Responsible lecturer: Illés, Zoltán associate professor
Media & Educational Informatics Department

Sockets

Goal: We learned about different IPC mechanisms like pipes, message queues, shared memories but they are useful only to deliver data between processes working on the same computer. Between different computers you can use sockets for communication purposes (though they may be used in local communications as well). In this lesson we will learn about the possibilities of sockets.

We learn about*: socketpair – creates a pair of socket (only for local usage)(include sys/types.h, sys/socket.h); read, write – reads, writes to / from a socket too; socket – create an endpoint for communication (include sys/types.h, sys/socket.h); bind – bind a name to the socket (include sys/types, sys/socket); recvfrom, sendto – receive or send data through sockets; connect – initiate a connection on a socket (include sys/types.h, sys/socket.h); listen – listen for connections (include sys/types.h, sys/socket.h); accept – accept a connection on a socket (include sys/types);*

Tasks

1. Write a C program which has got a child process. The parent process should send a „Hello" to the child using a socket. As it is a small amount of data we use datagrams, because it is quicker. *(In this case you can use the very simple local ipc using AF_UNIX s domain. If you use sockpair function you get something very similar to pipes. If you have a small amount of data you can use SOCK_DGRAM type otherwise SOCK_STREAM. After creating the socket descriptors you may use simple read and write functions.)*

```
int socketpair(int domain, int type, int protocol, int sv[2])
//domain – AF_UNIX; type – SOCK_DGRAM or SOCK_STREAM; type – 0 means default;
//sv – pair of socket descriptors
```

2. Modify the program and send back an answer too! *(Whilest you have a socket pair you do not have to think of syncronization! Just only use one of them for sending the message from parent to child and the other to send back the answer!)*

3. Modify the above written program but use connection based type! (*SOCK_STREAM*)

4. Write a server program which sends a random number to the calling client and then stops! Use datagrams in this task too! *(First you have to create a socket, bind the socket then you can receive and send data. You have to start the program as administrator. You can try the server without writing a client by using command line command from another telnet window :* ***netcat -vv localhost 8888 -u)***

```
int socket(int domain, int type, int protocol)
 //parameters the same as socketpair, return type socket descriptor

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
 //sockfd – descriptor;
//struct sockaddr {sa_family_t sa_family;…]; sa_family AF_UNIX; AF_INET;…}

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                         struct sockaddr *src_addr, socklen_t *addrlen)
//src_addr – means the client address to be able to send back the data

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                     const struct sockaddr *dest_addr, socklen_t addrlen)
//parameters the same as before
```

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government

5. Write the client program which sends an „?" to the server and gets back the random number! *(In the client program you have to create the socket, use localhost and the same port you used at the server! After it you can receive the answer!)*

6. Write a web client program! You have to give the server name and the port with arguments! (*Now, we have to use the connected mode: SOCK_STREAM. So you have to create a socket, connect itt o the server. If it is ready you may send the message to the server and receive the answer (the header and the content of the HTML file… There is a server you may try with an index.html file: 157.181.161.135)*

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
//the same as in bind function
```

7. Write the web server program as well! (*You have to create a socket, bind it as we did before and then listen to the clients and accepts them.)*

```
int listen(int sockfd, int backlog)
//backlog – how many request can be

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
//parameters are the same as bind, and connect
```

Financed from the financial support ELTE won from the Higher Education Restructuring Fund of the Hungarian Government