Outline

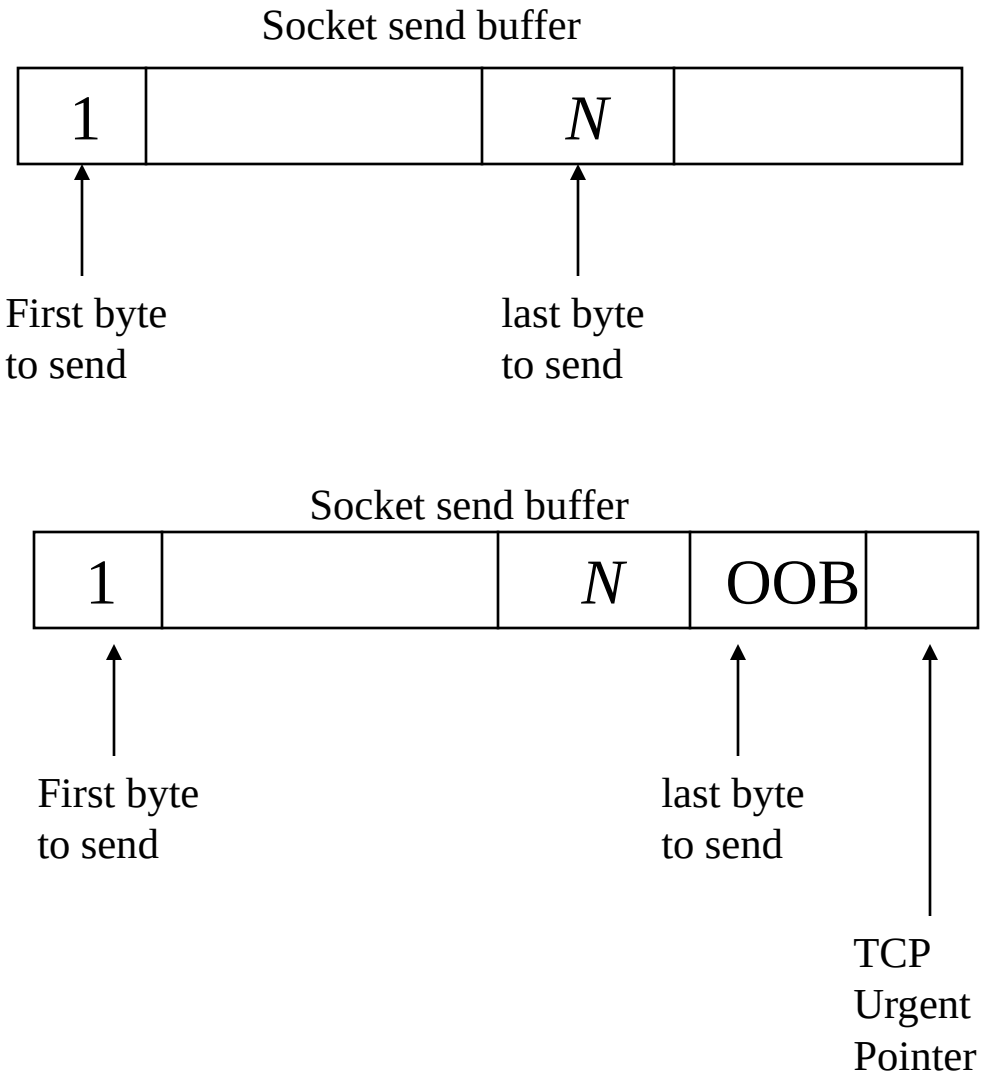
- Out-of-Band Data
  - Introduction
  - TCP Out-of-Band Data
  - **sockatmark** Function
  - Examples

Introduction

- Out-of-band data
  - Expedited data
  - Notification should be sent before any normal (*in-band*) data that is already queued to be sent
  - Higher priority than normal data
  - Out-of-band data mapped onto existing connection (instead of using two connections)
- UDP has no implementation of out-of-band data
- TCP has its own flavor of out-of-band data

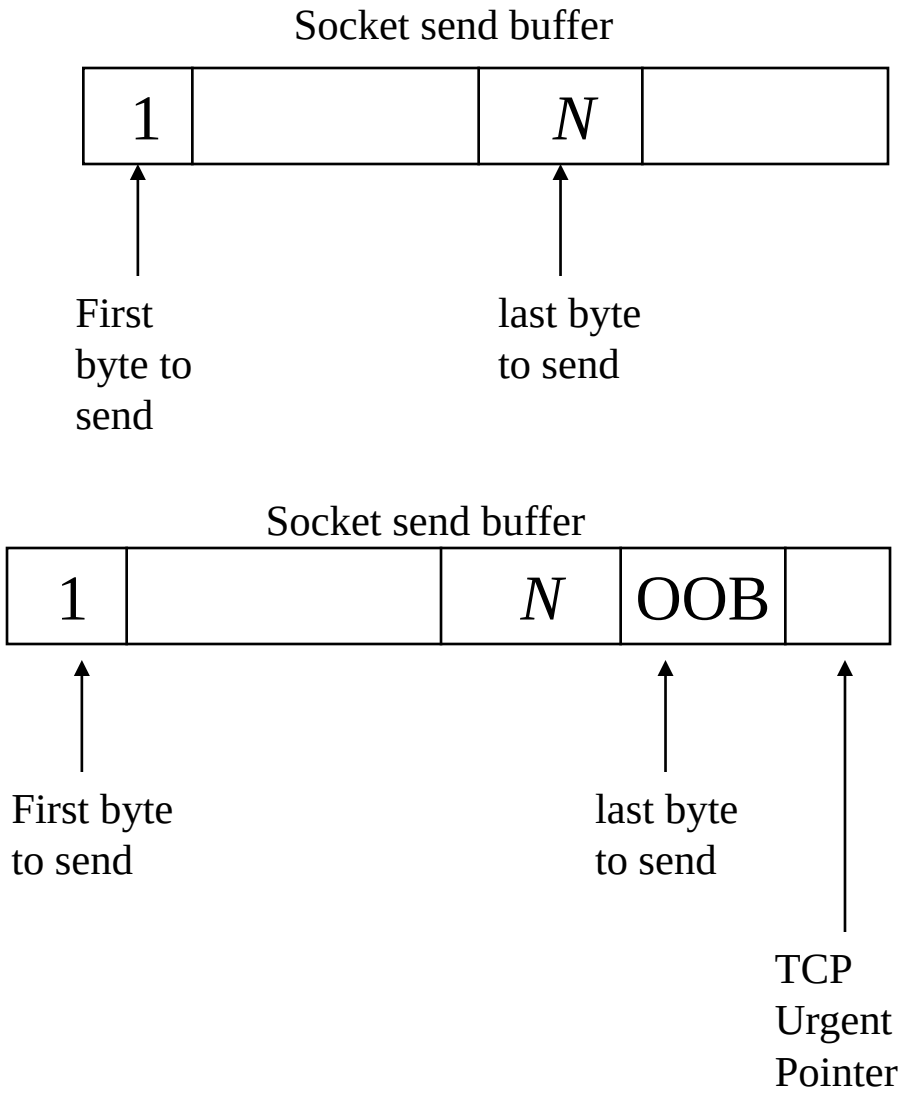
TCP Out-of-Band Data 1/5

- TCP does not have a true out-of-band data mode
- TCP provides an *urgent mode*
- *N* bytes in TCP socket send buffer
- Process writes a single byte of out-of-band data
- **send (fd,"a",1,MSG_OOB);**



TCP Out-of-Band Data 2/5

- Next segment sent by TCP will have URG flag set in TCP header
- *Urgent offset* in TCP header points to byte following the out-of-band byte
  - Add urgent offset to sequence number field to obtain value of urgent pointer
- Segment may or may not contain the byte labeled as OOB
- Depends on number of bytes ahead of it, segment size, and current receiver window



TCP Out-of-Band Data 3/5

- TCP header indicates that sender has entered urgent mode (actual byte of data referred to by urgent pointer need not be sent)
- IF sending TCP is stopped by flow control
  - *Urgent notification is sent without any data*
  - One of the reasons why applications use TCP's urgent mode
- If multiple bytes are sent out-of-band
  - **send (fd,"abc",3,MSG_OOB);**
  - Urgent pointer points one beyond the final byte → last byte is considered the out-of-band byte

TCP Out-of-Band Data 4/5

- Receiver's response to out-of-band data
  - TCP Checks urgent pointer to see if it refers to new out-of-band data (TCP can send multiple segments containing URG flag, but referring to same byte of data)
  - Only first segment causes receiving process to be notified
  - **SIGURG** signal delivered to socket owner
  - If process blocked in a call to **select** (waiting for an exception condition), **select** returns
  - Only one OOB mark, if a new OOB byte arrives before old is read, old byte is discarded

TCP Out-of-Band Data 5/5

- Receiver's response to out-of-band data
  - Actual OOB byte can be pulled out-of-band or left inline
  - **SO_OOBINLINE** socket option (by default not set)
    - ✓Byte not placed in socket receive buffer
    - ✓Byte placed into a separate one-byte out-of-band buffer for this connection
    - ✓To read from that buffer, use **recv** and specify **MSG_OOB** flag
  - If **SO_OOBINLINE** socket option is set
    - ✓Byte left in normal socket receive buffer
    - ✓Process knows when it reaches this byte of data by checking the *out-of-band mark* for this connection

Simple TCP OOB Data Example 1/2

Handles **SIGURG** scenario

- Source code in **oob/tcpsend01.c** and **oob/tcprecv01.c**
- Nine bytes are sent, with a one-second **sleep** between each output operation
- Receiver establishes signal handler for **SIGURG**, and uses **fnctl** function to set the owner of the connected socket
  - **F_SETOWN** command sets the socket owner (the process ID to receive **SIGURG** (see section 7.11))
  - **SIGURG** (and **SIGIO**) are generated for a socket *only if the socket has been assigned an owner*
  - When a new socket created by calling socket, it has no owner
  - When a new socket created from listening socket
    - Socket owner inherited from listening socket by connected socket

Simple TCP OOB Data Example 2/2

Handles **select** scenario

- (Could be a Problem) source code in **oob/tcprecv02.c**
  - **select** indicates an exception condition until the process reads beyond the out-of-band data
  - Can not read the out-of-band data more than once
  - After first read, kernel clears the one-byte out-of-band buffer
  - When call **recv** with **MSG_OOB** flag the second time, it returns **EINVAL**
  - The problem is reproducible on Solaris platforms, not on Linux platforms. Attempted on (SunOS <MC name> 5.9 Generic_112233-07 sun4u sparc SUNW,Sun-Blade-1000) and (Linux <MC name> 2.6.14-1.1656_FC4smp #1 SMP <Date> i686 i686 i386 GNU/Linux)
- Correct source code in **oob/tcprecv03.c**
  - **select** for an exception condition only after reading normal data

sockatmark Function

- Associated out-of-band mark, when out-of-band data is received
  - Position in normal stream of data *at the sender*, when sending process sent out-of-band byte
- Determined by calling **sockatmark** function

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

int sockatmark (int sockfd)

//Returns: 1 if at out-of-band mark, 0 if not at mark, -1 on
  error
```

- Out-of-band mark applies regardless of whether receiving process is receiving OOB data inline or out-of-band

sockatmark Function Example

- Source code in **oob/tcpsend04.c** and **oob/tcprecv04.c**
- Call **sockatmark** to determine when out-of-band byte is encountered
- Out-of-band marks always points one beyond the final byte of normal data
  - If received inline → **sockatmark** returns 1 if next byte to be read is the byte sent with **MSG_OOB** flag
  - If received out-of-band → **sockatmark** returns 1 if next byte to be read is the first byte that was sent following the out-of-band
- A read operation stops at the out-of-band mark
- Try this example on lab machines (Linux) → what do you conclude?
- Modify this example to receive out-of-band not inline

Another OOB Example 1/2

- Illustrates two features
  - TCP sends notification of OOB data, *even though it is stopped by flow control from sending data*
  - A receiving process can be notified about OOB data *before the OOB data arrives!*
- Source code in **oob/tcpsend05.c** and **oob/tcprecv05.c**
  - Sending process sets the size of socket send buffer to 32,768, writes 16,384 bytes of normal data, and then sleeps for 5 seconds
  - Receiver sets socket receive buffer to 4,096 bytes → What will happen?
  - Sender sends 1 byte of OOB data, followed by 1,024 bytes of normal data, and terminates

Another OOB Example 2/2

- **SIGURG** signal is caught
  - OOB data was detected and receiving process notified
- Receiver calls **recv** specifying **MSG_OOB** flag
- OOB byte not available to be read (since it was not transmitted yet)
  - generate **EWOULDBLOCK** error
  - **recv error: Resource temporarily unavailable**

Yet Another OOB Example

- Only one OOB mark for a given TCP connection
- If new OOB data arrives before the receiving process reads existing OOB data, previous mark lost
- Source code in **oob/tcpsend06.c** and **oob/tcprecv06.c**
- Arrival of second OOB byte overwrites the mark stored when first OOB byte arrived

TCP OOB Data Recap

- OOB data conveys 3 different pieces of information to receiver
  - Sender went into urgent mode (notification transmitted immediately after sender sends OOB byte)
  - Existence of an OOB mark
  - Actual value of OOB byte
- Usefulness of OOB data depends on why it is being used by the application
  - Special mode of processing for any data it receives after the OOB
  - Discard all data up to the OOB mark