IP Multiplexing

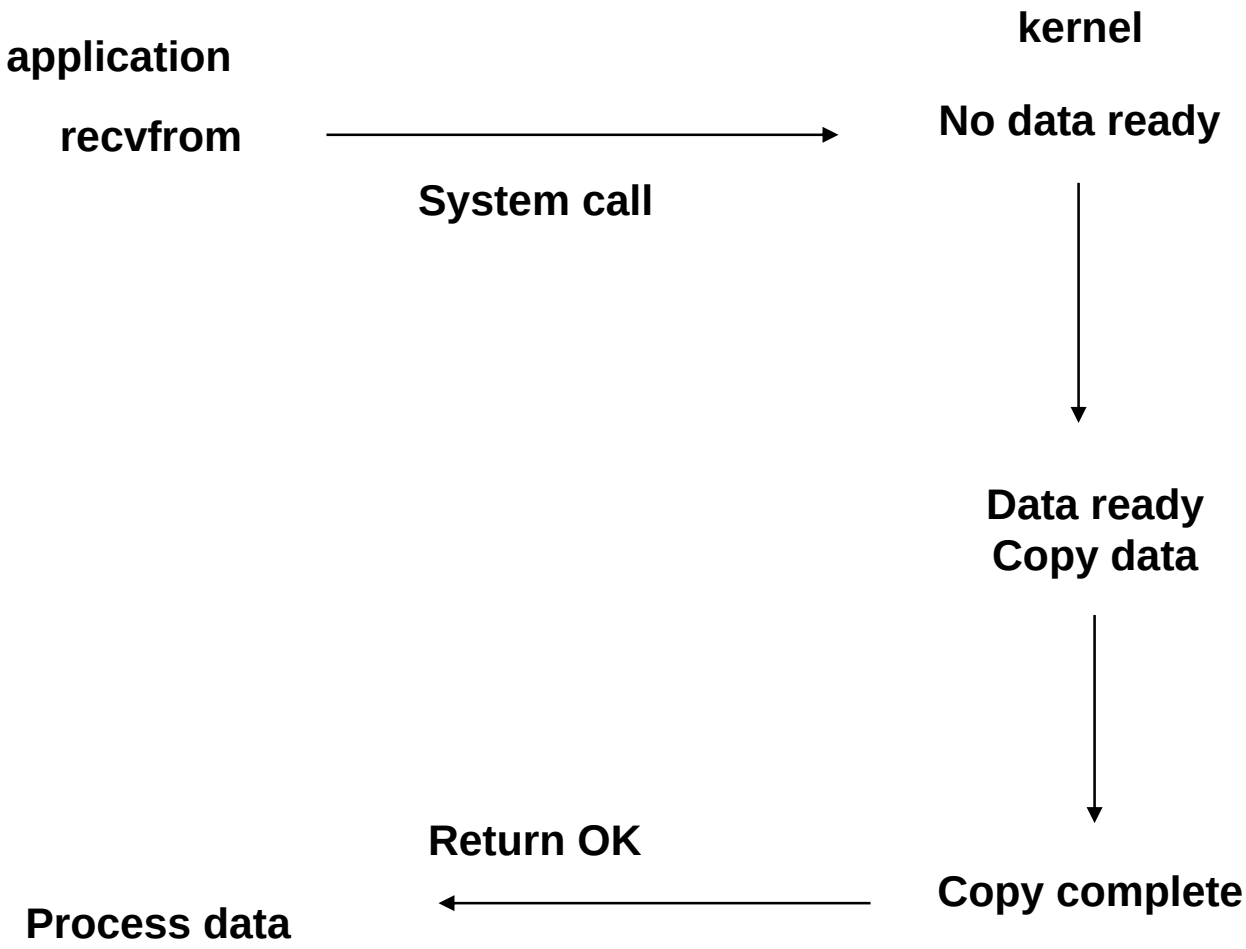
Ying Zhang

EECS 489 W07

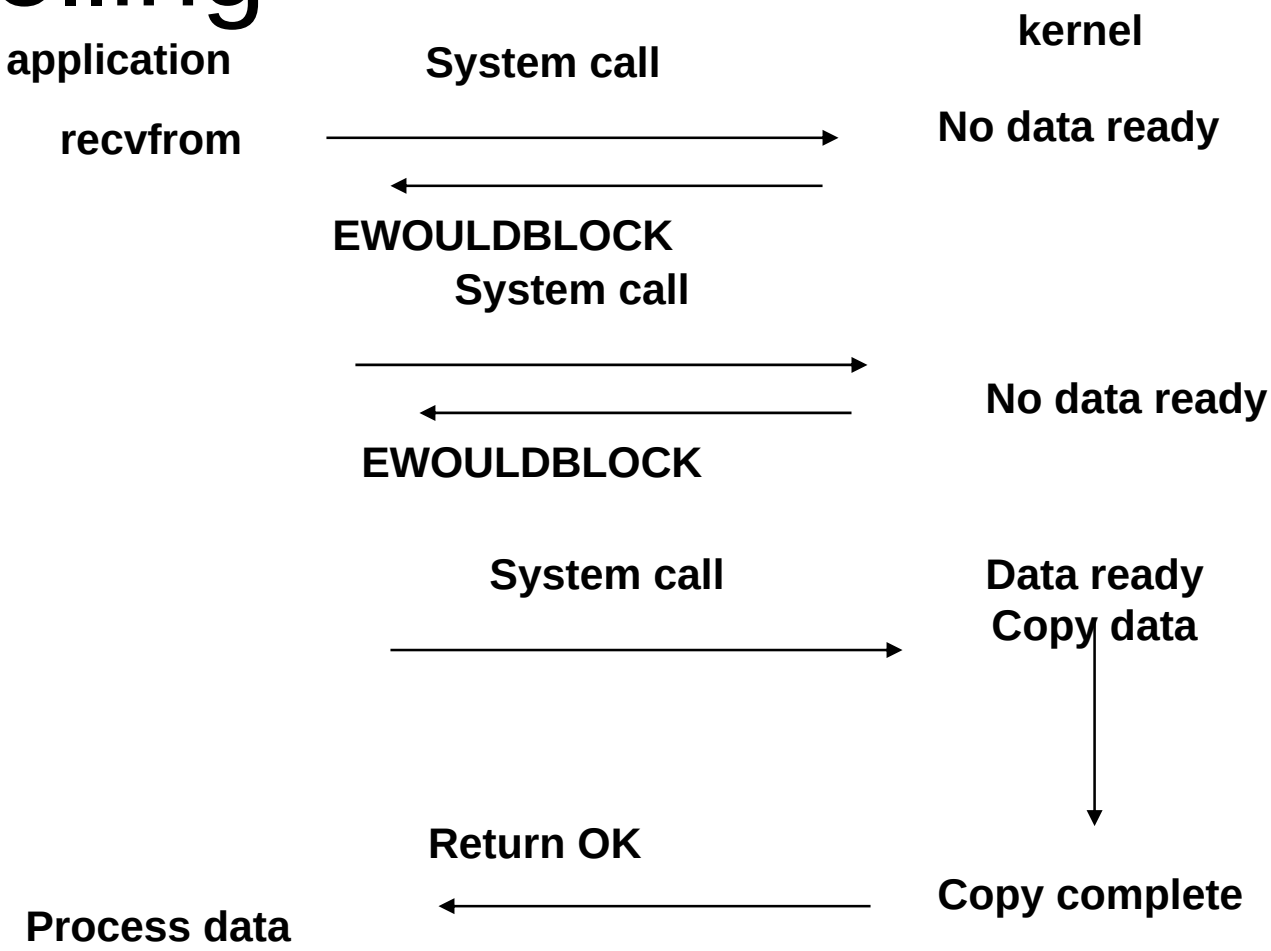
IP Multiplexing

- Multiple descriptors (interactive inputs, sockets)
- Multiple protocols' daemon
- Example:
  - A client two inputs: standard input and TCP socket
  - Block in a call to fgets; server TCP sends a FIN
  - Capability to tell the kernel : " we want to be notified if one or more I/O conditions are ready"
  - IP Multiplexing! (select and poll)

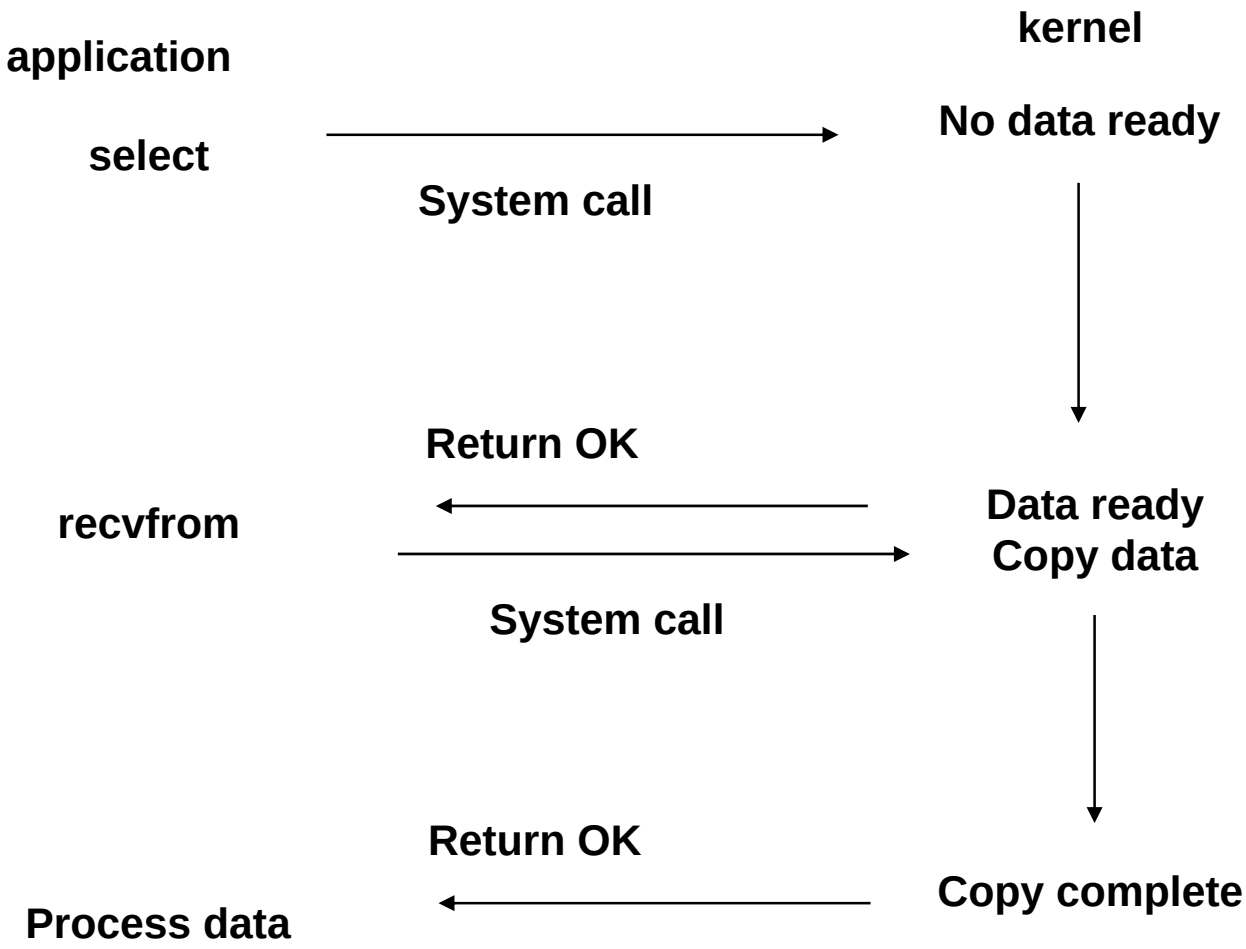
Blocking


application
kernel
recvfrom
No data ready
System call
Data ready
Copy data
Return OK
Copy complete
Process data

Polling


application
kernel
System call
recvfrom
No data ready
EWOULDBLOCK
System call
No data ready
EWOULDBLOCK
System call
Data ready
Copy data
Return OK
Copy complete
Process data

Select


application
kernel
select
No data ready
System call
Return OK
recvfrom
Data ready
Copy data
System call
Return OK
Copy complete
Process data

Select

- int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout);

  It takes these parameters:

- · *int nfds* - The highest file descriptor in all given sets plus one
- · *fd_set *readfds* - File descriptors that will trigger a return when data is ready to be read
- · *fd_set *writefds* - File descriptors that will trigger a return when data is ready to be written to
- · *fd_set *errorfds* - File descriptors that will trigger a return when an exception occurs
- · *struct timeval *timeout* - The maximum period *select()* should wait for an event

- · *FD_ZERO(fd_set *)* - Initializes an *fd_set* to be empty
- · *FD_CLR(int fd, fd_set *)* - Removes the associated fd from the *fd_set*
- · *FD_SET(int fd, fd_set *)* - Adds the associated fd to the *fd_set*
- · *FD_ISSET(int fd, fd_set *)* - Returns a nonzero value if the fd is in *fd_set*
- Upon return from *select()*, *FD_ISSET()* can be called for each fd in a given set to identify whether its condition has been met.

- Any descriptors in set[1,4,5] are ready to read
  - fd_set rset;
  - FD_ZERO(1,&rest)
  - FD_SET(1,&rset)
  - FD_SET(4,&rset)
  - FD_SET(5,&rset)
  - Maxfdp=6
- Any descriptors in set[2,7] are ready to write
- Any descriptors in set[1,4] have an exception condition X

Data ready

- Number of bytes is greater than low-water mark for socket receive buffer
- Read half of the connection is closed(FIN,EOF)
- Listen socket, incoming connection;
- Socket error pending

I/O Multiplexing: Polling

```
int opts = fcntl (sock, F_GETFL);
if (opts < 0) {
    perror ("fcntl(F_GETFL)");
    abort ();
}
opts = (opts | O_NONBLOCK);
if (fcntl (sock, F_SETFL, opts) < 0) {
    perror ("fcntl(F_SETFL)");
    abort ();
}
while (1) {
    if (receive_packets(buffer, buffer_len, &bytes_read) != 0) {
        break;
    }
    if (read_user(user_buffer, user_buffer_len,
                  &user_bytes_read) != 0) {
        break;
    }
}
```

first get current socket option settings

then adjust settings

finally store settings back

get data from socket

get user input

I/O Multiplexing: Select (2)

set up parameters for select()

run select()

interpret result

```
fd_set read_set;
struct timeval time_out;
while (1) {
    FD_ZERO (read_set);
    FD_SET (stdin, read_set); /* stdin is typically 0 */
    FD_SET (sock, read_set);
    time_out.tv_usec = 100000; time_out.tv_sec = 0;
    select_retval = select(MAX(stdin, sock) + 1, &read_set, NULL,
                           NULL, &time_out);
    if (select_retval < 0) {
        perror ("select");
        abort ();
    }
    if (select_retval > 0) {
        if (FD_ISSET(sock, read_set)) {
            if (receive_packets(buffer, buffer_len,
&bytes_read) != 0) {
                break;
            }
        if (FD_ISSET(stdin, read_set)) {
            if (read_user(user_buffer, user_buffer_len,
                          &user_bytes_read) != 0) {
                break;
            }
        }
}
```