Мрежово програмиране

Програмиране на приложно ниво – I Iчаст

доц. д-р Йордан Денев
denev@fmi.uni-sofia.bg

Мрежово програмиране?

- Базово ниво- софтуерът на нива 1-4, част от операционната система.
- Приложно ниво– протоколите на 5 (приложно) ниво.
- Високо ниво – програмиране ЧРЕЗ протоколите на приложните нива.

Мрежова информация

```
#include <netdb.h>
struct hostent *gethostbyaddr(const void *addr, size_t
  len, int type)
*addr ->pointer to struct in_addr
len -> sizeof(struct in_addr),
type -> AF_INET
struct hostent *gethostbyname(const char *name)
struct hostent{
  char *h_name;          /*host name*/
  char **h_aliases;      /*list of aliases*/
  int h_addrtype;        /*address type (AF_INET)*/
  int h_length;          /*address length (4)*/
  char **h_addr_list     /*array of struct in_addr *s */
};
```

Подобрен клиент

Следващият код показва, как сървърът може да се специфицира с име:

```
/*  Make the necessary includes and set up the
   variables.  */
   #include <sys/types.h>
   #include <sys/socket.h>
   #include <stdio.h>
   #include <netinet/in.h>
   #include <arpa/inet.h>
   #include <unistd.h>
   #include <netdb.h>
```

```
int main(int argc, char *argv[] )
   {
        int sockfd;
        int client_len;
        int i;
        char* host;
        struct sockaddr_in client_address;
        struct hostent *hostinfo;
        int result;
        char ch = 'A';

        if(argc == 1)
            host = "localhost";
        else
            host = argv[1];
```

```
/*  Find the host address and report an error if
  none is found.  */
      hostinfo = gethostbyname(host);
      if(!hostinfo) {
          fprintf(stderr, "no host: %s\n", host);
          exit(1); }

  /*  Create a socket for the client.  */
      sockfd = socket(AF_INET, SOCK_STREAM, 0);

  /* Name the socket, as agreed with the server.  */
      client_address.sin_family = AF_INET;
      client_address.sin_addr = *(struct in_addr
  *)*hostinfo -> h_addr_list;
      client_address.sin_port =htons(9734);
      client_len = sizeof(client_address);
  /*  Now connect our socket to the server's socket
  and etc…*/
```

- Кой съм аз?

```
#include <unistd.h>

int gethostname (char* name, int length);
```

- Преобразуване на IP адреса към стандартна “dot” форма

```
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr addr);
```

❑ Пример – инфомация за сървера

```
/*  As usual, make the appropriate includes and
declare the variables.  */

#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    char *host, **names, **addrs;
    struct hostent *hostinfo;
```

```
/*  Set the host in question to the argument
  supplied with the getname call,
    or default to the user's machine.  */

    if(argc == 1) {
        char myname[256];
        gethostname(myname, 255);
        host = myname;
    }
    else
        host = argv[1];
```

```
*  Make the call to gethostbyname and report an
   error if no information is found.  */

    hostinfo = gethostbyname(host);
    if(!hostinfo) {
        fprintf(stderr, "cannot get info for host:
   %s\n", host);
        exit(1);
    }
```

```
/*  Display the hostname and any aliases it may
  have.  */

    printf("results for host %s:\n", host);
    printf("Name: %s\n", hostinfo -> h_name);
    printf("Aliases:");
    names = hostinfo -> h_aliases;
    while(*names) {
        printf(" %s", *names);
        names++;
    }
    printf("\n");
```

```
/*  Warn and exit if the host in question isn't an
   IP host.  */

    if(hostinfo -> h_addrtype != AF_INET) {
        fprintf(stderr, "not an IP host!\n");
        exit(1);
    }
```

```
/*  Otherwise, display the IP address(es).  */

    addrs = hostinfo -> h_addr_list;
    while(*addrs) {
        printf(" %s", inet_ntoa(*(struct in_addr
  *)*addrs));
        addrs++;
    }
    printf("\n");
    exit(0);
}
```

Сървер за много клиенти

- За всеки клиент се създава отделен процес.
- Пример:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>
```

```
int main()
{
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    int server_n = 0;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;

    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr =
  htonl(INADDR_ANY);
    server_address.sin_port = 9734;
    server_len = sizeof(server_address);
```

```
bind(server_sockfd, (struct sockaddr
  *)&server_address, server_len);

/*  Create a connection queue, ignore child exit
  details and wait for clients.  */

    listen(server_sockfd, 5);

    signal(SIGCHLD, SIG_IGN);
```

```
while(1) {
        char ch;

        printf("server waiting\n");

/*  Accept connection.  */

        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd,
            (struct sockaddr *)&client_address,
  &client_len);
```

```
*  Fork to create a process for this client*/
        server_n++;
        if(fork() == 0) {
/*  If we're the child, we can now read/write to the
   client on client_sockfd.
    The five second delay is a demonstration.  */
            while (read(client_sockfd, &ch, 1)!=0) {
               printf("server %d receives=
                              %c\n",server_n,ch);
            sleep(5);
      }
      printf ("server closes\n");
      close(client_sockfd);
      exit(0);
       }
```

```
/*  Otherwise, we must be the parent and our work
   for this client is finished.  */

        else {
            close(client_sockfd);
        }
    }
}
```

I/O Мултиплексиране

- Множество от дескриптори

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

FD_SET(int fd, fd_set *set); Add fd to the set.
FD_CLR(int fd, fd_set *set); Remove fd from the set.
FD_ISSET(int fd, fd_set *set); Return true if fd is
  in the set.
FD_ZERO(fd_set *set); Clear the set.
```

❑ Функция select

```
int select(int numfds, fd_set *readfds, fd_set
  *writefds,  fd_set *exceptfds,  struct timeval
  *timeout);
```

- `numfds` – проверяват се дескриптори в интервала [`0, numfds-1`];
- `readfds , writefds, exceptfds` - множествата се проверяват, за готовност за четене, за писане и за изключение;
-
```
struct timeval {                    // timeout
    int tv_sec; // seconds
    int tv_usec; // microseconds };
```

След изпълнение select връща:

-1 - при грешка;

0 - при timeout;

общ брой на дескритори, при които има готовност за дадената операция.

Освен това в множествата, сочени от **`readfds, writefds, exceptfds`** остават само дескрипторите на файлове с готовността за дадената операция

❑ Пример

```
/*
** select.c -- a select() demo
*/
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#define STDIN 0 // standard input descriptor
int main()
{       struct timeval tv;
        fd_set readfds;
        tv.tv_sec = 2;
        v.tv_usec = 500000;
```

```
        FD_ZERO(&readfds);
        FD_SET(STDIN, &readfds);
        // don't care about writefds and exceptfds:
        select(STDIN+1, &readfds, NULL, NULL, &tv);
        if (FD_ISSET(STDIN, &readfds))
            printf("A key was pressed!\n");
        else
            printf("Timed out.\n");
        return 0;
}
```

- Схема на сървер, обслужващ много клиенти, от които той само чете

```
множество дескриптори setd, setw;
Създаване на socket на услугата – sockserv;
setw = {sockserv };
while (1) {
   setd = setw;
   select (..,setd,NULL,NULL,NULL);
   for i in setd {
       if (i == sockserv) {
              sockconn = accept(..);
              setw = setw + sockconn; }
       else обслужва се i-тия клиент (ako i-тия
       клиент изпраща низ с нулева дължина, неговият
       дескриптор се премахва от setw);
   }
}
```