Προγραμματισμός Δικτύων – Ε-01

4η Διάλεξη

Διδάσκων: Νίκος Ντάρμος

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

Τμήμα Πληροφορικής
Πανεπιστήμιο Ιωαννίνων

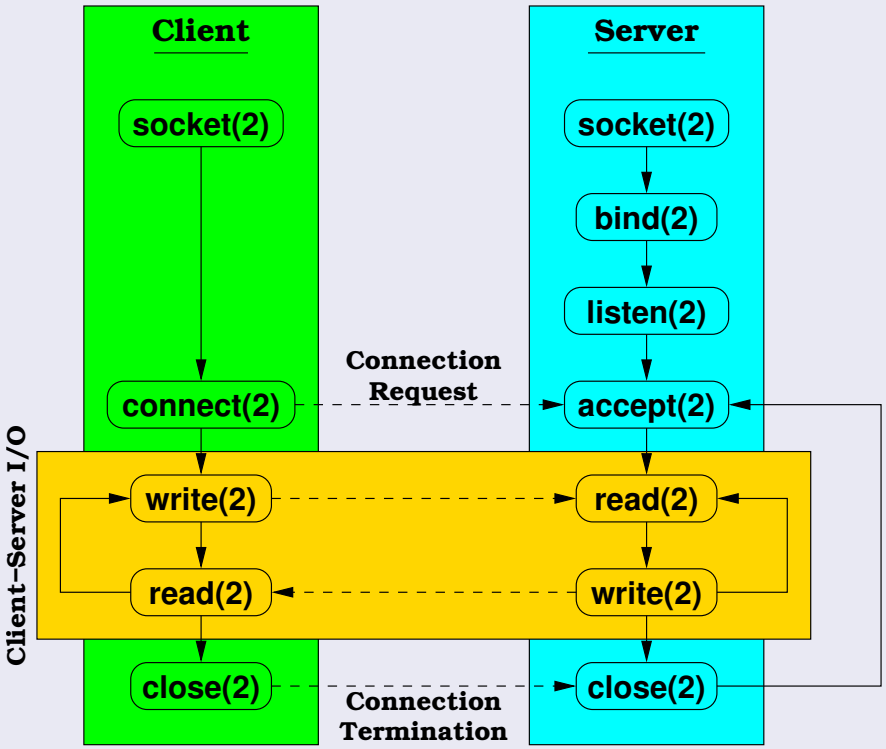
Στο σημερινό μάθημα...

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

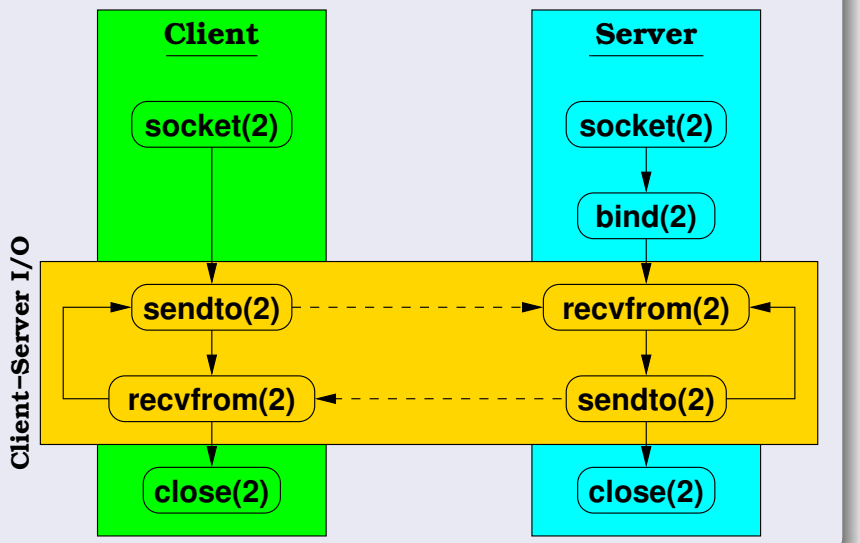
Μεταφέρσιμος προγραμματισμός με sockets

Βασική δομή προγραμμάτων

Συνδεσμοστραφής επικοινωνία



Ασυνδεσμική επικοινωνία



Βασική δομή προγραμμάτων

Πελάτης

1. Αναγνώριση της τοποθεσίας του εξυπηρετητή.
   - Διεύθυνση IP ή συμβολικό όνομα κόμβου.
   - Αριθμός θύρας δικτυακής επικοινωνίας.
   - Μετατροπή δεδομένων σε αναπαράσταση δικτύου.
2. Δημιουργία socket πελάτη.
3. Ανάθεση αριθμού θύρας στο socket πελάτη.
4. *Σύνδεση στο socket του εξυπηρετητή.*
5. Ανταλλαγή δεδομένων με τον εξυπηρετητή.
6. Τερματισμός σύνδεσης.

Βασική δομή προγραμμάτων

Εξυπηρετητής

1. Αναγνώριση της τοποθεσίας του εξυπηρετητή.
   - Διεύθυνση IP ή συμβολικό όνομα κόμβου.
   - Αριθμός θύρας δικτυακής επικοινωνίας.
   - Μετατροπή δεδομένων από αναπαράσταση κόμβου σε αναπαράσταση δικτύου.
2. Δημιουργία socket εξυπηρετητή.
3. Δέσμευση του socket στην προκαθορισμένη διεύθυνση και θύρα.
4. *Δημιουργία ουράς αναμονής στο socket του εξυπηρετητή.*
5. *Αποδοχή επόμενης αίτησης σύνδεσης πελάτη.*
6. Ανταλλαγή δεδομένων με τον πελάτη.
7. Τερματισμός σύνδεσης.

Προβληματικές περιοχές

Προσοχή!

Η διαλειτουργικότητα είναι ευθύνη του προγραμματιστή!

- Διαφορετικοί τύποι δεδομένων ανάλογα την αρχιτεκτονική & το λειτουργικό σύστημα του κόμβου.
- Διαφορετικές δομές δεδομένων και κλήσεις συστήματος, ανάλογα το επιλεγμένο πρωτόκολλο επιπέδου διαδικτύου.
  - Μετατροπή διεύθυνσης κόμβου σε δυαδική αναπαράσταση.
    - Μετατροπή διεύθυνσης IP από συμβολοσειρά σε δυαδική αναπαράσταση.
    - Μετατροπή συμβολικού ονόματος κόμβου από συμβολοσειρά σε δυαδική αναπαράσταση.
    - Μετατροπή των παραπάνω σε αναπαράσταση δικτύου.
  - Μετατροπή θύρας/υπηρεσίας σε δυαδική αναπαράσταση.
    - Μετατροπή αριθμού θύρας από συμβολοσειρά σε δυαδική αναπαράσταση.
    - Μετατροπή συμβολικού ονόματος υπηρεσίας από συμβολοσειρά σε δυαδική αναπαράσταση.
    - Μετατροπή των παραπάνω σε αναπαράσταση δικτύου.
  - Δημιουργία socket συγκεκριμένης οικογένειας πρωτοκόλλων.
  - Υποχρησιμοποίηση διαθέσιμων διευθύνσεων κόμβου.

Βασικοί τύποι δεδομένων

- Κύρια προβλήματα: **δυαδική αναπαράσταση** και **μέγεθος** του τύπου δεδομένων.
- Δυαδική αναπαράσταση:
  - Big-endian ή little-endian.
  - Αλλάζει ανάλογα την αρχιτεκτονική του επεξεργαστή & το λειτουργικό σύστημα.
  - Παράδειγμα:
    - x86, x86_64: little-endian.
    - Sun Sparc <V9, PowerPC: big-endian.
    - Alpha AXP, Sun Sparc $\geq$V9: επιλέγει το firmware ή/και το λειτουργικό.
    - Δίκτυο: πάντα big-endian.
- Μέγεθος:
  - Τα πρότυπα ορίζουν κάποια **ελάχιστα** μεγέθη για τους βασικούς τύπους.
  - Το πραγματικό μέγεθος αλλάζει ανάλογα την αρχιτεκτονική του επεξεργαστή & το λειτουργικό σύστημα.
  - Παράδειγμα:
    - `long int`:
      16-bit CPU: $\Rightarrow$ 2 bytes (Intel Pentium 8086--80286).
      32-bit CPU: $\Rightarrow$ 4 bytes (Intel Pentium $\leq$4, Motorola 68k, ARM, Sun Sparc <V9).
      64-bit CPU: $\Rightarrow$ 8 bytes (Alpha AXP, Intel Pentium $\geq$4, Sun Sparc $\geq$V9).

Βασικοί τύποι δεδομένων

Αναπαράσταση

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

Μέγεθος

- Τα πρότυπα ορίζουν βασικούς τύπους δεδομένων σταθερού/προκαθορισμένου μεγέθους.
  - `int_16_t, int_32_t, int_64_t, u_int_16t, ...`
  - `<sys/types.h>, <inttypes.h>, ...`
- Το API των sockets ορίζει βασικούς τύπους δεδομένων σταθερού/προκαθορισμένου μεγέθους.
  - `sa_family_t, in_port_t, ...`
  - `<sys/types.h>, <sys/socket.h>, <netinet/in.h>, ...`
- Ο τελεστής `sizeof` επιστρέφει το μέγεθος του τύπου σε bytes.
- Καλό είναι, όταν μεταφέρουμε δεδομένα, να συμπεριλαμβάνουμε και το μέγεθός τους.

Εξάρτηση από το πρωτόκολλο επιπέδου διαδικτύου

- Ο κώδικάς μας μπορεί να χρησιμοποιηθεί σε πολλά διαφορετικά συστήματα.
  - Κόμβοι μόνο με IPv4 διευθύνσεις.
  - Κόμβοι μόνο με IPv6 διευθύνσεις.
  - Κόμβοι με «*dual IP stack*» (υποστηρίζουν IPv4 και IPv6 ταυτόχρονα).
- Οι διευθύνσεις των κόμβων και των υπηρεσιών αποθηκεύονται σε δομές δεδομένων διαφορετικές για κάθε πρωτόκολλο επιπέδου διαδικτύου.
- Πολλές συναρτήσεις του BSD sockets API παίρνουν ως όρισμα την «οικογένεια πρωτοκόλλων» ($\longrightarrow$ πρωτόκολλο επιπέδου διαδικτύου).
- Πολλοί κόμβοι έχουν περισσότερες από μια διευθύνσεις.
- **Πως μπορούμε να κάνουμε το πρόγραμμά μας ανεξάρτητο του πρωτοκόλλου στο επίπεδο διαδικτύου;**

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
```

Μέγεθος των δομών `struct sockaddr*`

- Οι συναρτήσεις του BSD sockets API παίρνουν ως όρισμα δείκτη σε `struct sockaddr`. Πάντα όμως χρησιμοποιούμε κάποια από τις άλλες δομές.
- **Πρόβλημα:** πως γνωρίζει η συνάρτηση το μέγεθος ή/και τον τύπο της δομής που χρησιμοποιήσαμε;
- **Λύση Α΄:** Δίνουμε και το μέγεθος της δομής ως όρισμα στη συνάρτηση.

```
struct sockaddr_in sin;
bind(..., (struct sockaddr*)sin, sizeof(sin));
```

- **Λύση Β΄:** Η συνάρτηση ελέγχει το πεδίο `sa_family` και αποφασίζει ανάλογα.

```
struct sockaddr* sa;
int salen;
if (sa->sa_family == AF_INET)
    salen = sizeof(struct sockaddr_in);
else if (sa->sa_family == AF_INET6)
    salen = sizeof(struct sockaddr_in6);
else if (...)
```

- **Λύθηκε το πρόβλημα;**

Μέγεθος των δομών `struct sockaddr*`

- Κάποιες συναρτήσεις «επιστρέφουν» δείκτες σε δομή `sockaddr`. Παράδειγμα:

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

- **Πρόβλημα:** Πως ξέρουμε τι είδους δομή θα μας επιστρέψει για να έχουμε δεσμεύσει τον κατάλληλο χώρο;
- **Λύση:** Χρησιμοποιούμε την δομή `sockaddr_storage`.

```
struct sockaddr_storage ss;
socklen_t sslen = sizeof(ss);
recvfrom(..., (struct sockaddr *)&ss, &sslen);
```

Ανάλυση ονομάτων κόμβων και υπηρεσιών

Κλήσεις που πρέπει να ξεχάσουμε!

- `inet_ntoa(3)`, `inet_aton(3)`, `inet_ntop(3)`, `inet_pton(3)`,
- `inet_netof(3)`, `inet_lnaof(3)`, `inet_makeaddr(3)`, `inet_addr(3)`, `inet_network(3)`.
- `gethostbyname(3)`, `gethostbyname2(3)`, `gethostbyaddr(3)`.
- `getservbyname(3)`, `getservbyport(3)`.

Ανάλυση ονομάτων κόμβων και υπηρεσιών

- Οι «παλαιότερες» συναρτήσεις ανάλυσης ονομάτων κόμβων δουλεύουν **μόνο για διευθύνσεις IPv4**.

```
struct hostent* gethostbyname(const char *name);
int inet_aton(const char *cp, struct in_addr *pin);
in_addr_t inet_addr(const char *cp);
in_addr_t inet_network(const char *cp);
char* inet_ntoa(struct in_addr in);
in_addr_t inet_lnaof(struct in_addr in);
in_addr_t inet_netof(struct in_addr in);
```

- Οι «νεώτερες» συναρτήσεις ανάλυσης ονομάτων κόμβων **δέχονται ως όρισμα το address family**.

```
struct hostent* gethostbyname2(const char *name, int af);
struct hostent* gethostbyaddr(const void *addr, socklen_t len,
        int af);
const char* inet_ntop(int af, const void *src, char *dst,
        socklen_t size);
int inet_pton(int af, const char *src, void *dst);
```

Ανάλυση ονομάτων κόμβων και υπηρεσιών

- Οι κλασσικές συναρτήσεις μετατροπής συμβολικών ονομάτων κόμβων και διευθύνσεων IP σε δυαδική αναπαράσταση **περιπλέκουν πολύ τον κώδικα**.

```
1   char *ep;
2   struct hostent *hp;
3   struct sockaddr_in sin;
4
5   hp = gethostbyname(hostname);
6   if (hp) {
7       if (sizeof(sin.sin_addr) != hp->h_length) {
8           fprintf(stderr, "unexpected address length\n");
9           exit(1);
10      }
11      memcpy(sin.sin_addr, hp->h_addr, sizeof(sin.sin_addr));
12  } else {
13      if (inet_pton(AF_INET, hostname, &sin.sin_addr) != 1) {
14          fprintf(stderr, "%s: invalid hostname\n");
15          exit(1);
16      }
17  }
```

Ανάλυση ονομάτων κόμβων και υπηρεσιών

- Οι κλασσικές συναρτήσεις μετατροπής συμβολικών ονομάτων υπηρεσιών και αριθμών θύρας σε δυαδική αναπαράσταση **περιπλέκουν πολύ τον κώδικα**.

```
1   unsigned long ul;
2   struct sockaddr_in sin;
3   struct servent *sp;
4   char *ep = NULL;
5
6   sp = getservbyname(servname, protoname);
7   if (sp) {
8       sin.sin_port = sp->s_port;
9   } else {
10      errno = 0;
11      ul = strtoul(servname, &ep, 10);
12      if (servname[0] == '\0' || errno != 0 || !ep ||
13              *ep != '\0' || ul > 0xffff) {
14          fprintf(stderr, "%s: invalid servname\n");
15          exit(1);
16      }
17      sin.sin_port = htons(ul & 0xffff);
18  }
```

Η απάντηση: `getaddrinfo(3)`

- Δουλεύει **ανεξάρτητα από το πρωτόκολλο επιπέδου διαδικτύου**.
- Αντικαθιστά **όλες τις προηγούμενες συναρτήσεις** και **απλοποιεί σημαντικά τον κώδικα**.
- Είναι **ιδιαίτερα ευέλικτη**.
  - Με `hints.ai_flags = AI_NUMERICHOST` μπορούμε να αποφύγουμε αναζητήσεις στο DNS.
  - Με `hostname = NULL` μας δίνει μία δομή `addrinfo` που αντιστοιχεί στο loopback.
  - Με `hints.ai_flags = AI_PASSIVE` μας δίνει μία δομή `addrinfo` που αντιστοιχεί στο wildcard address (0.0.0.0, ::), κατάλληλη για να δημιουργήσουμε socket εξυπηρετητή.
  - Χειρίζεται όλες τις διευθύνσεις IPv4 **και** IPv6 (συμπεριλαμβανομένων και scoped διευθύνσεων).

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

```
struct addrinfo {
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    socklen_t ai_addrlen;
    struct sockaddr *ai_addr;
    char *ai_canonname;
    struct addrinfo *ai_next;
};
```

Η απάντηση: getaddrinfo(3)

```
1   int error;
2   struct addrinfo hints, *res;
3   static struct sockaddr_storage ss;
4   memset(&hints, 0, sizeof(hints));
5   hints.ai_socktype = SOCK_STREAM;
6   if(getaddrinfo(hostname, servname, &hints, &res)) {
7       fprintf(stderr, "%s/%s: %s\n", hostname, servname,
8               gai_strerror(error));
9       exit(1);
10  }
11  if (res->ai_addrlen > sizeof(ss)) {
12      fprintf(stderr, "sockaddr too large\n");
13      exit(1);
14  }
15  memcpy(&ss, res->ai_addr, res->ai_addrlen);
16  freeaddrinfo(res);
```

Ανάλυση ονομάτων κόμβων και υπηρεσιών

- Τα παραπάνω ισχύουν και για τη μετατροπή από δυαδική αναπαράσταση σε συμβολοσειρά.

```
1  struct in_addr in;
2  struct hostent *hp;
3  struct servent *sp;
4  struct sockaddr_in sin;
5  char hbuf[INET_ADDRSTRLEN];
6
7  printf("address: %s\n", inet_ntoa(in)); /* not thread safe */
8
9  if (inet_ntop(AF_INET, &in, buf, sizeof(buf)) != 1) { /* thread safe */
10     fprintf(stderr, "could not translate address\n");
11     exit(1);
12 }
13 printf("address: %s\n", hbuf);
14
15 if (!(hp = gethostbyaddr(&in, sizeof(in), AF_INET))) { /* not thread safe */
16     fprintf(stderr, "could not reverse-lookup address\n");
17     exit(1);
18 }
19 printf("FQDN: %s\n", hp->h_name);
20
21 if ((sp = getservbyport(sin.sin_port, "tcp")))
22     printf("port: %s\n", sp->s_name);
23 else
24     printf("port: %u\n", ntohs(sin.sin_port));
```

Η απάντηση: `getnameinfo(3)`

- Δουλεύει ανεξάρτητα από το πρωτόκολλο επιπέδου διαδικτύου.
- Αντικαθιστά όλες τις σχετικές παλαιότερες συναρτήσεις και απλοποιεί σημαντικά τον κώδικα.
- Το τελευταίο όρισμα καθορίζει την συμπεριφορά της.
  - Με `NI_NUMERICHOST` ή/και `NI_NUMERICSERV` μπορούμε να αποφύγουμε αναζητήσεις στο DNS και στο αρχείο των υπηρεσιών αντίστοιχα.
  - Με `NI_NAMEREQD` επιστρέφει συμβολικό όνομα, αλλιώς λάθος.
  - Χειρίζεται όλες τις διευθύνσεις IPv4 **και** IPv6 (συμπεριλαμβανομένων και scoped διευθύνσεων IPv6).

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

Η απάντηση: getnameinfo(3)

```
1  struct sockaddr *sa;
2  int salen, error;
3  char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];
4
5  if (getnameinfo(sa, salen, hbuf, sizeof(hbuf), sbuf, sizeof(sbuf), 0)) {
6      fprintf(stderr, "error: %s\n", gai_strerror(error));
7      exit(1);
8  }
9  printf("addr: %s port: %s\n", hbuf, sbuf);
10
11 if(getnameinfo(sa, salen, hbuf, sizeof(hbuf), sbuf, sizeof(sbuf),
12         NI_NUMERICHOST | NI_NUMERICSERV)) {
13     fprintf(stderr, "error: %s\n", gai_strerror(error));
14     exit(1);
15 }
16 printf("addr: %s port: %s\n", hbuf, sbuf)
17
18 if (getnameinfo(sa, salen, hbuf, sizeof(hbuf), NULL, 0, NI_NAMEREQD)) {
19     fprintf(stderr, "error: %s\n", gai_strerror(error));
20     exit(1);
21 }
22 printf("FQDN: %s\n", hbuf);
```

Κλήσεις συστήματος και οικογένεια πρωτοκόλλων

- Η `socket(2)` δέχεται ως όρισμα την οικογένεια πρωτοκόλλων διαδικτύου.
  - **Μπορούμε να δημιουργήσουμε ένα socket ανεξάρτητο του πρωτοκόλλου διαδικτύου;**
  - **Όχι**, αλλά μπορούμε να **μην χρησιμοποιούμε απευθείας** τα `AF_INET` και `AF_INET6`!
- Οι υπόλοιπές συναρτήσεις του BSD sockets API δέχονται ως όρισμα δείκτη σε δομή `sockaddr`.
  - **Πως θα αρχικοποιήσουμε τις δομές αυτές χωρίς να χρησιμοποιήσουμε τα AF_INET ή AF_INET6;**
- Πολλοί κόμβοι έχουν περισσότερες από μια διευθύνσεις.
  - **Τι γίνεται αν κάποιες από αυτές δεν είναι «λειτουργικές»;**
- **Η `getaddrinfo(3)` μας δίνει ό,τι χρειαζόμαστε.**

Πελάτης

```
1   int error, sd;
2   struct addrinfo hints, *res, *cur;
3   memset(&hints, 0, sizeof(hints));
4   hints.ai_socktype = SOCK_STREAM;
5   if(getaddrinfo(hostname, servname, &hints, &res)) {
6       fprintf(stderr, "%s/%s: %s\n", hostname, servname, gai_strerror(error));
7       exit(1);
8   }
9   if (res->ai_addrlen > sizeof(ss)) {
10      fprintf(stderr, "sockaddr too large\n");
11      exit(1);
12  }
13  for (cur = res; cur; cur = cur->ai_next) {
14      if ((sd = socket(cur->ai_family, cur->ai_socktype, cur->ai_protocol)) < 0)
15          continue;
16      if (!connect(sd, cur->ai_addr, cur->ai_addrlen))
17          break;
18      close(sd);
19  }
20  if (!cur) {
21      fprintf(stderr, "No destination to connect to\n");
22      exit(1);
23  }
24  freeaddrinfo(res);
```

Εξυπηρετητής

```
1   int error, sd;
2   struct addrinfo hints, *res, *cur;
3   memset(&hints, 0, sizeof(hints));
4   hints.ai_socktype = SOCK_STREAM;
5   hints.ai_flags = AI_PASSIVE;
6   if(getaddrinfo(NULL, servname, &hints, &res)) {
7       fprintf(stderr, "%s/%s: %s\n", hostname, servname, gai_strerror(error));
8       exit(1);
9   }
10  if (res->ai_addrlen > sizeof(ss)) {
11      fprintf(stderr, "sockaddr too large\n");
12      exit(1);
13  }
14  for (cur = res; cur; cur = cur->ai_next) {
15      if ((sd = socket(cur->ai_family, cur->ai_socktype, cur->ai_protocol)) < 0)
16          continue;
17      if (!bind(sd, cur->ai_addr, cur->ai_addrlen))
18          break;
19      close(sd);
20  }
21  if (!cur) {
22      fprintf(stderr, "Unable to bind socket\n");
23      exit(1);
24  }
25  freeaddrinfo(res);
26  if (listen(sd, 5)) {
27      perror("listen");
28      exit(1);
29  }
```

Σχεδιαστικά μοντέλα δικτυακών εφαρμογών

Εισαγωγή

- Το API των BSD sockets και το επίπεδο μεταφοράς εν γένει προσφέρει συνδέσεις από άκρο σε άκρο.
- Δεν καθορίζει όμως:
  - Πως κάποιος πελάτης θα βρει έναν αντίστοιχο εξυπηρετητή.
  - Πως θα συγχρονίζονται τα δύο άκρα επικοινωνίας.
  - Την μορφή και τη σημασιολογία των δεδομένων που θα ανταλλάξουν.
  - Την εσωτερική λειτουργία (business logic) των πελατών και των εξυπηρετητών.
- Η μορφή και η σημασιολογία των δεδομένων, καθώς και η εσωτερική λειτουργία, καθορίζονται σε μεγάλο βαθμό από το πρωτόκολλο του επιπέδου εφαρμογών που υλοποιούν τα δύο άκρα.
- Τα προβλήματα της εύρεσης εξυπηρετητή και του «συγχρονισμού» των δύο άκρων καθορίζονται από το **σχεδιαστικό μοντέλο της εφαρμογής**.
- Βασικά σχεδιαστικά μοντέλα:
  - Μοντέλο Πελάτη–Εξυπηρετητή (client--server).
  - Μοντέλο Πελάτη–Ουράς–Πελάτη (client--queue--client).
  - Μοντέλο Ομοτίμων Εταίρων (peer-to-peer).
  - Υβριδικά μοντέλα.

Μοντέλο Πελάτη–Εξυπηρετητή

- Το βασικότερο μοντέλο όλων.
  - Τα υπόλοιπα μοντέλα μπορούν να περιγραφούν ως ειδικές περιπτώσεις του.
- Υπάρχουν δύο διαφορετικοί ρόλοι:
  - Πελάτης: συνδέεται στον εξυπηρετητή προκειμένου να λάβει κάποια υπηρεσία.
  - Εξυπηρετητής: αναμένει συνδέσεις από πελάτες προκειμένου να τους παρέχει κάποια υπηρεσία.
- Η επικοινωνία έχει τη μορφή:
  αίτημα πελάτη $\rightarrow$ επεξεργασία στον εξυπηρετητή $\rightarrow$ απάντηση εξυπηρετητή.

Μοντέλο Πελάτη–Ουράς–Πελάτη

- Στο μοντέλο αυτό υπάρχουν δύο ρόλοι: πελάτες και παθητικές ουρές (passive queues).
- Βασική αρχή λειτουργίας:
  1. Ο πελάτης εισάγει σε μία ουρά ένα αίτημά του.
  2. Κάποιος άλλος πελάτης ελέγχει την ουρά για νέες αιτήσεις. Αφαιρεί μία, την επεξεργάζεται και αποθηκεύει το αποτέλεσμα στην ουρά.
  3. Ο αρχικός πελάτης ελέγχει την ουρά για απαντήσεις.
- Πολύ απλός τρόπος επικοινωνίας εφαρμογών.
- Υπερκεράστηκε από το μοντέλο ομοτίμων εταίρων.

Μοντέλο Ομοτίμων Εταίρων

- Κάθε κόμβος είναι ταυτόχρονα πελάτης και εξυπηρετητής.
- Όλοι οι κόμβοι είναι ίσοι.
  - Εκτελούν τον ίδιο κώδικα.
  - Είναι το ίδιο σημαντικοί (ασήμαντοι;)
  - Δεν υπάρχει single point of failure.
- Πολλά «παρακλάδια»:
  - «Καθαρά» (pure) και «υβριδικά» (hybrid).
  - Δομημένα και αδόμητα.
  - Κεντρικοποιημένα και αποκεντρικοποιημένα.

Σχεδιαστικά ζητήματα

Σχεδιαστικά ζητήματα

- Διαλειτουργικότητα σε διαφορετικούς κόμβους και με διαφορετικά πρωτόκολλα.
- Συνδεσμοστραφής ή ασυνδεσμική επικοινωνία;
- Διατήρηση κατάστασης στον εξυπηρετητή, στον πελάτη, σε κανέναν ή και στους δύο;
- Παραμετροποίηση πελάτη και εξυπηρετητή και επιλογή διευθύνσεων και θυρών.
- Ταυτοχρονισμός σε πελάτη και εξυπηρετητή και αποφυγή αδιεξόδων και λιμοκτονίας.
- Ζητήματα ασφαλείας και αντοχής σε λάθη και «αντίξοες συνθήκες».

Διαλειτουργικότητα

- Διαλειτουργικότητα σε διαφορετικές αρχιτεκτονικές.
- Διαλειτουργικότητα σε διαφορετικά λειτουργικά συστήματα/περιβάλλοντα.
- Διαλειτουργικότητα σε IPv4 και IPv6.

Συνδεσμοστραφής/ασυνδεσμική επικοινωνία

- Η επιλογή εξαρτάται από τις απαιτήσεις της εφαρμογής.
  - Εφαρμογές μετάδοσης φωνής και βίντεο, εφαρμογές πραγματικού χρόνου και γενικά εφαρμογές που μπορούν να ανεχθούν απώλεια πληροφοριών επιλέγουν ασυνδεσμική επικοινωνία.
  - Για μεταφορά μεγάλου όγκου δεδομένων, αλληλεπιδραστικές εφαρμογές και γενικά εφαρμογές που απαιτούν αξιοπιστία επιλέγουν συνδεσμοστραφή επικοινωνία.
- Χαρακτηριστικά ασυνδεσμικής επικοινωνίας:
  - «Γρήγορη»: δεν απαιτεί την εγκαθίδρυση σύνδεσης πριν την αποστολή δεδομένων και δεν προβλέπει μηνύματα αποδοχής ή αναμετάδοση πακέτων.
  - Μικρές απαιτήσεις σε πόρους συστήματος.
  - Ευκολία στην υλοποίηση **αν δεν απαιτείται αξιοπιστία**, αρκετά «επίπονη» διαφορετικά.
    - Απαιτείται σχεδιασμός και υλοποίηση μηχανισμών ανίχνευσης σφαλμάτων και ανάνηψης.
    - Απαιτείται καλή γνώση των πρωτοκόλλων του επιπέδου διαδικτύου.
- Χαρακτηριστικά συνδεσμοστραφούς επικοινωνίας:
  - Αυξημένες απαιτήσεις πόρων συστήματος.
  - Αξιόπιστη επικοινωνία.
    - Ευκολία υλοποίησης: ο χρήστης διαχειρίζεται μόνο την εγκαθίδρυση σύνδεσης,
    - Το επίπεδο μεταφοράς παρέχει μηχανισμούς ανίχνευσης σφαλμάτων και ανάνηψης.
    - «Αργή» όταν το δίκτυο είναι ασταθές/αναξιόπιστο.
  - Δε μπορεί να χρησιμοποιηθεί για εφαρμογές που απαιτούν broadcast και multicast.

Διατήρηση κατάστασης

- Η διατήρηση κατάστασης σχετικά με την επικοινωνία (state) μπορεί να γίνεται στο εξυπηρετητή, στον πελάτη, και στους δύο ή σε κανέναν.
- Η επιλογή και πάλι εξαρτάται από τις ανάγκες της εφαρμογής και του πρωτοκόλλου που υλοποιεί.
- Το state (μπορεί να) αφορά:
  - Πληροφορίες ταυτοποίησης του πελάτη ή/και του εξυπηρετητή.
  - Πληροφορίες για τα δεδομένα που έχουν μεταφερθεί.
  - Πληροφορίες για τα εκκρεμόντα αιτήματα.

Διατήρηση κατάστασης

- Λειτουργία χωρίς διατήρηση κατάστασης:
  - Μόνο στις απλούστερες των περιπτώσεων (π.χ. daytime, echo, κτλ.)
  - Κατάλληλη μόνο για εφαρμογές που δεν απαιτούν εγκαθίδρυση «συνεδρίας» (session).
  - Μικρότερη πολυπλοκότητα: καμία διατήρηση/ανταλλαγή στοιχείων σύνδεσης.
- Λειτουργία με διατήρηση κατάστασης στον πελάτη:
  - Ο πελάτης διατηρεί πληροφορία για την σύνδεσή του με τον εξυπηρετητή (π.χ. δεδομένα ταυτοποίησης, αιτήματα σε εκκρεμότητα, πληροφορίες συνεδρίας, κτλ.)
  - Ο πελάτης επισυνάπτει τα δεδομένα αυτά σε κάθε αίτημά του.
  - \+ Πιο ανθεκτική σε λάθη του δικτύου και καλύτερα κλιμακώσιμη.
  - – Μεγαλύτερες απαιτήσεις εύρους ζώνης δικτύου.
  - – Απαιτείται έλεγχος στον εξυπηρετητή για «λάθη» στα δεδομένα του πελάτη.
- Λειτουργία με διατήρηση κατάστασης στον εξυπηρετητή:
  - Ο εξυπηρετητής διατηρεί πληροφορία για τον κάθε πελάτη/σύνδεση.
  - \+ Καλύτερη απόδοση: δεν απαιτείται η αποστολή πληροφοριών σύνδεσης από τον πελάτη σε κάθε επικοινωνία.
  - – Απαιτείται έλεγχος στον εξυπηρετητή για τα δεδομένα και την ταυτότητα του κάθε πελάτη.
  - – Σαφώς μεγαλύτερες απαιτήσεις πόρων συστήματος στον εξυπηρετητή.

Παραμετροποίηση

- Επιλογή από τον χρήστη των βασικών παραμέτρων της σύνδεσης/υπηρεσίας.
  - Διεύθυνση/συμβολικό όνομα και αριθμός θύρας/όνομα υπηρεσίας εξυπηρετητή στον οποίο θα συνδεθεί ο πελάτης.
  - Διεύθυνση/συμβολικό όνομα και αριθμός θύρας/όνομα υπηρεσίας στο οποίο θα περιμένει εισερχόμενες αιτήσεις ο εξυπηρετητής.
  - Επιλογή πρωτοκόλλων επιπέδου διαδικτύου και μεταφοράς.
- Αυξάνει την χρησιμότητα των εφαρμογών πελάτη και εξυπηρετητή.
- Διευκολύνει τον έλεγχο και την αποσφαλμάτωσή τους.
- Μπορεί να γίνεται:
  - Απευθείας στον κώδικα.
  - Στην γραμμή εντολών.
  - Μέσω κάποιου αρχείου ρυθμίσεων.
  - Μέσω κάποιας άλλης (τοπικής ή απομακρυσμένης) υπηρεσίας.

Ταυτοχρονισμός

- Αφορά την δυνατότητα παράλληλης αποστολής ή/και επεξεργασίας αιτημάτων.
- Υπάρχει σε διάφορα επίπεδα:
  - Επίπεδο δικτύου: υπεύθυνα είναι τα πρωτόκολλα με την πολυπλεξία που παρέχουν.
  - Επίπεδο πελάτη: συνήθως απλά παρέχεται από το λειτουργικό σύστημα.
  - Επίπεδο εξυπηρετητή: σημαντικό κομμάτι του σχεδιασμού και της υλοποίησης μιας εφαρμογής.
- Ταυτοχρονισμός μπορεί να επιτευχθεί με:
  - Πολλαπλές διεργασίες (processes).
    - Κάθε διεργασία αποτελεί ακριβές αντίγραφο της αρχικής, με δικό της χώρο διευθύνσεων (address space, heap, stack, registers, κτλ.)
    - Πληρώνουμε το κόστος του context switching.
  - Πολλαπλά νήματα (threads).
    - Κάθε νήμα εκτελεί τον κώδικα μίας «συνάρτησης» και μοιράζεται των χώρο διευθύνσεων με τα υπόλοιπα νήματα της διεργασίας.
    - Θεωρητικά δεν υπάρχει κόστος για context switching· πρακτικά όμως υπάρχει...
  - Σχεδίαση βασισμένη σε συμβάντα (data/event-driven design) και ασύγχρονη Ε/Ε.
    - Ο πελάτης/εξυπηρετητής αποτελείται από μία διεργασία με ένα νήμα εκτέλεσης.
    - Δεν υπάρχει καθόλου context switching.
    - Ενδείκνυται για περιπτώσεις που ο χρόνος εξυπηρέτησης των αιτημάτων είναι μικρός ή στον οποίο επικρατεί το κόστος Ε/Ε.
  - Υβριδικές λύσεις που συνδυάζουν τα παραπάνω.

Ασφάλεια

- Από τη στιγμή που ένα σύστημα έχει περισσότερους από έναν χρήστες, υπάρχει απαίτηση για διαχωρισμό και ασφάλεια.
- Έτσι, προβλέπονται μηχανισμοί για:
  - Ταυτοποίηση και πιστοποίηση (authentication): επαλήθευση ταυτότητας πελάτη.
  - Εξουσιοδότηση (authorization): έλεγχος πρόσβασης και δικαιωμάτων πελάτη.
  - Εμπιστευτικότητα (confidentiality): μόνο ο αποστολέας και ο παραλήπτης έχουν πρόσβαση στα δεδομένα.
  - Ακεραιότητα (integrity): τα δεδομένα δεν αλλάζουν κατά τη μεταφορά τους.
  - Διαθεσιμότητα (availability): οι υπηρεσίες και τα δεδομένα θα είναι διαθέσιμα όταν ζητηθούν.
  - Μη αποκήρυξη (non-repudiation): καταγραφή συναλλαγών ώστε να μη μπορεί κάποιο μέλος να αρνηθεί τις πράξεις του αργότερα.

Στο επόμενο μάθημα...

Στο επόμενο μάθημα...

- Ταυτοχρονισμός πελάτη και εξυπηρετητή.
- Έλεγχος πρόσβασης, συγχρονισμός.