε bytes.
  - Επιστρέφουν τον αριθμό των bytes που διαβάστηκαν/γράφηκαν από/στο socket, 0 για EOF και -1 για αποτυχία.
  - Οι κλήσεις αυτές είναι συνήθως blocking:
    → Η `read(2)` θα περιμένει έως ότου διαβάσει `nbytes` bytes.
    → Η `write(2)` θα περιμένει έως ότου γράψει `nbytes` bytes (η αποστολή θα γίνει σε δεύτερο χρόνο από τον πυρήνα).
  - Υπάρχει το ενδεχόμενο να διακοπούν, οπότε επιστρέφουν -1 και θέτουν το `errno` στην τιμή `EINTR`.
  - Ίσως διαβάσουν/γράψουν λιγότερα από `nbytes` bytes, ανάλογα με την κατάσταση του δικτύου...

Αποστολή δεδομένων σε πακέτα

send(2)

```
ssize_t send(int s, const void *msg, size_t len, int flags);
```

- Χρησιμοποιείται μόνο όταν το socket είναι σε συνδεδεμένη κατάσταση.
- `s`: ο socket descriptor.
- `msg`: δείκτης σε θέση μνήμης, μεγέθους `len` απ' όπου θα διαβασθούν τα δεδομένα.
- `flags`: καθορίζει ειδικές απαιτήσεις επικοινωνίας.
  - `MSG_OOB`: στείλε «επείγοντα» δεδομένα.
  - `MSG_DONTROUTE`: παρέκαμψε την δρομολόγηση και στείλε τα δεδομένα κατευθείαν στο interface.
  - `MSG_EOR`: τέλος αποστολής εγγραφής, όταν το πρωτόκολλο υποστηρίζουν κάτι αντίστοιχο.
  - `MSG_EOF`: κάλεσε τη `shutdown(2)` για την πλευρά μας και ειδοποίησε την άλλη πλευρά.
  - `MSG_NOSIGNAL`: μη στείλεις `SIGPIPE` αν το socket είναι κλειστό.
- Επιστρέφει τον αριθμό των bytes που γράφηκαν στο socket ή -1 για σφάλμα.
- Αν τα δεδομένα δε χωρούν σε ένα πακέτο, επιστρέφεται σφάλμα `EMSGSIZE`.

Αποστολή δεδομένων σε πακέτα

sendto(2)

```
ssize_t sendto(int s, const void *msg, size_t len, int
flags, constr struct sockaddr *to, socklen_t tolen);
```

- Χρησιμοποιείται οποτεδήποτε.
- `s`: ο socket descriptor.
- `msg`: δείκτης σε θέση μνήμης, μεγέθους `len` απ' όπου θα διαβασθούν τα δεδομένα.
- `to`: δείκτης σε δομή τύπου `struct sockaddr`, μεγέθους `tolen`, που περιέχει πληροφορίες για τον παραλήπτη των δεδομένων. Μπορεί να είναι `NULL` αν έχουμε προηγουμένως καλέσει την `connect(2)`.
- `flags`: καθορίζει ειδικές απαιτήσεις επικοινωνίας, όπως και στην `send(2)`.
- Επιστρέφει τον αριθμό των bytes που γράφηκαν στο socket ή -1 για σφάλμα.
- Αν τα δεδομένα είναι πολλά για να σταλούν ως ένα πακέτο, επιστρέφεται σφάλμα `EMSGSIZE`.

Αποστολή δεδομένων σε πακέτα

sendmsg(2)

```
ssize_t sendmsg(int s, const struct msghdr *msg, int flags);
```

- Χρησιμοποιείται οποτεδήποτε.
- `s, flags` και τιμή επιστροφής όπως και στην `send(2)`.
- `msg`: δείκτης σε δομή τύπου `struct msghdr`.
  - Το πεδίο `msg_name` είναι δείκτης σε δομή τύπου `struct sockaddr` μεγέθους `msg_namelen` και περιέχει την διεύθυνση του παραλήπτη, αν δεν έχουμε καλέσει προηγουμένως την `connect(2)`.
  - Τα πεδία `msg_iov` και `msg_control` χρησιμοποιούνται για «προχωρημένες» λειτουργίες.

struct msghdr

```
struct msghdr {
    void*         msg_name;       /* optional address */
    socklen_t     msg_namelen;    /* size of msg_name */
    struct iovec* msg_iov;        /* scatter/gather array */
    int           msg_iovlen;     /* #elements in msg_iov */
    void*         msg_control;    /* ancillary data */
    socklen_t     msg_controllen; /* size of msg_control */
    int           msg_flags;      /* MSG_EOR, MSG_TRUNC,
                                     MSG_OOB, MSG_CTRUNC */
};
```

Λήψη δεδομένων σε πακέτα

`recv(2)`

```
ssize_t recv(int s, void *buf, size_t len, int flags);
```

- Χρησιμοποιείται μόνο όταν το socket είναι σε συνδεδεμένη κατάσταση.
- `s`: ο socket descriptor.
- `buf`: δείκτης σε θέση μνήμης, μεγέθους `len` όπου θα γραφούν τα δεδομένα.
- `flags`: καθορίζει ειδικές απαιτήσεις επικοινωνίας.
  - `MSG_OOB`: διάβασε «επείγοντα» δεδομένα.
  - `MSG_PEEK`: διάβασε δεδομένα χωρίς να τα βγάλεις από τον buffer του socket.
  - `MSG_WAITALL`: περίμενε να έρθουν όλα τα δεδομένα (ή επέστρεψε κατάλληλο σφάλμα στο `errno` αν αποτύχεις).
  - `MSG_DONTWAIT`: non-blocking επικοινωνία.
- Επιστρέφει τον αριθμό των bytes που διαβάστηκαν από το socket ή -1 για σφάλμα.

Λήψη δεδομένων σε πακέτα

recvfrom(2)/recvmsg(2)

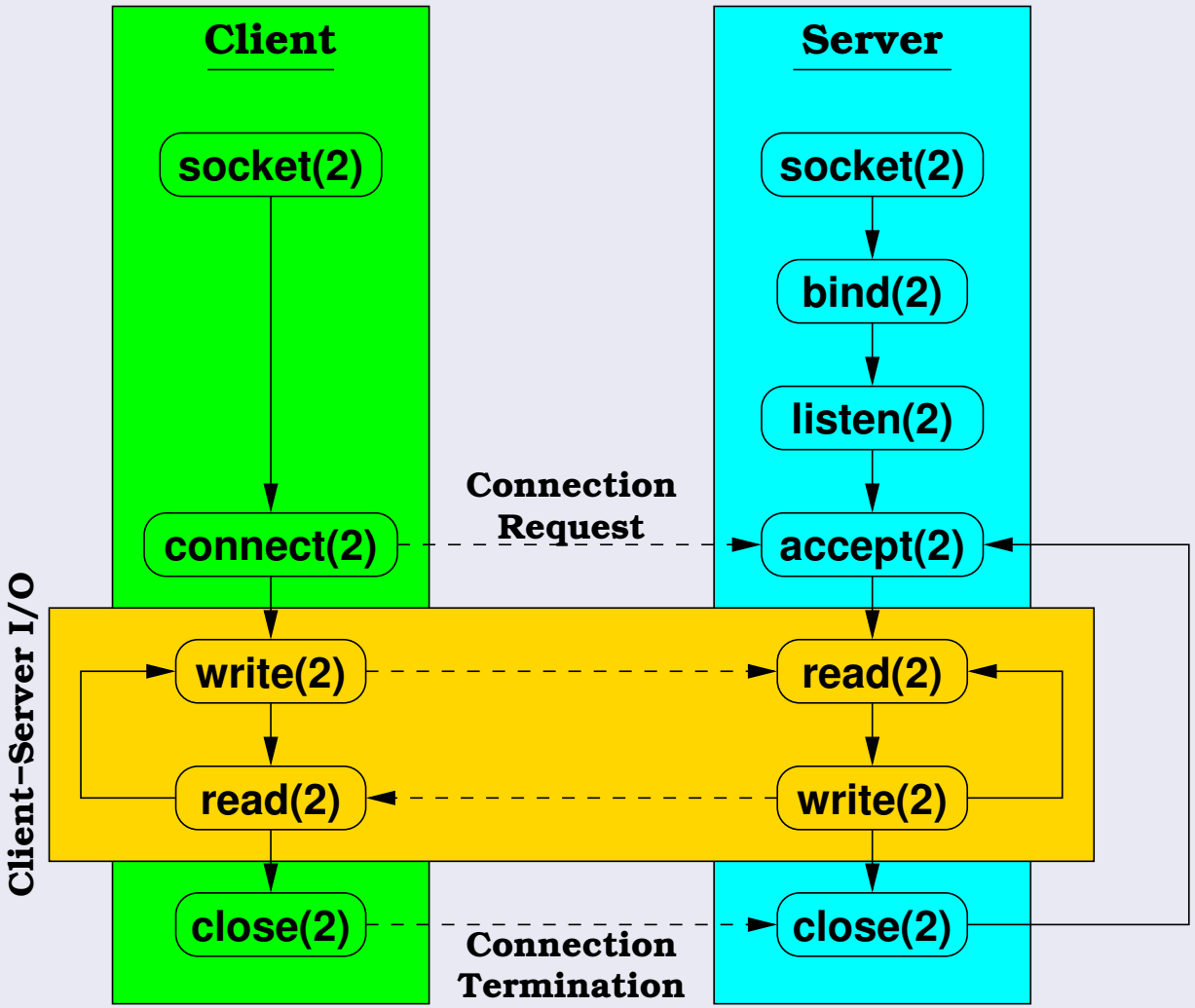
```
ssize_t recvfrom(int s, void *buf, size_t len, int flags,
                 struct sockaddr *from, socklen_t *fromlen);
ssize_t recvmsg(int s, struct msghdr *msg, int flags);
```

- Χρησιμοποιούνται οποτεδήποτε.
- `s`, `buf`, `flags` και τιμή επιστροφής όπως στην `recv(2)`.
- `msg` όπως και στην `sendmsg(2)`.
- Το πεδίο `from` είναι δείκτης σε δομή δεδομένων τύπου `struct sockaddr` και περιέχει την διεύθυνση απ' όπου περιμένουμε να λάβουμε δεδομένα. Μπορεί να είναι `NULL` αν έχουμε προηγουμένως καλέσει την `connect(2)`. Διαφορετικά, αν δεν έχουμε προηγουμένως καλέσει την `connect(2)`, όταν η συνάρτηση επιστρέψει θα έχει πληροφορία για τον αποστολέα των δεδομένων.
- Το `*fromlen` πρέπει αρχικά να περιέχει το μέγεθος του `from`, ενώ όταν επιστρέψει η συνάρτηση θα έχει το τελικό μέγεθος του τελευταίου.

Παραδείγματα

Συνδεσμοστραφής επικοινωνία

Βασική δομή



Συνδεσμοστραφής επικοινωνία – Πελάτης (1/2)

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void error(char *str) {
   errno ? perror(str) : fprintf(stderr, "%s\n", str);
   exit(1);
}

int main(int argc, char **argv) {
   int sd;
   char *ep;
   unsigned long port;
   struct sockaddr_in sin;
   const char *hello = "hello\n";
   const int hello_len = strlen(hello) + 1;

   if (argc != 3)
      error("Usage: test-client <ip addr> <port>");
```

Συνδεσμοστραφής επικοινωνία – Πελάτης (2/2)

```
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;

    port = strtoul(argv[2], &ep, 10);
    if (argv[2][0] == '\0' || errno != 0 || ep == NULL || *ep != '\0' || port > 0xffff)
        error("invalid port");
    sin.sin_port = htons(port);

    if (inet_pton(AF_INET, argv[1], &sin.sin_addr) != 1)
        error("unrecognizable address");

    if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        error("socket");

    if (connect(sd, (struct sockaddr *)&sin, sizeof(sin)) < 0)
        error("connect");

    if (write(sd, hello, hello_len) != hello_len)
        error("write");

    close(sd);
    return 0;
}
```

Συνδεσμοστραφής επικοινωνία – Εξυπηρετητής (1/3)

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void error(char *str) {
   errno ? perror(str) : fprintf(stderr, "%s\n", str);
   exit(1);
}

int main(int argc, char *argv[]) {
   char c, *ep;
   int sd, cd, tmp;
   unsigned long port;
   struct sockaddr_in sin;

   if (argc != 2)
      error("Usage: test-server <port>");
```

Συνδεσμοστραφής επικοινωνία – Εξυπηρετητής (2/3)

```
memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = htonl(INADDR_ANY);

port = strtoul(argv[1], &ep, 10);
if (argv[1][0] == '\0' || errno != 0 || ep == NULL || *ep != '\0' || port > 0xffff)
   error("invalid port");
sin.sin_port = htons(port);

if ((sd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
   error("socket");

if (bind(sd, (struct sockaddr*)&sin, sizeof(sin)) < 0)
   error("bind");

if (listen(sd, 5) < 0)
   error("listen");
```

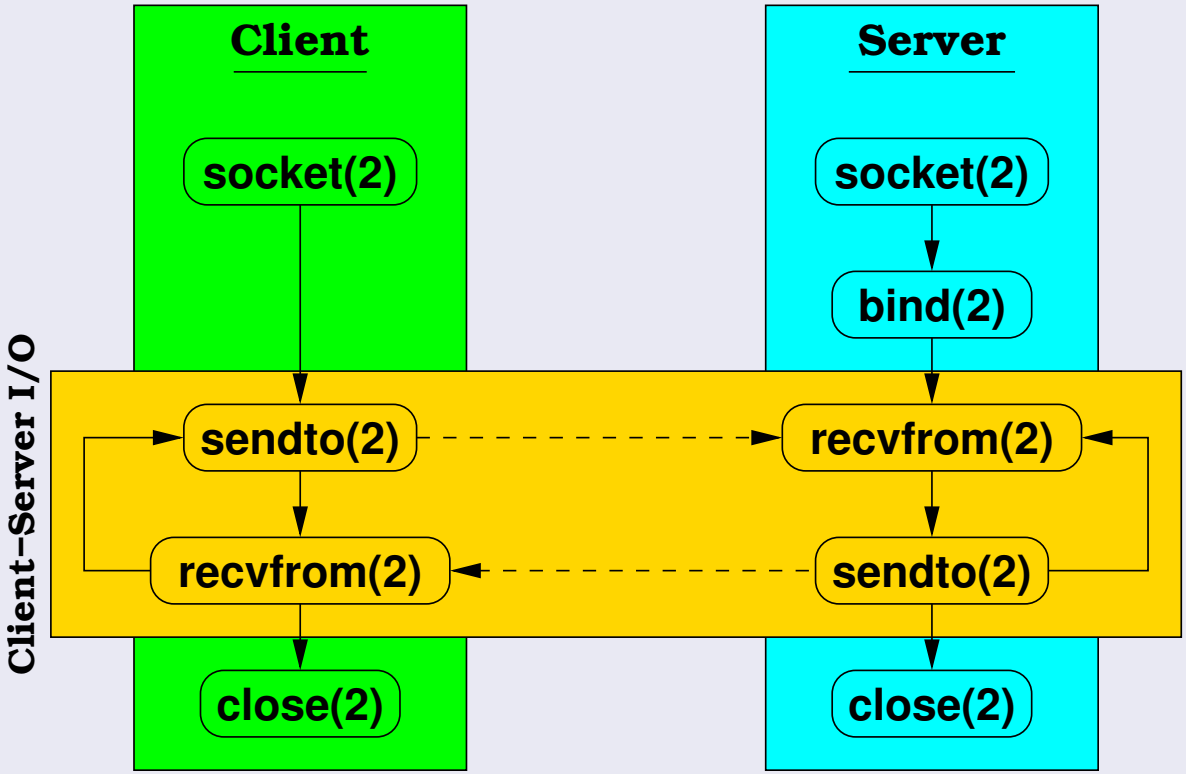
Συνδεσμοστραφής επικοινωνία – Εξυπηρετητής (3/3)

```
    while (1) {
        if ((cd = accept(sd, NULL, NULL)) < 0)
            error("accept");
        for (;;) {
            if ((tmp = read(cd, &c, sizeof(char))) == 0)
                break;
            if (tmp < 0) {
                if (errno == EINTR)
                    continue;
                else
                    error("read");
            }
            putchar(c);
            if (c == '\n')
                break;
        }
        if (close(cd) < 0)
            error("close");
    }

    if (close(sd) < 0)
        error("close");
    return 0;
}
```

Ασυνδεσμική επικοινωνία

Βασική δομή



Ασυνδεσμική επικοινωνία – Πελάτης (1/2)

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void error(char *str) {
   errno ? perror(str) : fprintf(stderr, "%s\n", str);
   exit(1);
}

int main(int argc, char **argv) {
   int sd;
   char *ep;
   unsigned long port;
   struct sockaddr_in sin;
   const char hello[16] = "hello\n";

   if (argc != 3)
      error("Usage: test <ip addr> <port>");
```

Ασυνδεσμική επικοινωνία – Πελάτης (2/2)

```
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;

    port = strtoul(argv[2], &ep, 10);
    if (argv[2][0] == '\0' || errno != 0 || ep == NULL || *ep != '\0' || port > 0xffff)
        error("invalid port");
    sin.sin_port = htons(port);

    if (inet_pton(AF_INET, argv[1], &sin.sin_addr) != 1)
        error("unrecognizable address");

    if ((sd = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
        error("socket");

    if (sendto(sd, hello, sizeof(hello), 0, (struct sockaddr*)&sin, sizeof(sin)) < 0)
        error("sendto");

    if (close(sd) < 0)
        error("close");

    return 0;
}
```

Ασυνδεσμική επικοινωνία – Εξυπηρετητής (1/2)

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>

void error(char *str) {
   errno ? perror(str) : fprintf(stderr, "%s\n", str);
   exit(1);
}

int main(int argc, char *argv[]) {
   int sd;
   char *ep, buf[16];
   unsigned long port;
   struct sockaddr_in sin;

   if (argc != 2)
      error("Usage: test-server <port>");
```

Ασυνδεσμική επικοινωνία – Εξυπηρετητής (2/2)

```
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = htonl(INADDR_ANY);

    port = strtoul(argv[1], &ep, 10);
    if (argv[1][0] == '\0' || errno != 0 || ep == NULL || *ep != '\0' || port > 0xffff)
       error("invalid port");
    sin.sin_port = htons(port);

    if ((sd = socket(PF_INET, SOCK_DGRAM, 0)) < 0)
       error("socket");

    if (bind(sd, (struct sockaddr*)&sin, sizeof(sin)) < 0)
       error("bind");

    while (1) {
       memset(buf, 0, sizeof(buf));
       if (recvfrom(sd, buf, sizeof(buf), 0, NULL, NULL) < 0)
          error("recvfrom");
       printf("%s", buf);
    }
    if (close(sd) < 0)
       error("close");
    return 0;
}
```

Στο επόμενο μάθημα...

- Μεταφέρσιμος προγραμματισμός με sockets.
- Σχεδιαστικά μοντέλα δικτυακών εφαρμογών.
- Εισαγωγή σε ζητήματα υλοποίησης πελατών και εξυπηρετητών.