Andrew Hanushevsky: Sendfile()

Goals

- Brief introduction to socket (network) I/O
- Using sendfile() to improve performance
- Avoiding performance issues
  - Short packets

TCP Network I/O

- Ethernet interface is usually a socket
  - Sockets are blocking devices
    - Sometimes ready sometimes not
      - When not ready can return 0 to <requested bytes
        - » Need to continue I/O until all bytes read or written
    - Can be opened **O_NONBLOCK** (non-blocking)
      - When not ready returns **EWOULDBLOCK**
        - » Retry request until all bytes read or written
      - Generally, **O_NONBLOCK** & threads make little sense
    - Use **poll()** to wait until device is ready
      - Normally for reads and rarely for writes (blocking)

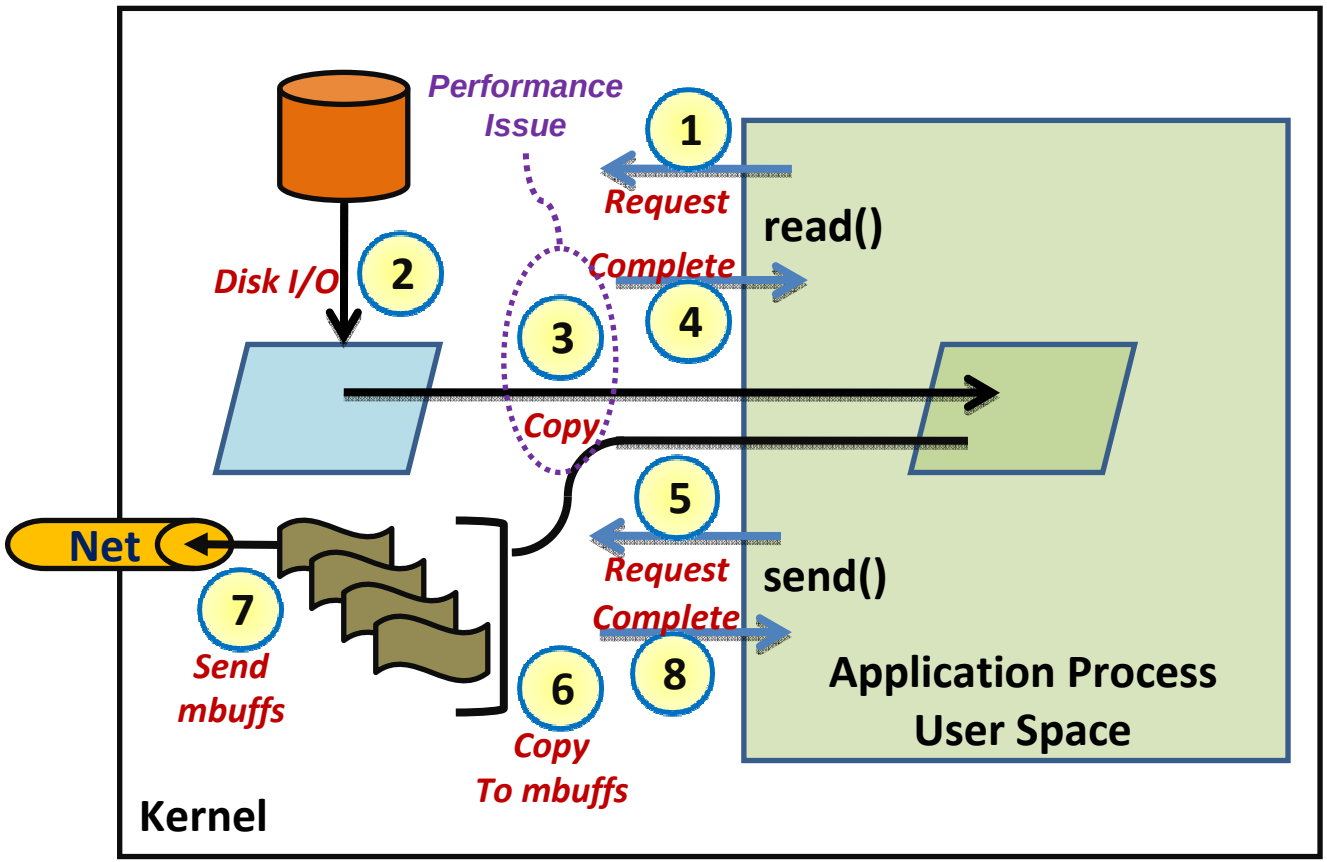
TCP Network I/O (input)

- Many API's to read from socket
  - Most standard interfaces work
    - **read()** and **readv()** (**pread()** *is not valid*)
  - Socket oriented API's also available
    - **recv()**, **recvmsg()**, and **recvfrom()** but *only* for UDP
  - Consult man pages for appropriate usage
- Very difficult to increase efficiency
  - Due to data copying requirements
  - So, program the obvious way

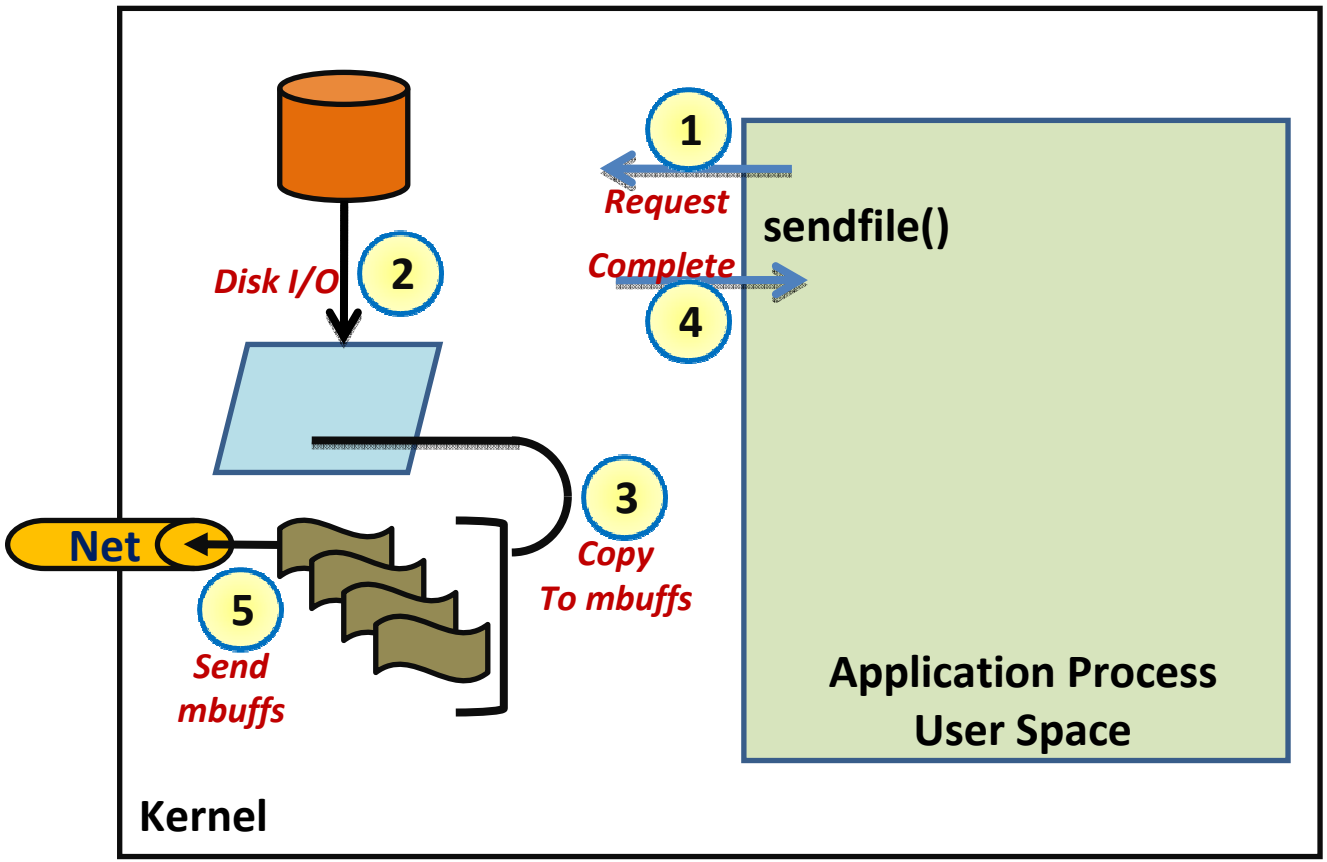
TCP Network I/O (output)

- Many API's to write to socket
  - Most standard interfaces work
    - **write()** and **writev()** (**pwrite()** *is not valid*)
  - Socket oriented API's also available
    - **send()**, **sendmsg()**, and **sendto()** but *only* for UDP
  - Consult man pages for appropriate usage
- Many ways to increase efficiency
  - Goal is to minimize data copying
    - Typically for transfers of data from disk to socket
      - Of great concern for web and file servers
  - This section explores the primary mechanism

The Performance Issue



The Performance Solution



Generic implementation detail; actual implementation is OS specific

Why Not Memory Mapped I/O?

- Actually, some implementations use mmap()
  - On some platforms no performance difference
- Linux implementation uses splice() syscall
  - Change in 2.6.17 kernel
- sendfile() is available in practically all OS's
  - So, generally more portable

sendfile() API

```
#include <sys/sendfile.h>
ssize_t sendfile(int out_fd, int in_fd,
                 off_t *offset, size_t count);
```

- out_fd is a socket file descriptor
- in_fd is a file descriptor for a regular file
- offset offset in the file to start transfer
- count number of bytes to send

- Returns number of bytes sent or -1 on error

What About Framing Data?

- Usually, one needs to send a data header
  - Sometimes trailer data as well
- Easy using writev() for memory mapped files
- But how with sendfile()?
  - No portable solution here
  - Typically issue write() followed by sendfile()
  - And now we have a new performance problem

The Short Packet Problem I

- Data is sent in discreet packets
  - Maximum size called MTU (see **netstat** cmd)
    - Typically, ||data|| + ||TCP/IP headers|| <= 1500
      - Usually leaves about 1460 bytes for application data
- Kernel minimizes sending short packets
  - Maximizes network utilization
  - Minimizes interrupts for sender *and* receiver

The Short Packet Problem II

- Kernel waits for packets to fill
  - Short packet can be delayed up to 500ms
    - Typically, 200ms in Linux
    - Known as the Nagle algorithm
- Kernel hopes more data will arrive
  - Kernel doesn't know if …
    - This is the only packet
    - This is the last of a series of packets
  - This introduces Request/Response latency

Nagle → Bad Performance

- Assume majority sends < 1460 bytes
  - Responsiveness bounded by Nagle delay
    - Typically, 200-300ms which is not speedy at all!
- Many applications turn Nagle delay off
  - **TCP_NODELAY setsockopt()** option
  - Packets are sent immediately after write()
    - Even if they have one byte of data in them!
  - Solves last packet problem
    - Which is usually short but needed by receiver

No Nagle → Bad Performance

- Assume majority sends < 1460 bytes
  - Net utilization bounded by TCP/IP overhead
    - Overhead includes TCP and IP header bytes
    - Ranges from 2.5% to 97% (if average is 50% this is bad)
- Turning Nagle off can be very bad
- Recall that **sendfile()** runs into this problem
  - Short header immediately sent when written
    - This would make **sendfile()** perform badly

Is There A Solution?

- Yes and no!
  - There are many non-portable solutions
  - Each OS has a mechanism dealing with this
    - Linux: **TCP_CORK setsockopt**() option or **MSG_MORE send**() option
    - MacOS: **sendfile()** plus header/trailer iovecs
    - Solaris: **sendfilev()**
  - The only portable solution is **writev()**
    - But does not solve the short last packet problem

TCP_CORK in Linux

- Allows you to temporarily turn on Nagle
  - Implemented in Linux 2.4+
  - Socket needs to have **TCP_NODELAY** set
    - Only possible after Linux 2.5.71
      - Needed other mind-bending games prior to this time
  - Useful for sending header *or* trailers
    - I.e., Framing data in front or back of disk data

TCP_CORK Example

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/sendfile.h>

const int Off = 0, On = 1;

// For expediency we don't use getprotoent() but you should!
//
if (setsockop(fd, IPPROTO_TCP, TCP_NODELAY, (char *)&On, sizeof(On))
   {handle error}

if (setsockopt(fd, SOL_TCP, TCP_CORK, (char *)&On, sizeof(On))
   {handle error}

// For easy reading no errors or partial writes are handled!
//
send(fd, hdr, hdrlen, 0);
sendfile(fd, dfd, &offset, numbytes);

if (setsockopt(fd, SOL_TCP, TCP_CORK, (char *)&Off, sizeof(Off))
   {handle error}
```

MSG_MORE in Linux

- Allows you to temporarily turn on Nagle
  - Implemented in Linux 2.4.4
  - Socket needs to have **TCP_NODELAY** set
  - Useful for sending headers *not* trailers
    - I.e., Framing data in front of disk data
  - Simpler alternative to **TCP_CORK**
    - **TCP_CORK** persists until cleared (3 syscalls!)
    - **MSG_MORE** applies only to the call at hand
      - Cleared on last byte of **sendfile()** or **send()** w/o option
      - Note: you *cannot* efficiently send a trailer with **sendfile()**

MSG_MORE Example

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/sendfile.h>

const int On = 1;

// For expediency we don't use getprotoent() but you should!
//
if (setsockop(fd, IPPROTO_TCP, TCP_NODELAY, (char *)&On, sizeof(On))
   {handle error}

// For easy reading no errors or partial writes are handled!
//
send(fd, hdr, hdrlen, MSG_MORE);
sendfile(fd, dfd, &offset, numbytes);
```

Other Considerations

- Solving short packet problem only one aspect of network performance
- For WAN's TCP window size another one
  - See **SO_RCVBUF** and **SO_SNDBUF** options of **getsockopt()** and **setsockopt()**
    - Displaced by auto-tuned TCP stacks in some kernels
- High performance TCP tricks outside the scope of this lecture

Conclusion

- The overall easiest and most performant way to send file data across the network
  - Memory mapped files can be equally good
    - But more difficult to deal with
- However…
  - Only useful for servers and network copy programs
    - Something few people actually implement