Network Programming(III)

Lenuta Alboaie
adria@info.uaic.ro

Content

- I/O Primitives - discussions
- UDP Concurrent Server
- TCP or UDP – aspects
- Instruments
- Sending and receiving data using *out-of-band* mechanism

I/O Primitives

- Reading Data
  - read() / recv() / **readv()** / recvfrom()/ **recvmsg()**
- Sending Data
  - write() / send() / **writev()** / sendto()/ **sendmsg()**

I/O Primitives

```
#include <sys/uio.h>
ssize_t readv (int filedes, const struct iovec *iov, int iovcnt);
ssize_t writev (int filedes, const struct iovec *iov, int iovcnt);
        struct iovec
        {
                void *iov_base; /* buffer start adress */
                size_t iov_len;   /* buffer size */
        };
```

- Wider than read()/write(), it provides the ability to work with data in non-contiguous memory areas
- Both calls return, in normal execution, the transfer length in bytes

I/O Primitives

```
#include <sys/socket.h>
ssize_t recvmsg (int sockfd, struct msghdr *msg, int flags);
ssize_t sendmsg (int sockfd, struct msghdr *msg, int flags);
```

Both functions have options included in *msghrd* structure

The most general I/O functions; read/readv/recv/recvfrom calls can be replaced by *recvmsg*

Both calls return, in normal execution, the transfer length in bytes; -1 in error case

I/O Primitives

Comparison among I/O primitives:

Function	Any descriptor	Just for Socket descriptor	One read/write buffer	Scatter/ gather read/write	Optional flags	*Peer Address*
read, write	●		●			
readv, writev	●			●		
recv, send		●	●		●	
recvfrom, sendto		●	●		●	●
recvmsg, sendmsg		●		●	●	●

UDP Server| Discussions

Most UDP servers are iterative

- A UDP Server reads the client`s request, processes the request and sends the response
- What happened in the situations when multiple datagrams should be exchanged with the client?

Concurrent UDP Server

- if processing the answer takes time, the server can create (*fork*()) a child process that will resolve the client request

UDP Server| Discussions

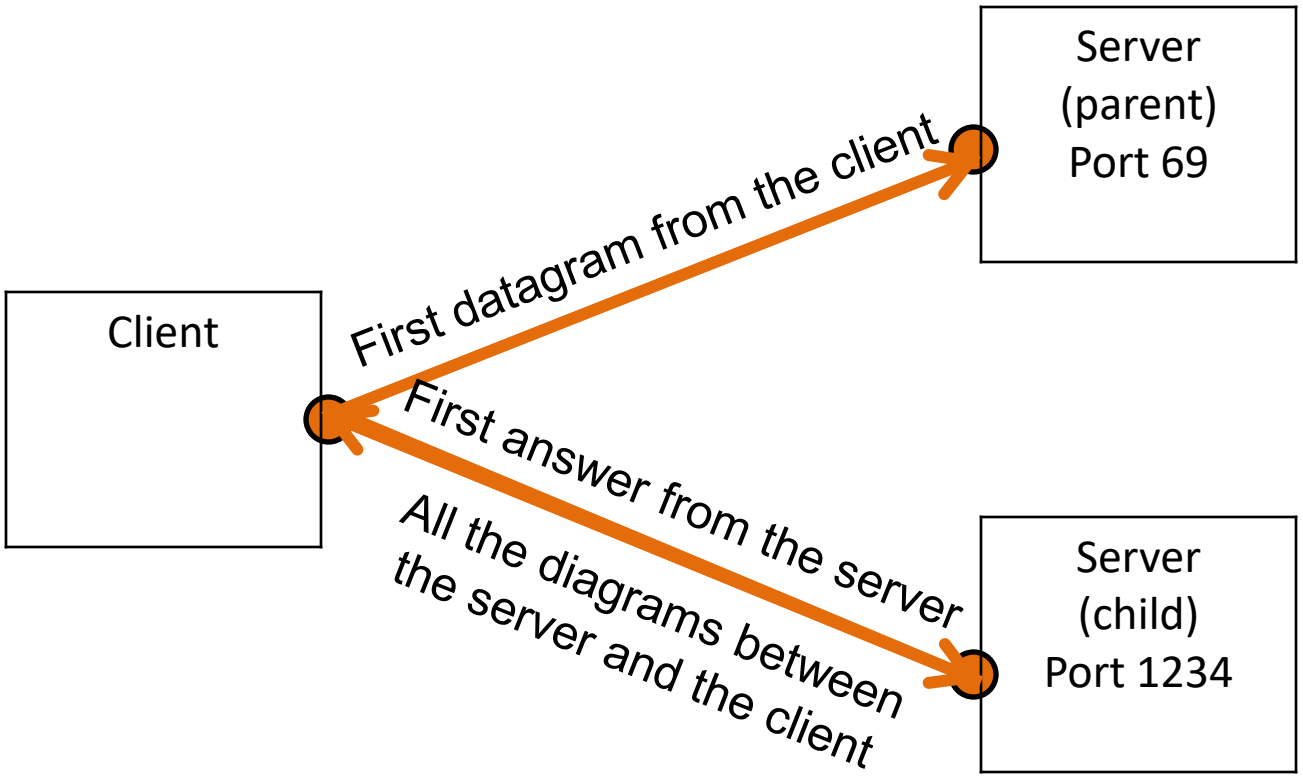
UDP Concurrent Server:

- UDP Server that exchanges multiple datagrams with clients
  - Problem: Just a port is known by the client as "*well-known*"
  - Solution: the server creates a new socket for each client, it attaches it to an "ephemeral" port and uses this socket for all answers
    - Mandatory: the client must take the port number form the server`s first response and the next requests will use that port

    Example: TFTP - Trivial File Transfer Protocol

UDP Concurrent Server

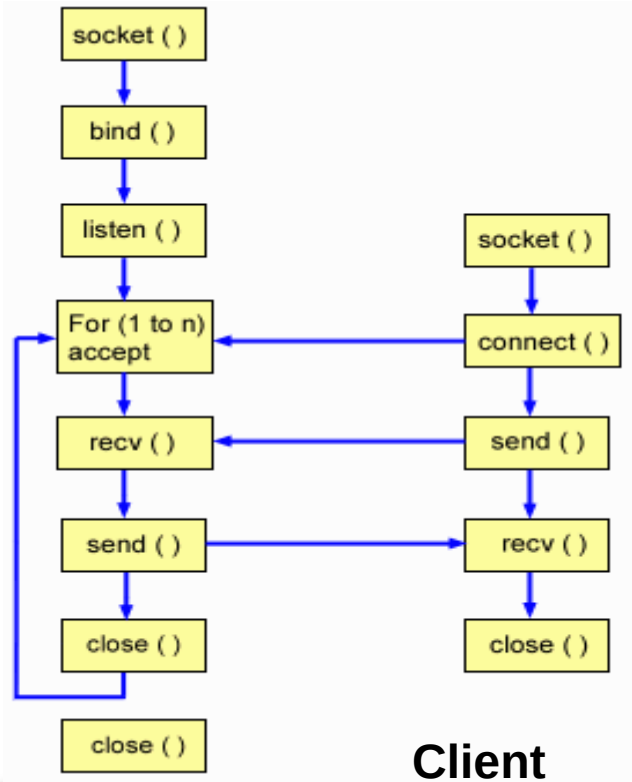
– TFTP use UDP and 69 port



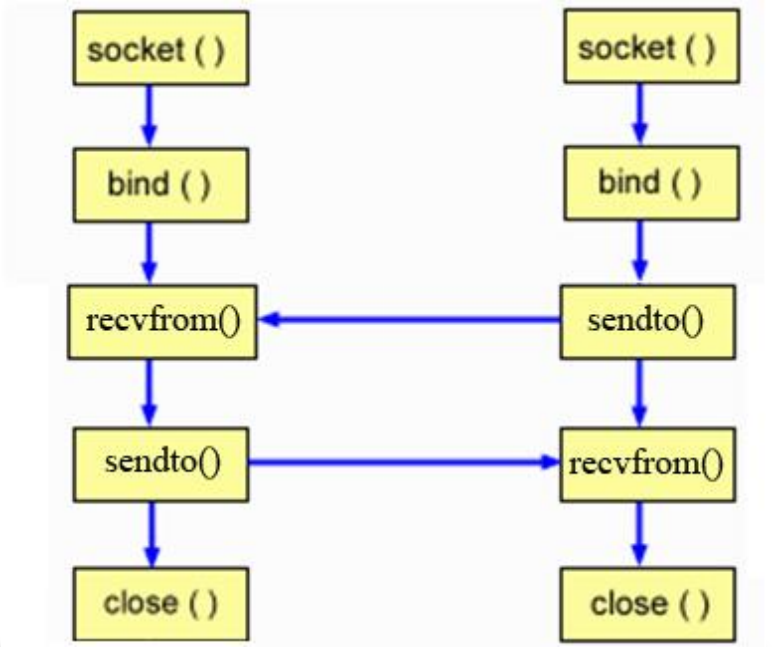
Steps: socket(), bind() to *well-known 69, recvfrom(), fork(), alt recvfrom(),…*

Steps: socket(), bind() to *1234 port,* the client`s request is processed, exchanges of the datagrams between client-server using the new port

TCP or UDP - discussions



TCP server/client Model



UDP server/client Model

TCP or UDP – Discussions

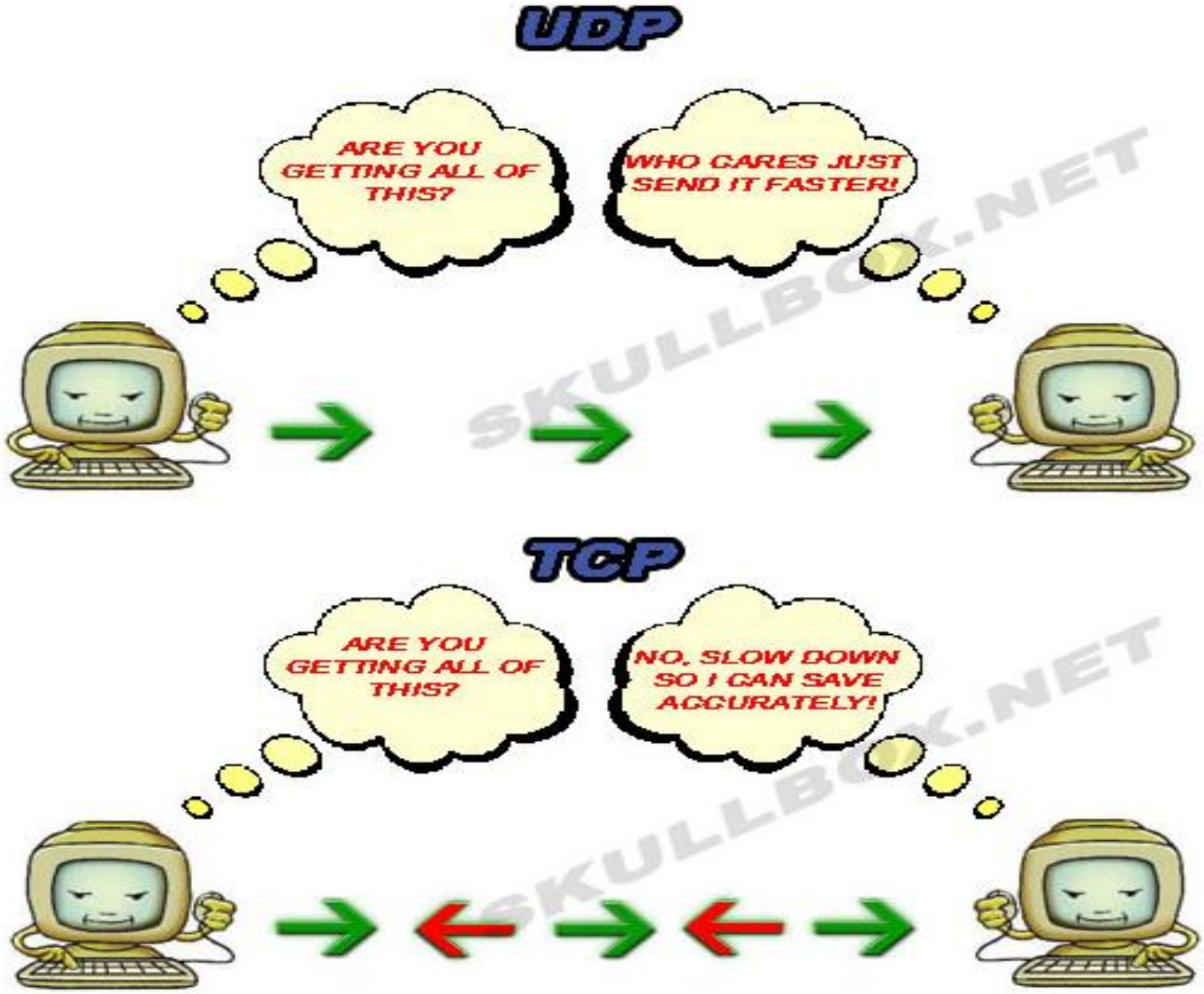
Aspects regarding UDP uses:

- UDP supports broadcasting and multicasting
- UDP does not require a mechanism to establish a connection
- The minimum time taken for a UDP transaction (request-response) is: RRT(*Round Trip* Time) + SPT (*server processing time*)

Aspects regarding TCP uses:

- TCP supports point-to-point
- TCP is connection-oriented
- Offers safety and in order data transmission;
- It provides mechanism for flow control and congestion control
- The minimum time taken for a TCP transaction is 2 *RRT + SPT

TCP or UDP – Discussions



[http://www.skullbox.net]

TCP or UDP – Discussions

UDP , TCP uses – recommendations

- UDP should be used for multicast or broadcast applications
- The error control must (eventually) be added to the server or client`s level
- UDP can be used for simple request-response operations; errors should be treated at the application level

Examples: streaming media, teleconferencing, DNS

TCP or UDP – Discussions

UDP , TCP uses – recommendations

- TCP *should be used for bulk data transfer* (e.g. file transfer)
  - Might you use UDP? → We reinvent TCP at the application level!

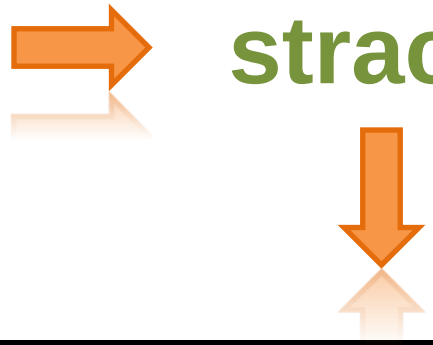
Examples: HTTP (Web), FTP (File Transfer Protocol), Telnet, SMTP

Instruments

- Multiple UNIX systems offer *"system call tracing"* facility

adria@ubu: ~/S6

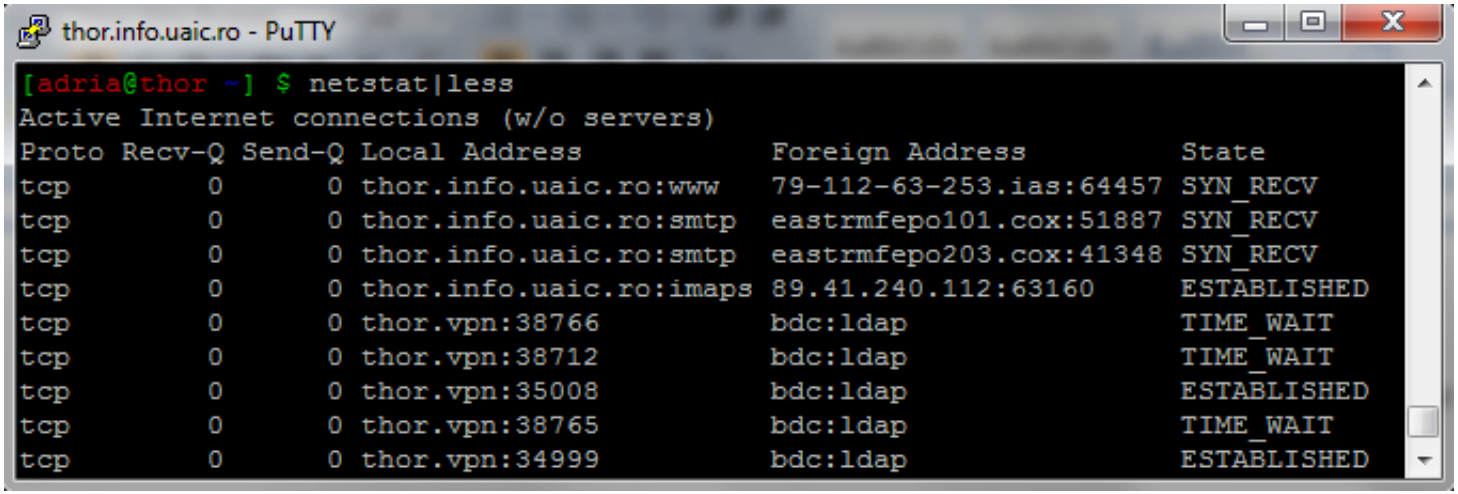
```
    I A  test.c (Modifi  Row 8    Col 28   8:15
#include <stdio.h>
#include <stdlib.h>

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

  char *sir=NULL;
  printf("program de debugs: ");
  //sir = (char *) malloc(100*sizeof(char));
  fgets(sir, 1024, stdin);
  printf(sir);
  return 1;
}
```




```
write(1, "program de debug\n"..., 17program de debug
)    = 17
fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7
fdb000
read(0, 0xb7fdb000, 1024)               = ? ERESTARTSYS (To be restarted)
--- SIGWINCH (Window changed) @ 0 (0) ---
read(0, Test in saptamina 6
"Test in saptamina 6\n"..., 1024) = 20
--- SIGSEGV (Segmentation fault) @ 0 (0) ---
+++ killed by SIGSEGV +++
```

Instruments

- Small test programs
- Instruments:
  - **tcpdump** – most versions of Unix
    - It provides information on packets from network
    - http://www.tcpdump.org/
  - **snoop** – Solaris 2.x
  - **lsof**
    - Identify what processes have an open socket to a specified IP address or port
  - **netstat**



Instruments

- Instruments:
  - **tcptrack**

Client	Server	State	Idle	A Speed
172.23.195.11:48328	67.39.222.44:22	ESTABLISHED	0s	38 KB/s
172.23.195.11:48646	196.30.80.10:80	ESTABLISHED	1s	30 KB/s
172.23.195.11:48661	64.37.246.17:80	ESTABLISHED	0s	387 B/s
172.23.195.11:48620	216.239.39.99:80	RESET	2s	0 B/s
128.230.225.95:3531	172.23.195.10:1220	ESTABLISHED	5s	0 B/s
172.23.195.11:48621	216.239.39.99:80	ESTABLISHED	7s	0 B/s
172.23.195.11:48606	64.233.167.99:80	ESTABLISHED	10s	0 B/s
172.23.195.11:48014	67.39.222.44:22	ESTABLISHED	16s	0 B/s
172.23.195.11:47988	67.39.222.44:22	ESTABLISHED	18s	0 B/s
TOTAL				69 KB/s
Connections 1-9 of 9			Unpaused	Sorted

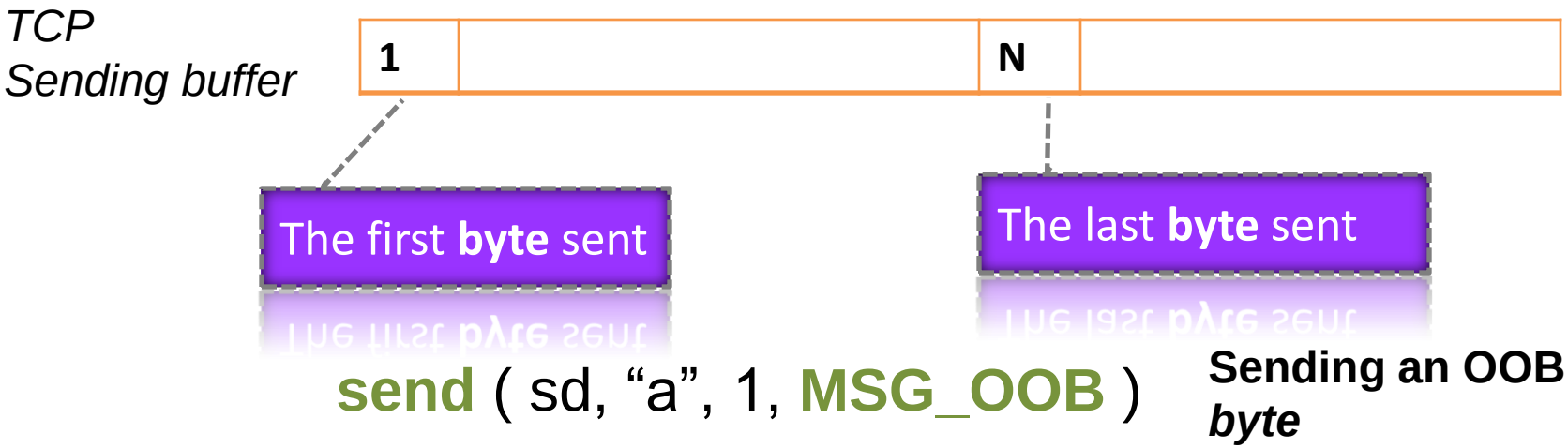
Sending and receiving data using out-of-band mechanism

- The idea: during a connection, when data are transmitted ("**in-band data**"), and at one end "something" happens it is necessary to send a priority notification ("**out-of-band data**") to the other peer
- The achieving mechanism:
  - It uses the URG bit in TCP header
  - The TCP header contains a field indicating the location of the urgent data to be sent
  - **OOB data sent**:
    - To send an urgent byte in a data stream **send()** primitive may be used:

      send (sd, buff, 1, **MSG_OOB**);

Sending and receiving data using out-of-band mechanism

Sending an OOB byte



send (sd, “a”, 1, **MSG_OOB**)

Sending an OOB *byte*

Sending and receiving data using out-of-band mechanism

Receiving OOB data :

- **SIGURG** signal is generated
- **select()** call will modify the exceptional list descriptors

  OOB data reading:

  - If the socket has no SO_OOBINLINE option, the OOB message is placed in a special buffer (*out-of-band buffer)*; reading the buffer can be done with **recv()** or **recvmsg()** setting **MSG_OOB**
  - If the socket has SO_OOBINLINE option, the OOB message is placed in the normal reception buffer;
- The process will know that it reached *the special* byte depending on *out of-band-mark* value associated with the connection

Sending and receiving data using out-of-band mechanism

sockatmark() call

- When OOB data is received, out-of-band-mark association is performed – and represents the OOB position in the data stream (sent by the transmitter)
- **sockatmark**() ensures that the receiver will determine whether *out-of-band mark* is associated or not with the connection

```
#include <sys/socket.h>
int sockatmark (int sockfd) ;
```

Returns: 1 – is *out-of-band mark*,

0 – is no *out-of-band mark*

–1 on error

Sending and receiving data using out-of-band mechanism

sockatmark() – observations

- *out-of-band mark* applies whether the process receives data in *out-of-band inline* mode(socket with SO_OOBINLINE option) or in an out-of-band mode (MSG_OOB flag)
- It is implemented using ioctl() and SIOCATMARK

```
int value;
error = ioctl(tcp_socket, ioctl_type, &value);
```

Sending and receiving data using out-of-band mechanism

sockatmark() – discussions & example

- If you have received *OOB inline* data, sockatmark() returns true if the next readable byte was sent with MSG_OOB flag
- If the *socket* has no SO_OOBINLINE option, sokatmark() return *true* if the next readable byte is the first byte sent after OOB data
- The read operation stops according to *out of-band-mark*
  - Example:

    If there are 100 bytes in buffer, but just 5 bytes up to *out of-band-mark,* even if the process reads 100 bytes, it will initially receive only the first 5 bytes

Sending and receiving data using out-of-band mechanism

Possible errors:

- OOB data is expected to be read, but they have not yet been sent – it returns EINVAL
- The process has been notified that it will receive OOB data(via select() or SIGURG), he tries to read but they haven`t reached yet – it returns EWOULDBLOCK
- It tries to read the same OOB byte multiple times – it returns EINVAL
- If the SO_OOBINLINE is set, but the reading is performed using MSG_OOB flag – it returns EINVAL

Sending and receiving data using out-of-band mechanism

- Uses:
  - Situations when one endpoint signals an exceptional circumstance to the peer endpoint (even when flow control has stopped the sender)
  - To detect early some communication errors between client and the server (*heart-beat*)

Sending and receiving data using out-of-band mechanism

OOB implementation using **SIGURG:**

```
if (listen (sd, 5) == -1)    { … }
…
while (1)
   {…    client = accept (sd, (struct sockaddr *) &from, &len);
         signal(SIGURG, urgHandler);
        fcntl(client, F_SETOWN, getpid()); /*setarea proprietarului socketului conectat */
     for(; ;)    {
          if((n=read(client,msg,sizeof(msg)-1))==0)
          {                  printf("Am primit EOF\n");
                             break;            }
          else
          {
                             msg[n]=0;
                             printf("Am citit %d octeti: %s\n",n, msg);      }
     } … } //while
void urgHandler(int signnr)
{
     int n;                 char buff[100];
     printf("SIGURG e primit\n");
     n=recv(client,buff, sizeof(buff)-1, MSG_OOB);
       buff[n]='\0';
      printf("Am citit %d octet OOB %s\n",n, buff);
}
```

Sending and receiving data using out-of-band mechanism

OOB implementation using **sokatmark**()

DEMO

- **The transmitter** sends 4 bytes of normal data, then an OOB byte and then another normal byte
- **The receiver** doesn't use SIGURG or select(); he calls **sockatmark**() primitive;

Summary

- I/O Primitives - discussions
- UDP Concurrent Server
- TCP or UDP – aspects
- Instruments
- Sending and receiving data using *out-of-band* mechanism

Bibliography

- UNIX Network Programming: The sockets networking API, W. Richard Stevens, Bill Fenner, Andrew M. Rudoff
- The Illustrated Network: How TCP/IP Works in a Modern Network (The Morgan Kaufmann Series in Networking), Walter Goralski

Questions?