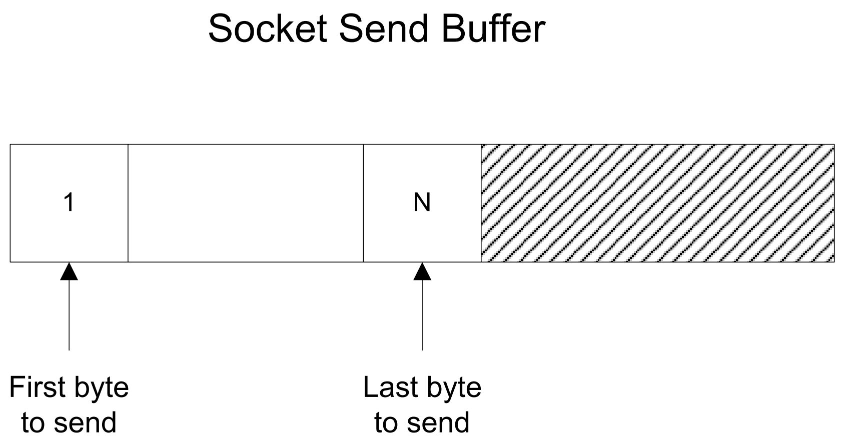
# TCP Out-of-Band Data

- The stream socket abstraction includes out-of-band data, which is a logically independent transmission channel between a pair of connected stream sockets.
- **Out-of-band** (**OOB**) data is delivered independently of normal data. This can be used by one side of a connection to **notify** its peer of an **important** or **urgent event**.
- This notification is sent before any "**normal**" (or "**in-band**") data already in the queue to be delivered.
- The out-of-band data facilities must support the reliable delivery of at least one out-of-band message at a time.
- This message can contain at least one byte of data, and at least one message may be pending delivery at any time.
- For communications protocols that support only in-band signaling (that is, urgent data is delivered in sequence with normal data), the message is extracted from the normal data stream and stored separately.
- This allows users to choose between receiving the urgent data in order and receiving it out of sequence, without having to buffer the intervening data.
- TCP does not have true OOB data. Instead, TCP provides an urgent mode to implement this functionality.

## Sending OOB Data

- To send an out-of-band message, the MSG_OOB flag is applied to ***send()***.
- A logical mark is placed in the data stream at the point at which the out-of-band data was sent.
- Assume that a process has written N bytes of data into a TCP socket and that data is queued in the socket send buffer, waiting to be sent to its peer on the other end of the connection.

- The following diagram illustrates the state of buffer and all its associated pointers after receiving data bytes 1 to N..



- The process now writes a **single byte** of OOB data (say and ASCII character "a") using the ***send*** function with the **MSG_OOB** flag set as follows:

  ***send (fd, "a", 1, MSG_OOB);***

- Following the send call, TCP places the data in the next available location in the send buffer and sets the urgent pointer for this connection to that location.

- The following diagram illustrates the state of the send buffer after 1 byte of OOB is written into it.

- The next segment sent by TCP will have its URG flag set in the TCP header and the urgent offset pointer field in the TCP header will point to the byte following the OOB byte.

- However, this segment may or may not contain the byte that is labeled as OOB. Whether the OOB byte is sent depends on the number of bytes ahead of it in the send buffer, the segment size, and the current window advertised by its peer.

- This is an important characteristic of TCP's urgent mode: the TCP header indicates that the sender has entered urgent mode, but the actual byte of data referred to by the urgent pointer need not be sent.

- In fact, if the sending TCP is stopped due to flow control (invoked by the receiver), the urgent notification is sent without any data.

- The urgent notification is always sent to the peer TCP even if the data flow has been stopped by flow control.

- Note what happens if the sending process sends multiple bytes of OOB as follows:

  ***send (fd, “abc”, 3, MSG_OOB);***

- Now the TCP urgent pointer points one beyond the final byte; that is, the final byte (“c”) is considered the OOB byte.

## Receiving OOB Data

- To receive out-of-band data, **MSG_OOB** is set in the ***recv()*** call.
- When TCP receives a segment with the URG flag set, the urgent pointer is examined to see whether this pointer refers to new OOB data.
- Typically the sending TCP peer wills end multiple segments with the URG flag set, but with the urgent pointer pointing to the same byte of data.
- Only the first of these segments causes the receiving process to be notified that new OOB data has arrived.
- The receiver is notified when a new urgent pointer arrives; a **SIGURG** signal is generated when the protocol is notified of its existence.
- First the SIGURG signal is sent to the owner of the socket, assuming either ***fcntl*** or ***ioctl*** has been called to establish the owner of the socket.
- A process can set the process group or process id to be informed by SIGURG with the appropriate ***fcntl()*** call for SIGIO.

- If multiple sockets have out-of-band data waiting delivery, a **select()** call for exceptional conditions can be used to determine the sockets with such data pending.
- If the process is blocked on a ***select()*** call waiting for the socket descriptor to have an exception condition, ***select*** will unblock.
- When the actual byte of data pointed to by the urgent pointer arrives at the receiving TCP peer, the data can be extracted OOB or left in-line.
- By default the SO_OOBINLINE socket option is not set for a socket so the single byte is not placed into the socket receive buffer.
- Instead, the data byte is placed into a separate 1-byte OOB buffer for this connection.
- The only way for the process to read from this special 1-byte buffer is to call ***recv()***, or ***recvfrom()*** and specify the MSG_OOB flag.
- However, if the process sets the **SO_OOBINLINE** socket option, then the single byte of data referred to by TCP’s urgent pointer is left in the normal socket receive buffer.

- In this case the process cannot specify the MSG_OOB flag to read the data byte. The process will know when it reaches this byte of data by checking the OOB mark for this connection.

- Whenever OOB is received, there is an associated OOB mark. This is the position in the normal stream of data at the sending side when the sending process sent the OOB data.

- The **SIOCATMARK** ***ioctl*** tells whether the read pointer currently points at the mark in the data stream:

  ***ioctl (sd, SIOCATMARK, &flag);***

- If **flag** is 1 on return, the next read returns data after the mark. Otherwise, assuming out-of-band data has arrived, the next read provides data sent by the client before sending the out-of-band signal.

- Consider the code fragment below. This code reads the normal data up to the mark (to discard it), then reads the out-of-band byte.

```
#include <sys/ioctl.h>
#include <sys/file.h>
...
oob()
{
    int out = FWRITE;
    char waste[BUFSIZ];
    int mark;

    /* flush local terminal output */
    ioctl (1, TIOCFLUSH, (char *) &out);

    while(1)
    {
            if (ioctl (rem, SIOCATMARK, &mark) == -1)
            {
                    perror ("ioctl");
                    break;
            }
            if (mark)
                    break;
            (void) read (rem, waste, sizeof (waste));
    }
    if (recv (rem, &mark, 1, MSG_OOB) == -1)
    {
            perror ("recv");
            ...
    }
    ...
}
```

- The receiving process determines whether or not it is at the OOB mark by calling the ***sockatmark()*** function while reading from the socket.

- The ***sockatmark*()** function replaces the historical **SIOCATMARK** command to ***ioctl*()** which implemented the same functionality.

- Using a **wrapper function** follows the adopted conventions to avoid specifying commands to the ***ioctl*()** function.

- The ***sockatmark*()** function could be implemented as follows:

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

int sockatmark(int s)
{
        int val;
        if (ioctl(sd,SIOCATMARK,&flag)< 0)
                return(-1);
        return(flag != 0 ? 1: 0);
}
```

- A process can also read or peek (with **MSG_PEEK**) at the out-of-band data without first reading up to the mark.

- This is more difficult when the underlying protocol delivers the urgent data in-band with the normal data, and only sends notification of its presence ahead of time (for example, TCP, the protocol used to provide socket streams in the Internet domain).

- With such protocols, the out-of-band byte may not yet have arrived when a recv() is done with the MSG_OOB flag. In that case, the call returns the error of EWOULDBLOCK.

- Also, there may be enough in-band data in the input buffer that normal flow control prevents the peer from sending the urgent data until the buffer is cleared.

- The process must then read enough of the queued data before the urgent data can be delivered.

- There is also a facility to retain the position of urgent in-line data in the socket stream.

- This is available as a socket-level option, **SO_OOBINLINE**. With this option, the position of urgent data (the mark) is retained, but the urgent data immediately follows the mark in the normal data stream returned without the **MSG_OOB** flag.

- Reception of multiple urgent indications causes the mark to move, but no out-of-band data are lost.