Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu
Zavod za elektroničke sustave i obradbu informacija
Laboratorij za sustave i signale

Programska podrška mjernih i procesnih sustava

vježba br. 6: **MREŽNO PROGRAMIRANJE**

NAPOMENE

- Prisjetiti se nekih cjelina iz programiranja u C-u: strukture, pokazivači, pokazivači na strukture.

ZADACI ZA VJEŽBU

1. SOCKET

Linux u sebi sadrži dobru podršku za rad u mrežnom okružju. Korisnik – programer ne mora nužno poznavati sve pojedinosti specifikacije npr. TCP/IP protokola da bi mogao napisati program koji pomoću TCP/IP protokola komunicira s nekim drugim programom na nekom drugom računalu na mreži. Dovoljno je da poznaje sistemske pozive koji apstrahiraju rad s mrežom. Sam će se *kernel* operacijskog sustava pobrinuti da se poslani podaci "zapakiraju" na odgovarajući način sa svim potrebnim (Ethernet, TCP, IP...) zaglavljima i kontrolnim poljima. Apstrakcija nastala na Sveučilištu u Berkeley-u (BSD Unix - Berkeley System Distribution) koja omogućava rad sa mrežom naziva se **socket**.

Socket je mehanizam za kreiranje virtualne veze između procesa. Omogućavaju interprocesnu komunikaciju (IPC) i kada su procesi na različitim računalima koji su spojeni preko mreže.

Socketi povezuju standardni I/O sistem sa mogućnostima mrežne komunikacije. Sistemski poziv `socket()` kreira *socket* i vraća *file descriptor*. Općenito, *file descriptor* je cijeli broj (int) koji unutar procesa identificira otvorenu datoteku, a dobiven je kao rezultat njezinog otvaranja. U Linux-u je većina toga predstavljeno datotekama. Naime, pisanje ili čitanje uvijek se vrši pomoću *file descriptora*. Operacijski sustav dopušta da *file descriptor* predstavlja mnoge druge stvari pored obične datoteke: cjevovod (*pipe*), imenovani cjevovod (FIFO), terminal ili u našem slučaju mrežnu vezu – *socket*.
Dakle, kada se želi obaviti neka operacija sa određenim *socketom*, potrebno je pozvati se na njegov *file descriptor*. Evo kako izgleda poziv funkcije `socket()`:

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

Prvi ulazni parametar `domain` označava porodicu adrese (*address family*). Evo nekoliko mogućih porodica adresa:

AF_INET (DARPA Internet adrese, IP based)
AF_UNIX (imena na lokalnon računalu, interni protokol)
AF_CCITT (CCITT X.25 adrese)

Za potrebe ove vježbe koristit će se isključivo AF_INET. Tip *socketa* (`type` u pozivu funkcije) može biti SOCK_STREAM ili SOCK_DGRAM.

SOCK_STREAM Sequenced, reliable, two-way-connection-based byte streams.

SOCK_DGRAM Datagrams (connectionless, unreliable messages of a fixed, typically small, maximum length).

Stream socketi su, kako se vidi, vrlo pouzdani i osiguravaju stalnu vezu za prijenos podataka. Oni koriste TCP (*Transmission Control Protocol*) protokol. Dobar primjer za TCP/IP protokol je telnet aplikacija. Za razliku od *stream socketa*, *datagram socketi* se ne oslanjaju na trajno uspostavljenu vezu. Paketi (tipično vrlo maleni) se šalju bez zahtjeva o potvrdi njihovog primitka. Datagram *socketi* koriste UDP (*User Datagram Protocol*). Ukoliko se želi osigurati pouzdanost prijenosa podataka, potrebno je da o tome vode brigu viši slojevi protokola (aplikacijski sloj).
Varijabla `protocol` određuje o kojem se protokolu radi. Obično za jedan tip *socketa* koji koristi određenu porodicu adresa postoji samo jedan mogući protokol. No, može se dogoditi da postoje više odgovarajućih protokola. U tom je slučaju potrebno navesti koji se protokol koristi. Ukoliko se varijabli protokol pridijeli 0, sistem će sam odlučiti koji će protokol koristiti. Tako se najčešće i radi.
Ukoliko dođe do greške funkcija `socket()` vraća vrijednost –1 i informaciju o grešci pohranjuje u globalnu varijablu `errno`. Poruku o grešci (sadržaj varijable `errno`) može se dobiti korištenjem funkcije `perror()`.

Da bi se poziv `socket()` mogao koristiti potrebno je u zaglavlju programa uključiti *header* datoteke:

```
#include <sys/types.h>
#include <sys/socket.h>
```

U datoteci `/usr/include/sys/socket.h` nalazi se deklaracija funkcije `socket()`, ali i mnogih srodnih sistemskih poziva.

2. FORMAT ZAPISA PODATAKA I STRUKTURE

Kada se podaci prenose preko mreže dogovor je da se prenose takvim redoslijedom da prvi ide najznačajniji byte (*most significant byte*). To je tzv. *Network Byte Order* (NBO). Na računalu se, naprotiv, podaci mogu (ali i ne moraju) spremati u suprotnom formatu, tako da prvi bude najmanje značajni byte (*Host Byte Order*). Bez obzira na format zapisa podataka na lokalnom računalu, važno je da podaci koji se šalju na mrežu budu u NBO formatu. Većina UNIX sustava imaju već gotove funkcije koje omogućavaju pretvorbe iz jednog zapisa u drugi:

```
htons()--"Host to Network Short"
htonl()--"Host to Network Long"
ntohs()--"Network to Host Short"
ntohl()--"Network to Host Long"
```

Oznaka `s` u nazivima pojedinih funkcija stoji za *short* (dva bytea), `l` za *long* (četiri bytea), `h` za *host*, a `n` za *network*.

U *header* datotekama sustava (npr. `socket.h`) već su definirane neke strukture podataka. Operacije sa *socketima* većinom se pozivaju na te strukture. Struktura `sockaddr` sadrži informacije o adresi za različite tipove *socketa*:

```
struct sockaddr {
     unsigned short sa_family;    // address family, AF_xxx
     char           sa_data[14];  // 14 bytes of protocol address
};
```

U našem slučaju, `sa_family` imati će vrijednost AF_INET. `sa_data` sadrži u sebi odredišnu IP adresu i broj porta za dotični *socket*. Da bi se olakšalo baratanje sa adresama socketa stvorena je još jedna struktura pod nazivom `struct sockaddr_in`, gdje `in` dolazi od Internet.

```
struct sockaddr_in {
     short int          sin_family;  // Address family
     unsigned short int sin_port;    // Port number
     struct in_addr     sin_addr;    // Internet address
     unsigned char      sin_zero[8]; // Sve nule, da bi imali
                                     // istu veličinu kao sockaddr
};
```

Uočite da je `sin_addr` deklarirana kao `struct in_addr`. To je struktura koja je takvom ostala zbog povijesnih razloga:

```
/* Internet address (a structure for historical reasons) */
struct in_addr {
     unsigned long s_addr;
};
```

Varijablom `sockaddr_in.sin_addr.s_addr` se pristupa četverobajtnoj Internet adresi *socketa*. Pošto se varijable `sockaddr_in.sin_port` i `sockaddr_in.sin_addr` šalju preko mreže, mora ih se obavezno pretvoriti u NBO format.

Gornje funkcije i strukture definirane su u `/usr/include/netinet/in.h` *header* datoteci. Zbog toga je potrebno u programima koji barataju sa Internet *socketima* navesti:

```
#include <netinet/in.h>
```

3. IP ADRESE I DNS

IP adrese obično se prikazuju zapisom koji je kombinacija brojki i točaka (npr. 161.53.64.3). Sam zapis IP adrese u prethodno navedenim strukturama je formata `unsigned long`, dakle običan četverobajtni broj. Da bi se IP adresa mogla po volji mijenjati iz jednog oblika zapisa u drugi postoje već gotove funkcije. Ukoliko npr. strukturi `moja_adresa_socketa` koja je tipa `sockaddr_in` želimo pridijeliti IP adresu 161.53.64.3, učinit ćemo to ovako:

```
moja_adresa_socketa.sin_addr.s_addr = inet_addr("161.53.64.3");
```

Prednost je funkcije **`inet_addr()`** što automatski vraća rezultat u NBO formatu. U slučaju greške funkcija vraća vrijednost –1 (OPREZ! Ukoliko se u programu ne provjerava greška, može se dogoditi da se poruka o grešci protumači kao IP adresa 255.255.255.255 kojoj odgovara `unsigned long` -1).
Evo deklaracije funkcije koja omogućava da se učini suprotna pretvorba:

```
char *inet_ntoa(struct in_addr in);          //network to ASCII
```

Uočite da kao argument funkcija **`inet_ntoa()`** uzima cijelu strukturu `in` tipa `in_addr`. Kao rezultat vraća pokazivač na niz znakova. Dakle, ukoliko bi htjeli ispisati na standardni izlaz IP adresu u zapisu pomoću brojki i točaka koristili bi slijedeću naredbu:

```
printf("%s",inet_ntoa(moja_adresa_socketa.sin_addr));
```

Često se neki program može pozvati sa imenom *hosta* na koji se želi spojiti (npr. spajanje na ftp poslužitelj na računalu maja: `ftp maja.zesoi.fer.hr`). Ime hosta potrebno je pretvoriti u njegovu IP adresu. To je omogućeno korištenjem funkcije **`gethostbyname()`**.

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
```

Funkcija vraća pokazivač na strukturu `hostent`:

```
struct hostent {
        char    *h_name;            //ime hosta
        char    **h_aliases;        //alternativna imena hosta
        int     h_addrtype;         //tip adrese, obično AF_INET
        int     h_length;           //duljina adrese u byteovima
        char    **h_addr_list;      //mrežne adrese hosta (može
                                    //biti više mrežnih kartica)
    };

#define h_addr h_addr_list[0]       //prva adresa u h_addr_list
```

Evo primjera programa kojeg se poziva sa imenom hosta. Program vraća traženu IP adresu.

adresa.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
       struct hostent *h;  //definiran pokazivac na strukturu h tipa hostent

       if (argc != 2)      //ukoliko broj ulaznih parametara ukljucujuci i ime
                           //programa nije jednak 2
                           //javi poruku o gresci sa objasnjenjem poziva programa
       {
            fprintf(stderr,"upotreba: adresa ime_hosta\n");
            exit(1);
       }

       if ((h=gethostbyname(argv[1])) == NULL) //pozovi funkciju s imenom
                                               //hosta iz poziva programa i provjeri
                                               //da nije doslo do greske
       {
            herror("gethostbyname");           //ispisi poruku o gresci
            exit(1);
       }

       printf("Ime hosta : %s\n", h->h_name);
       printf("IP adresa : %s\n", inet_ntoa(*((struct in_addr *)h->h_addr)));

       return 0;
}
```

Prisjetite se da se elemente strukture prikazane pomoću pokazivača referencira operatorom `->`. Pošto funkcija `inet_ntoa` kao ulaz zahtijeva strukturu tipa `in_addr`, bilo je potrebno *castirati* `h_addr` na tip strukture `in_addr`.

- Iskompajlirati program `adresa.c` i pokrenuti ga. Kakvu poruku se dobije ako se unese ime *hosta* koje ne postoji?

4. OSNOVNI POZIVI SUSTAVA ZA BARATANJE SOCKETIMA

Osnovni model komunikacije dva procesa je tipa klijent-server. Klijent se razlikuje od servera po tome što on inicira vezu dok server čeka konekciju. Jednom kada je veza uspostavljena oni ravnopravno razmjenjuju podatke.

Dolje su navedene često korištene funkcije u baratanju sa socketima. Neke funkcije se koriste samo na server strani a neke samo na klijent strani.

BIND

Asocira proces sa adresom. Koristi se obično samo u serveru. Time socket dobiva svoju adresu tako da ga klijenti mogu adresirati (pronaći).

```
int bind(int sockfd, struct sockaddr *my_addr, int addrlen);
```

Argumenti su: socket *file descriptor*, `struct sockaddr` definira IP adresu i port koji jednoznačno određuju adresu servera. Zadnji argument je veličina strukture `sockaddr`.

LISTEN

Omogućava da server sluša/očekuje konekcije na socket.

```
int listen(int sockfd, int input_queue_size);
```

Drugi argument predstavlja broj konekcija koje stanu u red čekanja.

ACCEPT

Server prihvaća novu konekciju. Kao rezultat vraća *file descriptor* novog socketa (tako da prvi i dalje može slušati), preko kojeg se odvija daljnja komunikacija.

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

Zadnja dva argumenta definiraju pokazivač na memorijski sadržaj koji se popuni sa klijentovim podacima tj. njegovom adresom.

CONNECT

Klijent spaja svoj socket sa udaljenim serverom:

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

Drugi argument pokazuje na strukturu koja definira adresu servera.

RECV

Prima (čita) poruke sa socketa.

```
int recv(int sockfd, void *buf, int len, unsigned int flags);
```

Argument `buf` je pokazivač na niz duljine `len` koji se prima. Flags je obično 0.

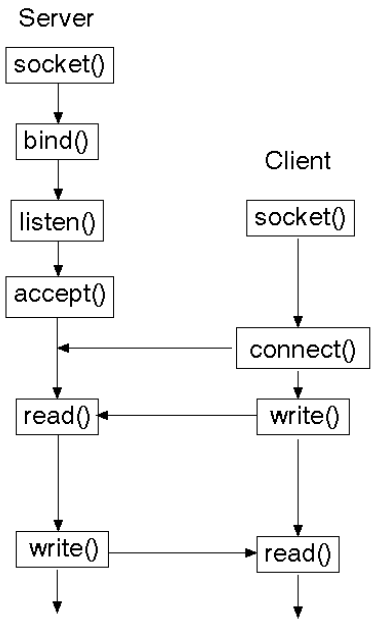
SEND

Šalje (piše) podatke na drugu stranu preko socketa.

```
int send(int sockfd, const void *msg, int len, int flags);
```

Argument `msg` je pokazivač na niz duljine `len` koji se šalje. Flags je obično 0.

Upotreba ovih poziva kod klijent-server modela je dana na slici:



U dodatnim materijalima dana su detaljnija objašnjenja poziva ovih funkcija i primjeri njihovog korištenja.

5. TCP SERVER

Osnovna struktura jedne serverske aplikacije (koja koristi sve "blagodati" BSD poziva) mogla bi biti slijedeća:

1. `socket()` -- Stvori novi *socket* pozivom `socket()`.
2. `bind()` -- Poveži (*bind*) *socket* sa IP adresom i brojem porta.
3. `listen()` -- Slušaj na prethodno određenom portu u očekivanju nadolazećih zahtjeva za uspostavom veze.
4. `accept()` -- Prihvati novu vezu.

Treba naglasiti da `accept()` vraća novi *file descriptor.* Time je omogućeno da se stari *socket* brine za posluživanje novih zahtjeva za spajanjem dok se novi može baviti primanjem i slanjem podataka (`send()` i `recv()`).

Evo primjera jednog jednostavnog, ali potpuno funkcionalnog *stream* (TCP) servera.

server.c

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

#define MYPORT 3490    /* port na kojem ce se server nalaziti */

#define BACKLOG 10     /* koliko konekcija stane u red */

main()
{
```

```
    int sockfd, new_sockfd;  /* slusaj na sock_fd, nova konekcija na new_fd */
    struct sockaddr_in my_addr;    /* moja adresa  */
    struct sockaddr_in their_addr; /* adresa klijenta (onaj koji se spaja) */
    int sin_size;

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    my_addr.sin_family = AF_INET;         /* Internet socket, host byte order */
    my_addr.sin_port = htons(MYPORT);     /* port. short, network byte order */
    my_addr.sin_addr.s_addr = INADDR_ANY; /* popuni sa mojom IP  adresom*/
    bzero(&(my_addr.sin_zero), 8);        /* ostatak stukture treba popunuiti sa 0 */

    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror("listen");
        exit(1);
    }

    while(1) {  /* glavna accept() petlja */
        sin_size = sizeof(struct sockaddr_in);
        if ((new_sockfd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size))==-1)
        {
            perror("accept");
            continue;
        }
        printf("server: konekcija od %s:%d\n",inet_ntoa(their_addr.sin_addr),
                    ntohs(their_addr.sin_port));

        if (send(new_sockfd, "Hello, world!\n", 14, 0) == -1)
                perror("send");

        close(new_sockfd);  /* zatvori novi socket */
    }
}
```

- Proučiti programski kod servera
- Iskompajlirati primjer `server.c`. U drugoj virtualnoj konzoli pokrenuti telnet aplikaciju i spojiti se na server na port na kojem se nalazi – port 3490:
  `telnet localhost 3490`

6. TCP KLIJENT

Umjesto telneta možemo sami napisati klijent koji će se spajati na dotični server.

client.c

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

#define PORT 3490    /* port na kojeg se klijent spaja */

#define MAXDATASIZE 100 /* max number of bytes koji primamo */

int main(int argc, char *argv[])
```

```
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE];
    struct hostent *he;
    struct sockaddr_in their_addr;      /* connector's address information */

    if (argc != 2) {
        fprintf(stderr,"uporaba: client hostname\n");
        exit(1);
    }

    if ((he=gethostbyname(argv[1])) == NULL) {  /* get the host info */
        perror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    their_addr.sin_family = AF_INET;           /* Internet socket, host byte order */
    their_addr.sin_port = htons(PORT);         /* Port. short, network byte order */
    their_addr.sin_addr = *((struct in_addr *)he->h_addr); /* adresa servera */
    bzero(&(their_addr.sin_zero), 8);          /* ostatak strukture ide na 0 */

    /* spajanje na server */
    if (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
    {
        perror("connect");
        exit(1);
    }

     /* primanje podataka sa servera */
    if ((numbytes=recv(sockfd, buf, MAXDATASIZE, 0)) == -1) {
        perror("recv");
        exit(1);
    }

    buf[numbytes] = '\0'; // primljeni string zavrsi sa '\0'

    printf("Received: %s",buf);

    close(sockfd);

    return 0;
}
```

- Proučiti programski kod klijenta
- Iskompajlirati primjer `client.c`. Pokrenuti klijenta narebom `client localhost`. Koju se poruku dobiva dok server nije aktivan? Pokrenuti sada i server u drugoj virtualnoj konzoli. Probati sada ostvariti komunikaciju između klijenta i servera.

7. ZADACI

A)
Napravite svoj *web server* te *web klijent*. Server neka se nalazi na portu 8000. Dodatno neka server svaki novi upit rješava u zasebnom procesu (dijete) tako da se roditelj može vratiti i prihvaćati nove konekcije.

Dakle na upit klijenta :
`GET /ime_filea HTTP/1.1`

treba vratiti sadržaj te datoteke. Postoji web stranica `one.html` koju možete zatražiti.

Probajte koristiti i web browser kao klijent. URL:
`http://localhost:8000/one.html`

Kao pomoć možete koristiti slijedeći kod za dohvat traženog filea:

read_html.c

```
char ret_buf [32768];  // globalna varijaba koja ce drzat sadrzaj filea

char *error_return = "<HTML>\n<BODY>File not found \n</BODY>\n</HTML>";

char *read_HTML(char *buf,int num_buf)
{
    int i;
    char *cp,*cp2;
    FILE *f;
    // izvuci ime filea iz input stringa
    cp =buf+5;
    cp2 =strstr(cp," HTTP");
    if (cp2 !=NULL)
       *cp2 = '\0';
    printf(" file:|%s|\n",cp);

    //dohvat filea :
    f =fopen(cp,"r");
    if (f ==NULL)   // file ne postoji
        return error_return;
    i =fread(ret_buf,1,32768,f);
    printf("%d bytes read from file %s \n",i,cp);
    if (i==0) {     // problem sa citanjem
       fclose(f);
       return error_return;
    }
    // stavi null character na kraj
    ret_buf[i] == '\0';
    fclose(f);
    return ret_buf;
}
```

B)
Napravite server koji će vraćat trenutno vrijeme ili datum ovisno što ga se pita.
Na upit od klijenta GETT vraća vrijeme a na upit GETD vraća datum. Veza je uspostavljena (mogu se ponavljati naredbe) sve dok klijent ne utipka naredbu QUIT.

Kao pomoć za dobivanje vremena i datuma neka vam posluži `gettime.c` kod programa.
Prvo kao klijent možete koristiti `telnet` program a onda izgradite svojeg klijenta.

gettime.c

```
#include <stdio.h>
#include <time.h>

int main() {

    time_t rawtime;
    struct tm * timeinfo;
    char text[100];

    time ( &rawtime );
    timeinfo = localtime ( &rawtime );

    sprintf(text,"Current date: %d.%d.%d.\n", timeinfo->tm_mday,
       timeinfo->tm_mon+1, timeinfo->tm_year+1900);
    printf("%s",text);
    sprintf(text,"Current time: %d:%d:%d\n", timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec);
    printf("%s",text);
    return 0;
}
```