Høyskolen Kristiania

Denne forelesningsøkten vil bli tatt opp og lagt ut i emnet i etterkant.

<u>Hvis</u> du ikke vil være med på opptaket:

	La være å delta med webkameraet ditt.
	La være å delta med mikrofonen din.
	Still spørsmål i Chat i stedet for som lyd. Hvis du ønsker kan spørsmålet også sendes privat til foreleser.

Høyskolen
Kristiania

Høyskolen Kristiania

PG3401 Programmering i C for Linux

Bengt Østby

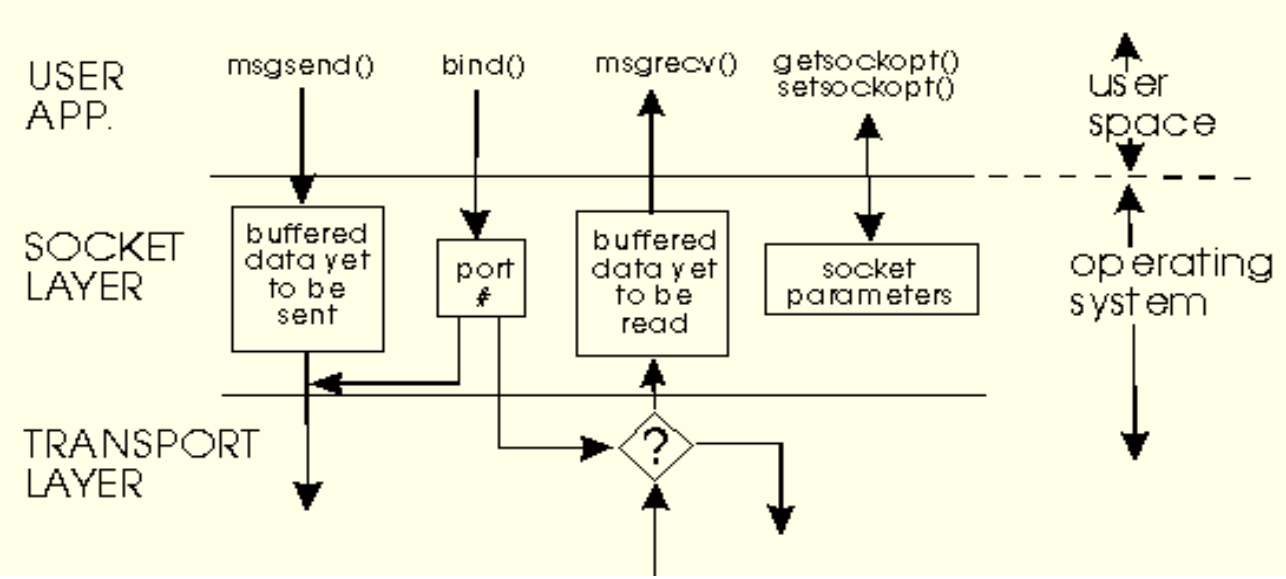
Høyskolen Kristiania

PG3400 Programmering i C for Linux

Lecture #9 : Network programming

Sockets (API)

- Definerer forbindelsen (”grensesnittet”) mellom applikasjons- og transport-laget
- Socket = ”Internett API”
  - To prosesser kommuniserer med hverandre over Internett ved å sende data inn i socket og lese data ut fra socket
- Adresse til ønsket kommunikasjons-partner dannes av IP-adresse (**vertsmaskin**-«id») **og** port-nummer (**prosess**-”id”)



Linux Sockets

(also called POSIX sockets, Berkley Sockets or BSD Sockets)

Include <sys/sockets.h>

Build with –lsocket (add to CFLAGS in makefile)

socket()

bind()

listen()

connect()

write()

accept()

Linux Sockets

Moving further down the rabbit hole ☺

Sockets can be a file in the filesystem used for inter-process communication, this is called a “unix domain” socket type, we will not cover this in this course and will only work with network sockets.

Same as pthreads – sockets are part of POSIX

As threads; sockets are easy to teach, but difficult to learn!

Creating a server

1. Include <sys/socket.h>

2. Link using –lsockets flag (not always required)

3. Create a socket by calling socket()

4. Bind to address and port by calling bind() + listen()

Creating a server

socket()

The function creates a socket. First argument can be either AF_UNIX or AF_INET, for a network socket we use INET. Second argument can be SOCK_STREAM or SOCK_DGRAM depending on if we want to read it “as a stream” or “as datagram”. Third argument is protocol, but we set it to 0 because TCP is implied with STREAM and UDP is implied with DGRAM sockets.

```
int sockFd;
sockFd = socket(AF_INET, SOCK_STREAM, 0);
if (sockFd < 0)
  printf("ERROR opening socket");
```

Creating a server

bind()

The function binds to a socket. It takes a IPv4 or IPv6 socket struct as argument, and the size of that structure shows if we want to use IPv4 or IPv6 (as well as the sin_family member of the struct).

```
struct sockaddr_in saAddr = {0}; int iPort = atoi(“80”);
saAddr.sin_family = AF_INET;
saAddr.sin_port = htons(iPort);
saAddr.sin_addr.s_addr = INADDR_ANY;
if (bind(sockFd, (struct sockaddr *) &saAddr,
sizeof(saAddr))<0)
  printf("ERROR on binding");
```

Creating a server

Høyskolen Kristiania

listen()

The function listens on a socket. Argument is number of connections that can be waiting, recommended value is 5.

```
listen(sockFd, 5);
```

Creating a client

1. Include <sys/socket.h>

2. Link using –lsockets flag (not always required)

3. Create a socket by calling socket()

4. Connect to address by calling connect()

Creating a client

socket()

The function creates a socket. First argument can be either AF_UNIX or AF_INET, for a network socket we use INET. Second argument can be SOCK_STREAM or SOCK_DGRAM depending on if we want to read it “as a stream” or “as datagram”. Third argument is protocol, but we set it to 0 because TCP is implied with STREAM and UDP is implied with DGRAM sockets.

```
int sockFd;
sockFd = socket(AF_INET, SOCK_STREAM, 0);
if (sockFd < 0)
  printf("ERROR opening socket");
```

Creating a client

connect()

The function connects to a socket. It takes a IPv4 or IPv6 socket struct as argument, and the size of that structure shows if we want to use IPv4 or IPv6 (as well as the sin_family member of the struct).

```
struct sockaddr_in saAddr = {0};
int iPort = atoi(“80”);
saAddr.sin_family = AF_INET;
saAddr.sin_port = htons(iPort);
saAddr.sin_addr.s_addr = htols(0x7F000001); // 127.0.0.1
if (connect(sockFd,&saAddr,sizeof(saAddr)) < 0)
  printf("ERROR connecting");
```

Send and receive data

Høyskolen Kristiania

5. Server calls accept() to “accept” the clients connect call

6. Call read() or write() depending

Send and receive data

accept()

The function accepts a connection on a socket. Receives a new sockaddr and FD.

```
  int i = sizeof(cli_addr), sockNewFd = 0;
  struct sockaddr_in saConClient = {0};
  sockNewFd = accept(sockFd, (struct sockaddr *) &saConClient,
&i);
  if (sockNewFd < 0)
    printf("ERROR on accept");
```

Note that the function returns a NEW socket file descriptor, this must be used when the server reads or writes from the socket. (A real server will typically send this socket FD to a new thread, and then the main thread will call accept function again to wait for more clients to connect.)

Send and receive data

write()

The function writes to a socket. Very simple function that takes a memory buffer and the number of characters to write.

```
n = write(sockFd /*sockNewFd*/, szBuffer, strlen(szBuffer));
if (n < 0)
  printf("ERROR writing to socket");
```

https://www.man7.org/linux/man-pages/man2/write.2.html

Send and receive data

send()

The function writes to a socket. Very simple function that takes a memory buffer and the number of characters to write. This is the socket specific function that is the preferred method for sending data, last parameter is flags that can be set to 0.

```
n = send(sockFd, szBuffer, strlen(szBuffer), 0);
if (n < 0)
  printf("ERROR writing to socket");
```

https://man7.org/linux/man-pages/man2/send.2.html

Send and receive data

read()

The function reads from a socket. Very simple function that reads into a memory buffer, to a maximum number of bytes.

```
bzero(szBuffer, 256);
n = read(sockFd /*sockNewFd*/, szBuffer, 256-1);
if (n < 0)
  printf("ERROR reading from socket");
```

It is easy if you know how many bytes you expect, more tricky if you don't. If you read 100 bytes and the sender writes 98 this CAN block (and documentation is a bit tricky as well since the read/write calls are generic file operations.

https://www.man7.org/linux/man-pages/man2/read.2.html

Send and receive data

recv()

The function reads from a socket. Very simple function that reads into a memory buffer, to a maximum number of bytes. Flags can be used to control if the function should block or not, MSG_PEEK is also interesting to use sometimes.

```
bzero(szBuffer, 256);
n = recv(sockFd, szBuffer, 256-1, MSG_DONTWAIT);
if (n < 0)
  printf("ERROR reading from socket");
```

MSG_DONTWAIT and other flags means you have more control than with read(), but you can still get into blocking problems. Peaking can tell you how much is available.

https://man7.org/linux/man-pages/man2/recv.2.html

Send and receive data

Høyskolen Kristiania

You need to make sure you read the correct amount of bytes, a way to handle that in a structured way is to have a header in all messages.

Note; I almost always send a struct of the common MTU size (1500 bytes – 120 bytes in TCP+IP headers) to hold the data as well, then I know how big it is…

```
typedef struct _DATAHEADER {int Type; int Bytes; } DH;
DH stHeader = {0};
bzero(szBuffer, 256);
n = recv(sockFd, &stHeader, sizeof(DH), MSG_DONTWAIT);
if (n < 0){
  printf("ERROR reading from socket");
} else {
  n = recv(sockFd, szBuffer, stHeader.Bytes, MSG_DONTWAIT);
  if (n < 0)
    printf("ERROR reading from socket");
}
```

Only scratches the surface :-)

Høyskolen Kristiania

This is the minimum you need to know about network programming.

For further reading read the MAN pages (following the links on the previous slides), there are many ways of sending and receiving data, many different flavors. Some functions you choose depending on problem to solve, some you choose because you prefer that style/flavor.

https://www.binarytides.com/socket-programming-c-linux-tutorial/

https://www.thegeekstuff.com/2011/12/c-socket-programming/

For “real programming” also check out gethostbyname() to perform a DNS lookup. If you need encryption use OpenSSL – supports creating a TLS layer under any protocol you have implemented (not only http), but wait until after exam ☺