Programovanie v operačných systémoch

05 - Network

Jozef Šiška

Department of Applied Informatics
Comenius University in Bratislava

2025/2026

Networking Recap

- HW
- Ethernet
- IP
  - TCP
  - UDP
  - DNS
- Local sockets (Unix sockets)

Socket

Create a socket (not yet connected to anything)

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

Domain

AF_UNIX, AF_LOCAL	local sockets (man 7 unix)
AF_INET	IPv4 (man 7 ip)
AF_INET6	IPv6 (man 7 ipv6)
AF_NETLINK	kernel userspace interface (man 7 netlink)
AF_IPX, AF_X25,...	

Type

SOCK_STREAM	reliable byte stream (i.e. TCP)
SOCK_DGRAM	connectionless, unreliable messages (UDP)
SOCK_SEQPACKET, SOCK_RAW, ...	

`man 2 socket`

`man 7 {unix,ip,ipv6}`

Connect and communicate

Server

- Bind a socket to an address

```
int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
```

- Listen on the socket for incomming connections

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

- Accept a connection

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

Client

```
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

```
man 2 {bind,listen,accept,connect}
```

Reading, writing, closing

Read	Write	
read	write	plain read/write
recv	send	specify additional flags
recvfrom	sendto	get / specify peer address (i.e. UDP packets)
recvmsg	sendmsg	readv/writev style, additional data
	shutdown	close (one direction of) a connection
	close	close (dispose of) the socket

Addresses

A general "some address" type (man 2 bind):

```
struct sockaddr {
    sa_family_t sa_family;
    char        sa_data[14];
}
```

IPv4 address (man 7 ip, IPv6 is similar):

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

Need to cast between types:

```
struct sockaddr_in  addr;
/* set the fields, open socket */
ret = bind(sockfd, (struct sockaddr *) &addr, sizeof(addr));
```

Obtaining, printing addresses

Any address (for server)

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
```

Network vs host order (ports)

```
uint16_t portno = 1234;
addr.sin_port = htons(portno);
portno = ntohs(addr.sin_port);
```

Convert IPv4 address to sockaddr_in

```
ret = inet_aton("127.0.0.1", &addr.sin_addr);
ret = inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
ret = inet_pton(AF_INET6, "::1", &addr.sin6_addr);
```

Convert sockaddr_in to IPv4 address

```
printf("%s\n", inet_ntoa(addr.sin_addr));
char str[INET_ADDRSTRLEN];
ret = inet_ntop(AF_INET, &addr.sin_addr, str, len);
```

Resolving DNS addresses

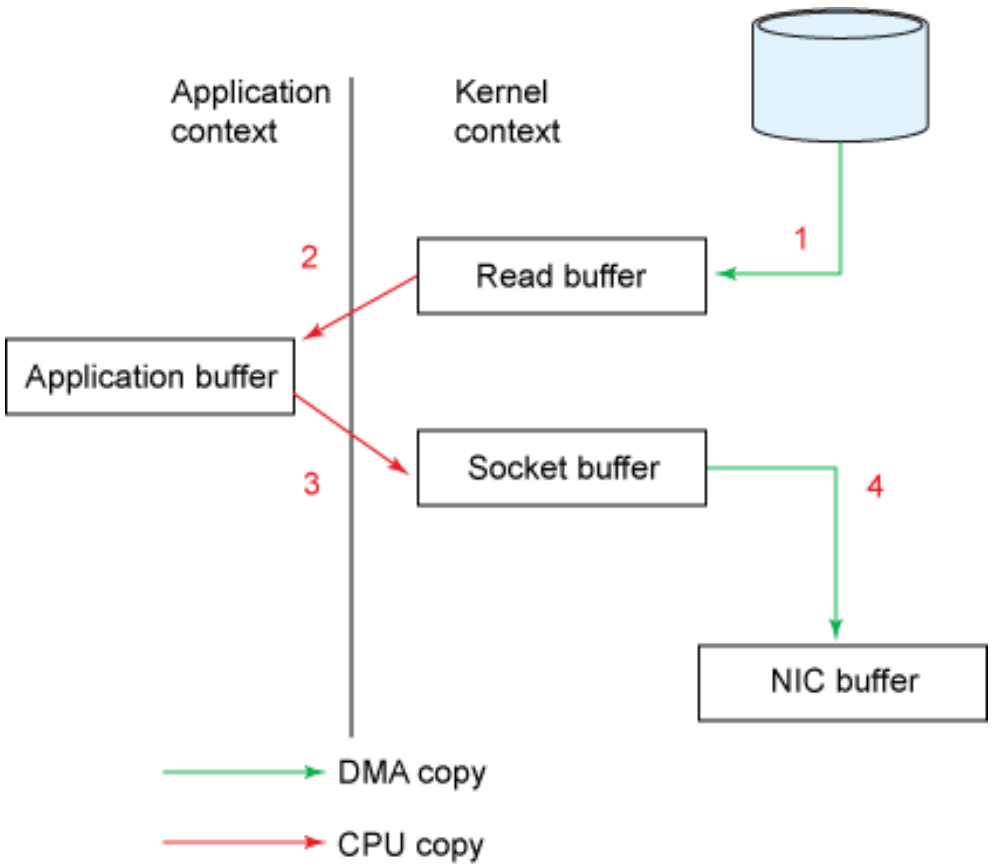
~~gethostbyname~~

```
struct sockaddr_in addr; struct hostent *server;
server = gethostbyname(str);
if (!server) { /* ... */ };
memcpy(server->h_addr, &addr.sin_addr.s_addr, server->h_length);
```

getaddrinfo

```
struct addrinfo hints, *result, *rp;
int sockFd = -1;
int ret = getaddrinfo("www.fmph.uniba.sk", "http", &hints, &result);
if (ret != 0) { /*...*/ }
for (rp = result; rp != null; rp = rp->ai_next) {
    sockFd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol));
    if (sockFd == -1)
        continue;
    if (connect(sockFd, rp->ai_addr, rp->ai_addrlen) == 0)
        break;
    close(sockFd);
}
freeaddrinfo(result);
if (rp == NULL) { /* could not connect to any of the addresses*/ }
else { /*  connected... */ }
```

Copying - problems



https://www.linuxjournal.com/article/6345

Image: https://www.ibm.com/developerworks/library/j-zerocopy/index.html

Scatter / gather operations

A final packet consist of multiple parts: ehternet frame header, ip(+tcp) header, user data. Older drivers required continuous buffers, so the parts needed to be copied over to a single buffer.
Newer driver allow scater / gater operations: multiple buffers can be specifed for a single packet (either when reading or when writing).

```
struct iovec {
    void  *iov_base;    /* Starting address */
    size_t iov_len;     /* Number of bytes to transfer */
};

ssize_t readv(int fd, const struct iovec *iov, int iovcnt);
ssize_t writev(int fd, const struct iovec *iov, int iovcnt);
```

Note: these still copy between user space and kernel space similar to `read` and `write`.

`readv(2)` `writev(2)`

Copying in the kernel

Copy a mmapped file (`in_fd` must be "mmap-able"):

```
ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

Moving data between pipes (or files and pipes):

```
ssize_t splice(int fd_in, loff_t *off_in, int fd_out,
               loff_t *off_out, size_t len, unsigned int flags);
ssize_t tee(int fd_in, int fd_out, size_t len, unsigned int flags);
ssize_t vmsplice(int fd, const struct iovec *iov,
                 unsigned long nr_segs, unsigned int flags);
```

`sendfile(2)` `splice(2)` `vmsplice(2)` `tee(2)`

https://blogs.oracle.com/linux/post/pipe-and-splice