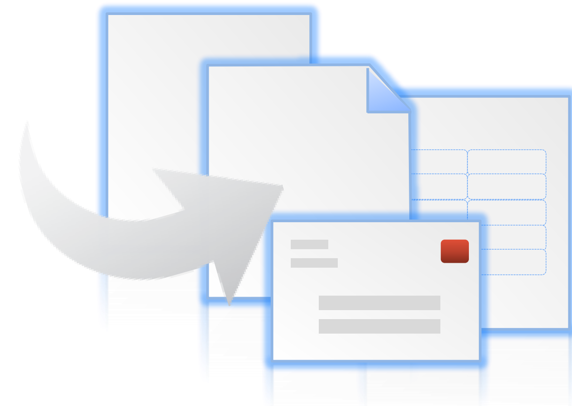
NETWORK PROGRAMMING

I/O MULTIPLEXING: THE SELECT AND POLL FUNCTIONS

BY

MR.PRASAD SAWANT



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

OUT LINE OF UNIT

1. I/O Models, select Function
2. str_cli Function (Revisited),
3. Batch Input
4. shutdown Function
5. str_cli Function (Revisited Again)
6. TCP Echo Server (Revisited)
7. pselect Function
8. poll Function
9. TCPEcho Server (Revisited Again)

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

INTRODUCTION 1/2

TCP echo client is handling two inputs at the same time: standard input and a TCP socket

- when the client was blocked in a call to read, the server process was killed
- server TCP sends FIN to the client TCP, but the client never sees FIN since the client is blocked reading from standard input
  - ✓ We need the capability to tell the kernel that we want to be notified if one or more I/O conditions are ready.
  - ✓ I/O multiplexing (**select**, **poll**, or newer **pselect** functions)

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

INTRODUCTION 2/2

Scenarios for I/O Multiplexing

- client is handling multiple descriptors (interactive input and a network socket).
- Client to handle multiple sockets (rare)
- TCP server handles both a listening socket and its connected socket.
- Server handle both TCP and UDP.
- Server handles multiple services and multiple protocols

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Models

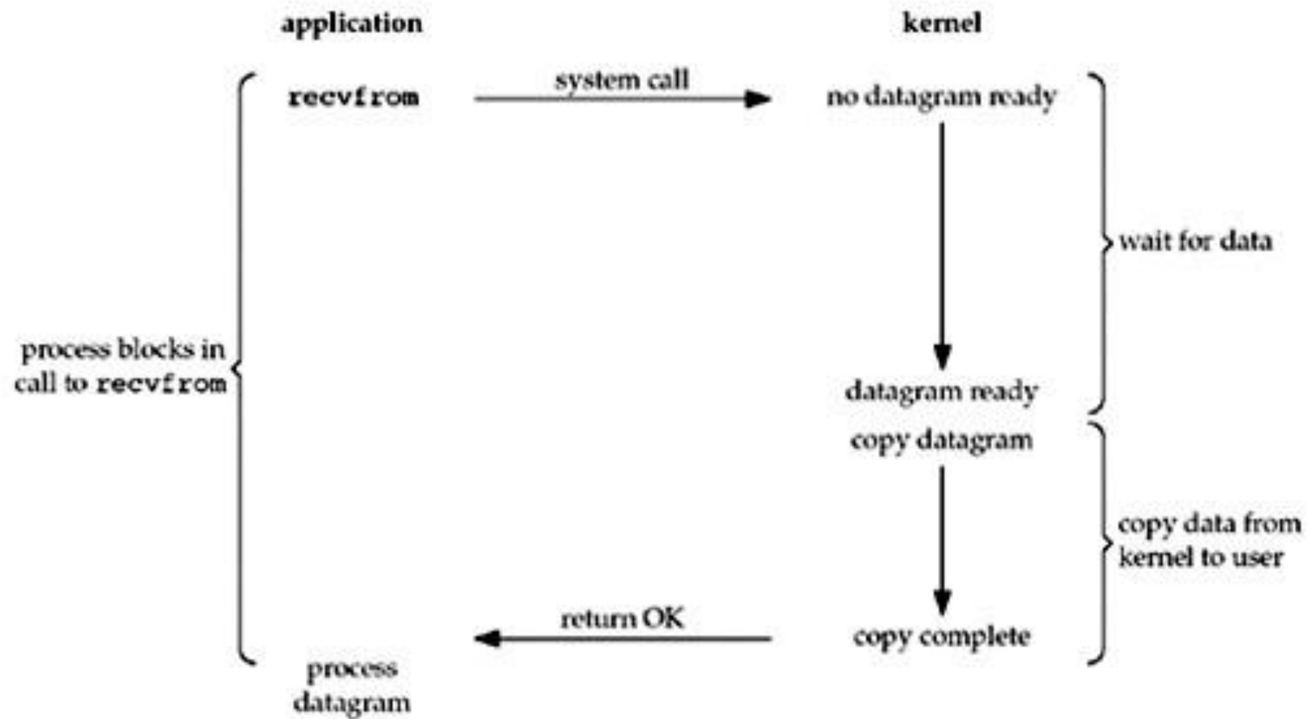
- Blocking I/O
- Nonblocking I/O
- I/O multiplexing(**select** and **poll**)
- Signal driven I/O (**SIGIO**)
- Asynchronous I/O

Two *distinct phases* for an input operation

- Waiting for the data to be ready (for a socket, wait for the data to arrive on the network, then copy into a buffer within the kernel)
- Copying the data from the kernel to the process (from kernel buffer into application buffer)

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

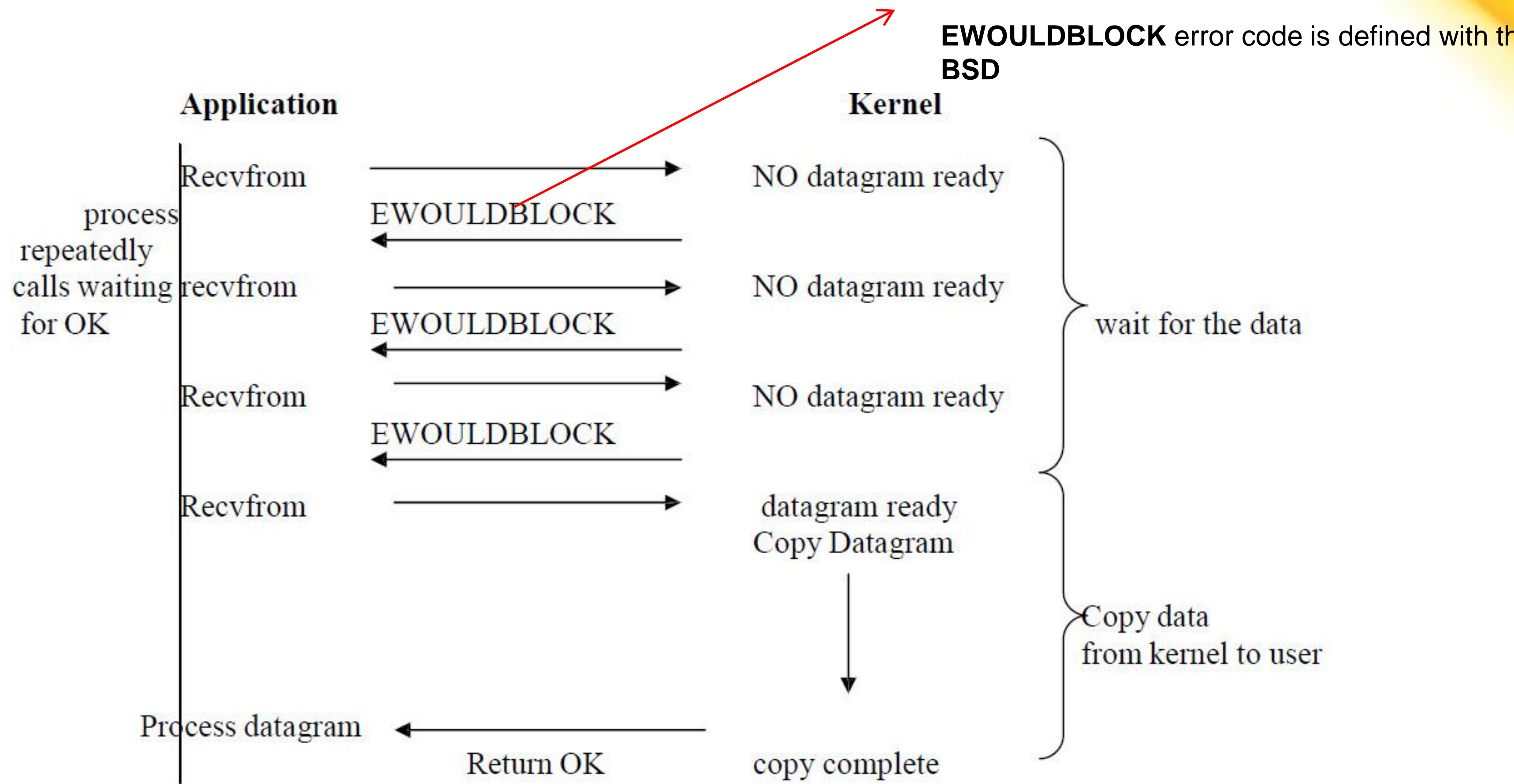
BLOCKING I/O



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

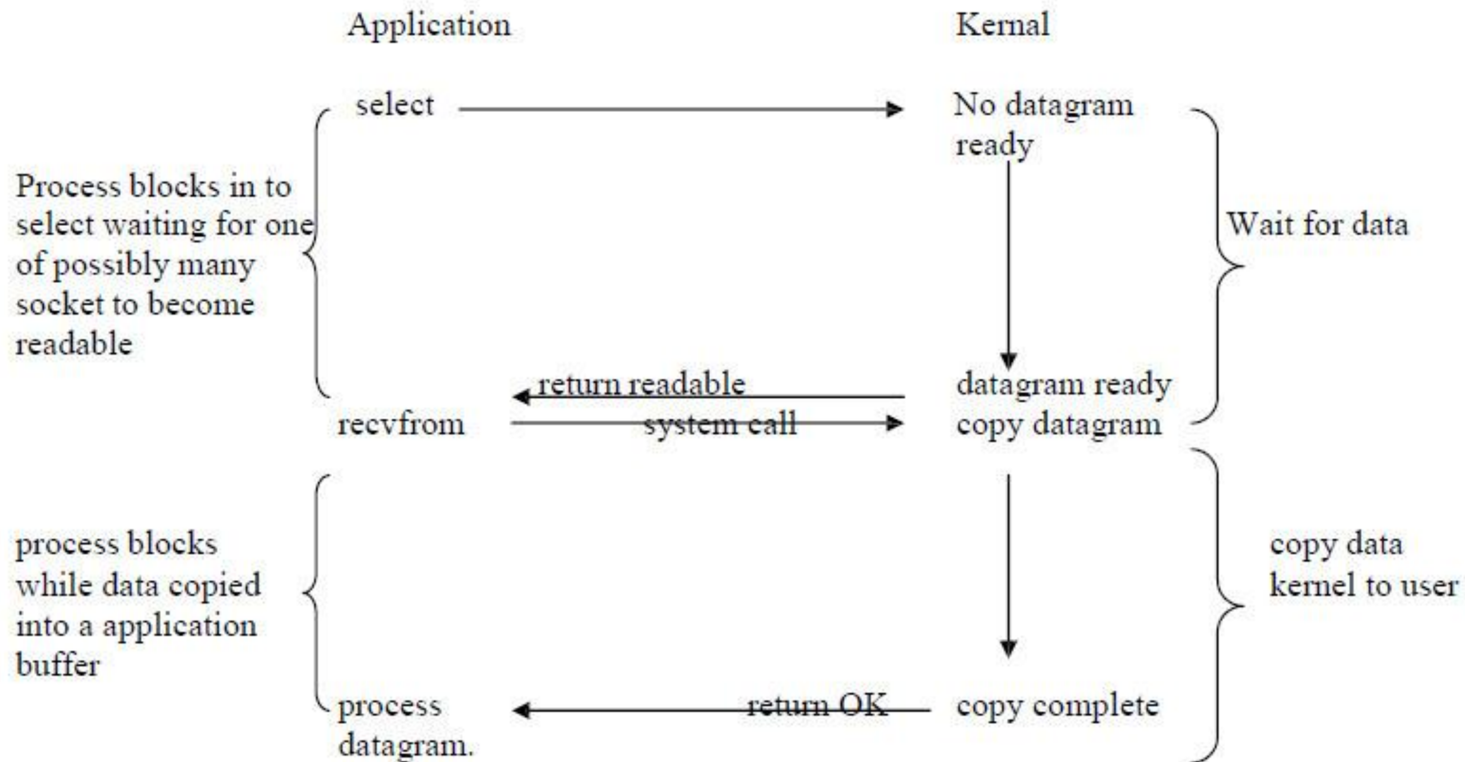
NON BLOCKING I/O

EWOULDBLOCK error code is defined with th
BSD



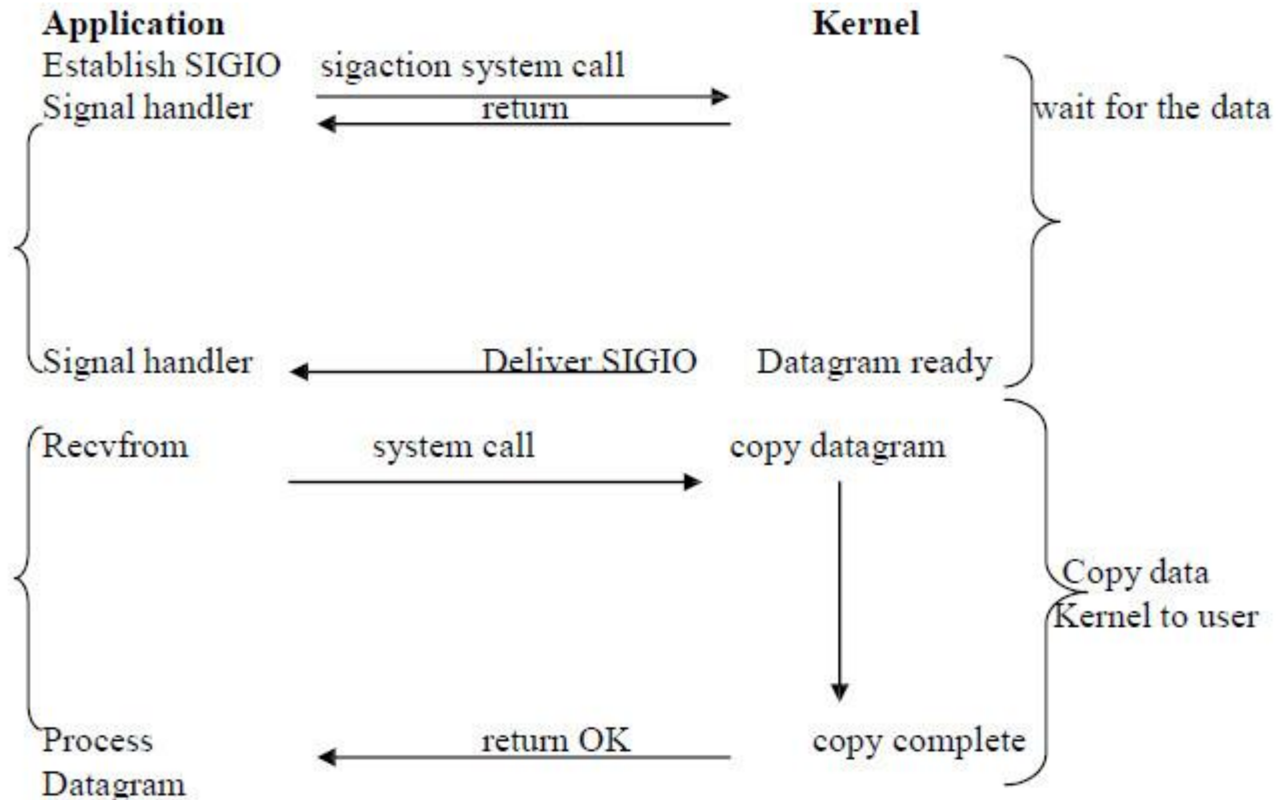
©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

I / O MULTIPLEXING MODEL



I / O Multiplexing Model

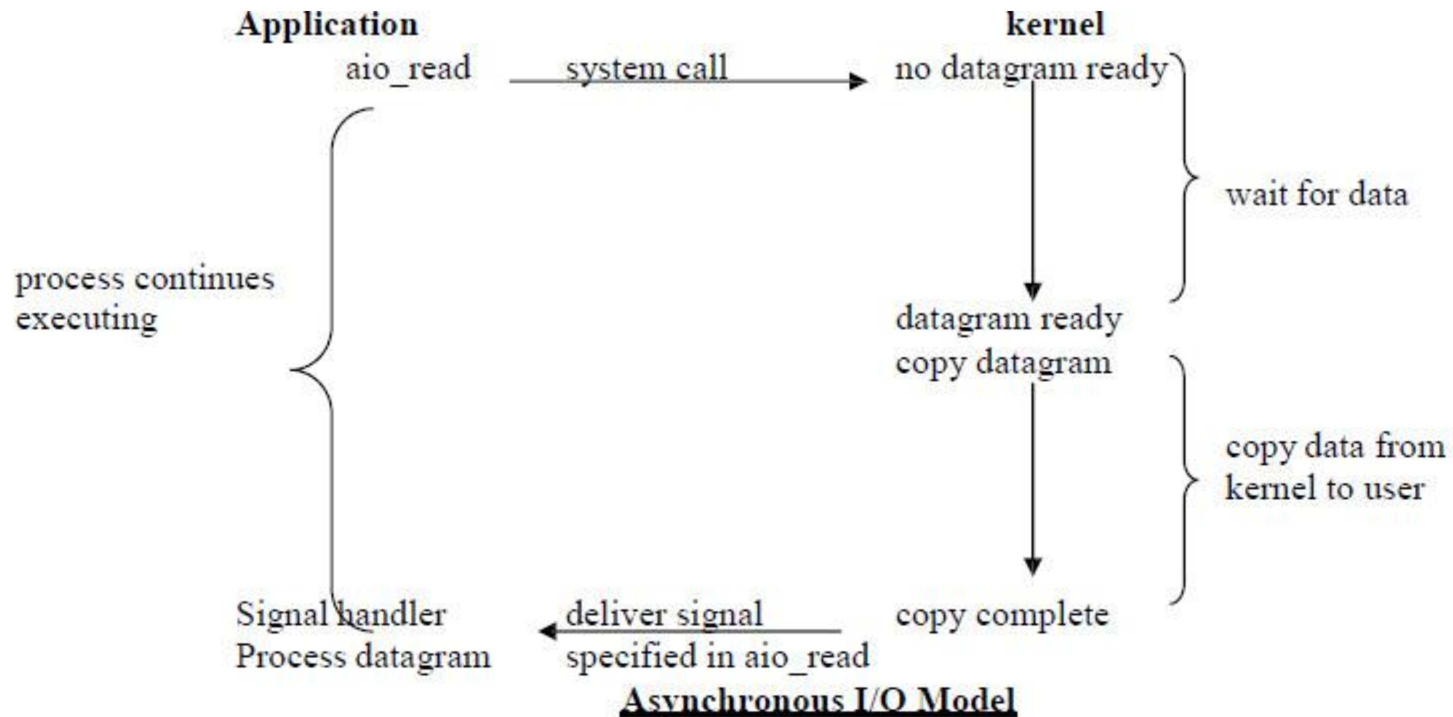
SIGNAL-DRIVEN I/O MODEL



Signal Driven I/O Model

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

ASYNCHRONOUS I/O MODEL



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

SELECT FUNCTION

Allows the process to instruct the kernel to *wait for any one of multiple events to occur* and to wake up the process only when one or more of these events occurs or *when a specified amount of time has passed.*

What descriptors we are interested in (readable ,writable , or exception condition) and how long to wait?

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

#include <sys/time.h>

int select(int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timeval *timeout);

                                      Returns: positive count of ready descriptors, 0 on timeout, -1 on error
```

```
struct timeval  {
  long   tv_sec;          /* seconds */
  long   tv_usec;         /* microseconds */
};
```

POSSIBILITIES FOR SELECT FUNCTION

Wait forever : return only when descriptor (s) is ready (specify **timeout** argument as NULL)

wait up to a fixed amount of time

Do not wait at all : return immediately after checking the descriptors. Polling (specify **timeout** argument as pointing to a **timeval** structure where the timer value is 0)

The wait is normally interrupted if the process catches a signal and returns from the signal handler

- **select** might return an error of **EINTR**
- Actual return value from function = -1

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

SELECT FUNCTION DESCRIPTOR ARGUMENTS

readset → descriptors for checking readable

writeset → descriptors for checking writable

exceptset → descriptors for checking exception conditions (2 exception conditions)

- ✓ arrival of out of band data for a socket
- ✓ the presence of control status information to be read from the master side of a pseudo terminal (Ignore)

If you pass the 3 arguments as NULL, you have a high precision timer than the sleep function

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

DESCRIPTOR SETS

Array of integers : each bit in each integer correspond to a descriptor (**fd_set**)

4 macros

- **void FD_ZERO(fd_set *fdset); /* clear all bits in fdset */**
- **void FD_SET(int fd, fd_set *fdset); /* turn on the bit for fd in fdset */**
- **void FD_CLR(int fd, fd_set *fdset); /* turn off the bit for fd in fdset*/**
- **int FD_ISSET(int fd, fd_set *fdset);/* is the bit for fd on in fdset ? */**

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

EXAMPLE OF DESCRIPTOR SETS MACROS

```
fd_set  rset;

FD_ZERO(&rset);        /*all bits off : initiate*/
FD_SET(1, &rset);      /*turn on bit fd 1*/
FD_SET(4, &rset);      /*turn on bit fd 4*/
FD_SET(5, &rset);      /*turn on bit fd 5*/
```


©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

maxfdp1 argument to **select** function

specifies the number of descriptors to be tested.

Its value is the maximum descriptor to be tested, plus one. (hence maxfdp1)

- Descriptors 0, 1, 2, up through and including **maxfdp1**-1 are tested
- example: interested in **fds** 1,2, and 5 → **maxfdp1** = 6
- Your code has to calculate the **maxfdp1** value

constant **FD_SETSIZE** defined by including **<sys/select.h>**

- is the number of descriptors in the **fd_set** datatype. (often = 1024)

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Value-Result arguments in select function

Select modifies descriptor sets pointed to by **readset**, **writeset**, and **exceptset** pointers

On function call

- Specify value of descriptors that we are interested in

On function return

- Result indicates which descriptors are ready

Use **FD_ISSET** macro on return to test a specific descriptor in an **fd_set** structure

- Any descriptor not ready will have its bit cleared
- You need to turn on all the bits in which you are interested on all the descriptor sets each time you call **select**

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

CONDITION FOR A SOCKET TO BE READY FOR ***SELECT***

Condition	*Readable?*	*writable?*	*Exception?*
Data to read *read-half of the connection closed* *new connection ready for listening socket*	• • •		
Space available for writing *write-half of the connection closed*		• •	
Pending error	•	•	
TCP out-of-band data			•

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

STR_CLI FUNCTION REVISITED

Problems with earlier version

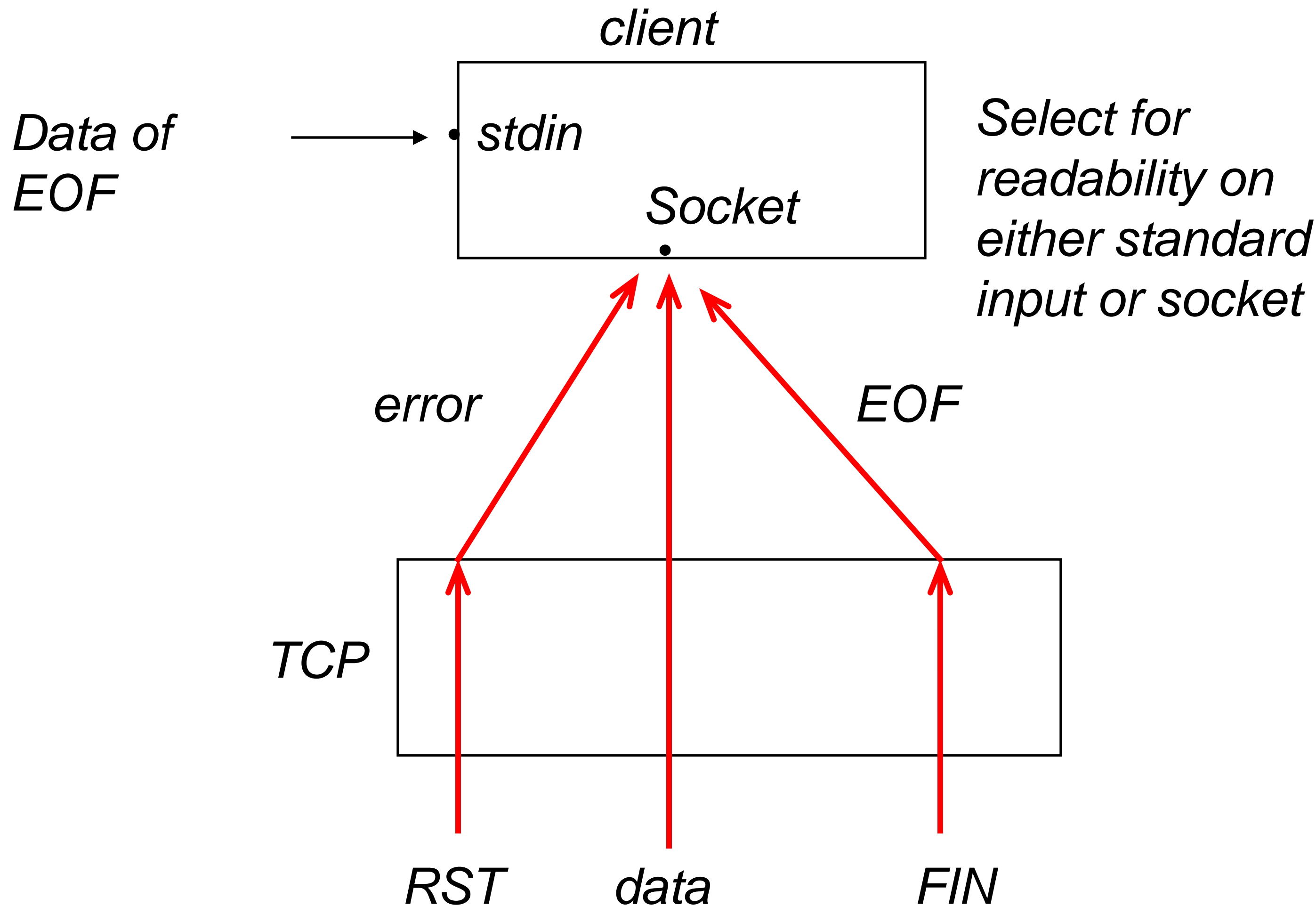
- could be blocked in the call to **fgets** when something happened on the socket
- We need to be notified as soon s the server process terminates

Alternatively

- block in a call to **select** instead, waiting for either standard input or the socket to be readable.

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

CONDITION HANDLED BY **SELECT** IN **STR_CLI**



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

CONDITIONS HANDLED WITH THE SOCKET

Peer TCP sends data

- the socket becomes readable and *read* returns greater than 0 (number of bytes of data)

Peer TCP sends a FIN (peer process terminates)

- the socket become readable and *read* returns 0 (EOF)

Peer TCP sends a RST (peer host has crashed and rebooted)

- the socket become readable and returns -1
- *errno* contains the specific error code

- Source code in **select/strcliselect01.c** tested by **select/tcpcli01.c**
- This version is OK for stop-an-wait mode (interactive input), will modify later for batch input and buffering

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

SELECT-BASED STR_CLI FUNCTION

```
 1 #include    "unp.h"

 2 void
 3 str_cli(FILE *fp, int sockfd)
 4 {
 5     int     maxfdp1;
 6     fd_set  rset;
 7     char    sendline[MAXLINE], recvline[MAXLINE];

 8     FD_ZERO(&rset);
 9     for ( ; ; ) {
10         FD_SET(fileno(fp), &rset);
11         FD_SET(sockfd, &rset);
12         maxfdp1 = max(fileno(fp), sockfd) + 1;
13         Select(maxfdp1, &rset, NULL, NULL, NULL);

14         if (FD_ISSET(sockfd, &rset)) {  /* socket is readable */
15             if (Readline(sockfd, recvline, MAXLINE) == 0)
16                 err_quit("str_cli: server terminated prematurely");
17             Fputs(recvline, stdout);
18         }

19         if (FD_ISSET(fileno(fp), &rset)) {  /* input is readable */
20             if (Fgets(sendline, MAXLINE, fp) == NULL)
21                 return;         /* all done */
22             Writen(sockfd, sendline, strlen(sendline));
23         }
24     }
25 }
```

Call select

Handle readable socket

Handle Readable Input

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

BATCH INPUT AND BUFFERING

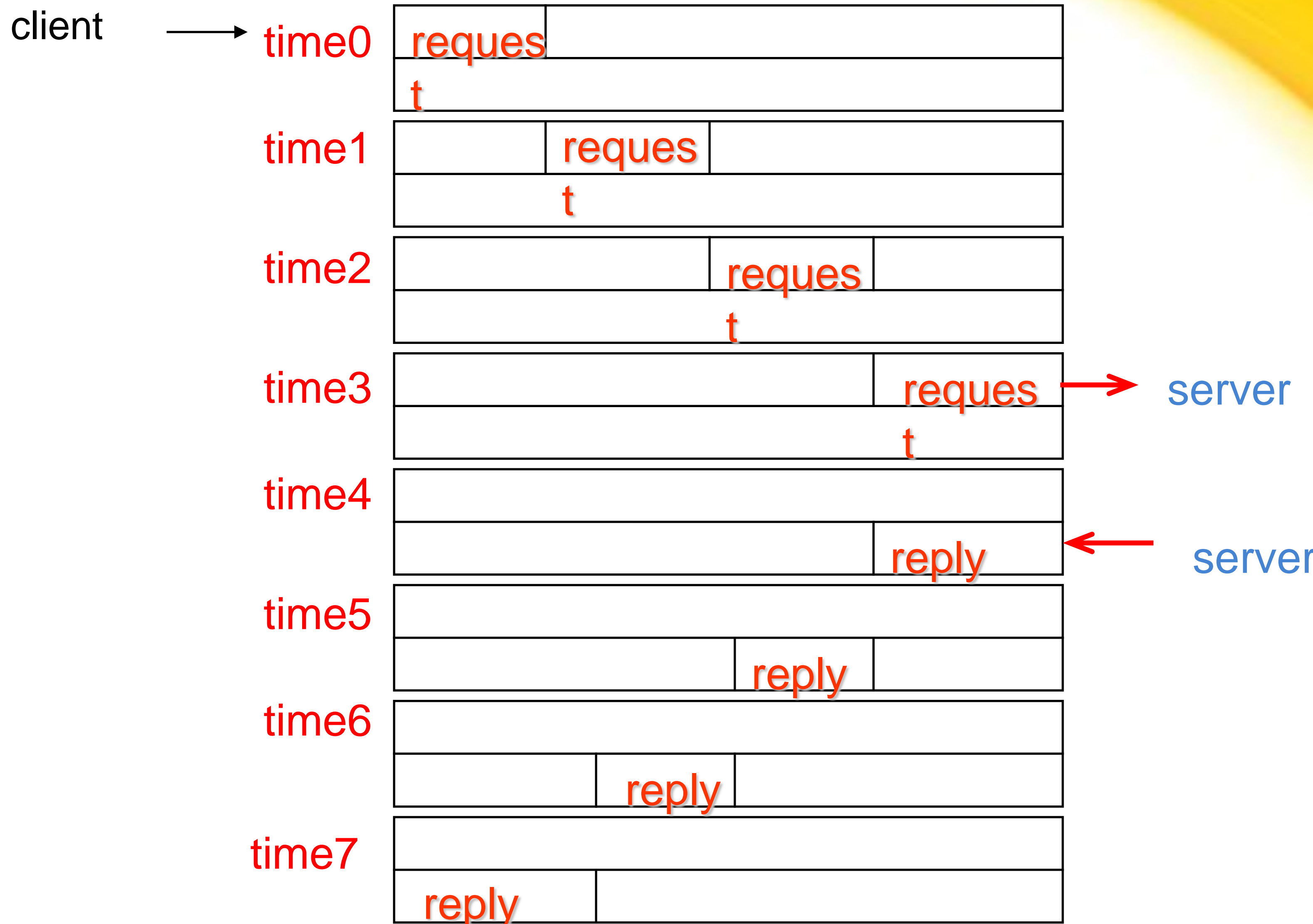
str_cli operates in a stop-and-wait mode → sends a line to the server and then waits for the reply

Assume RTT of 8 units of time → *only using one-eighth of the pipe capacity*

Ignore TCP ACKs

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Batch Input and Buffering



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

BATCH INPUT AND BUFFERING
2/2

With batch input, can send as fast as we can

The problem with the revised **str_cli** function

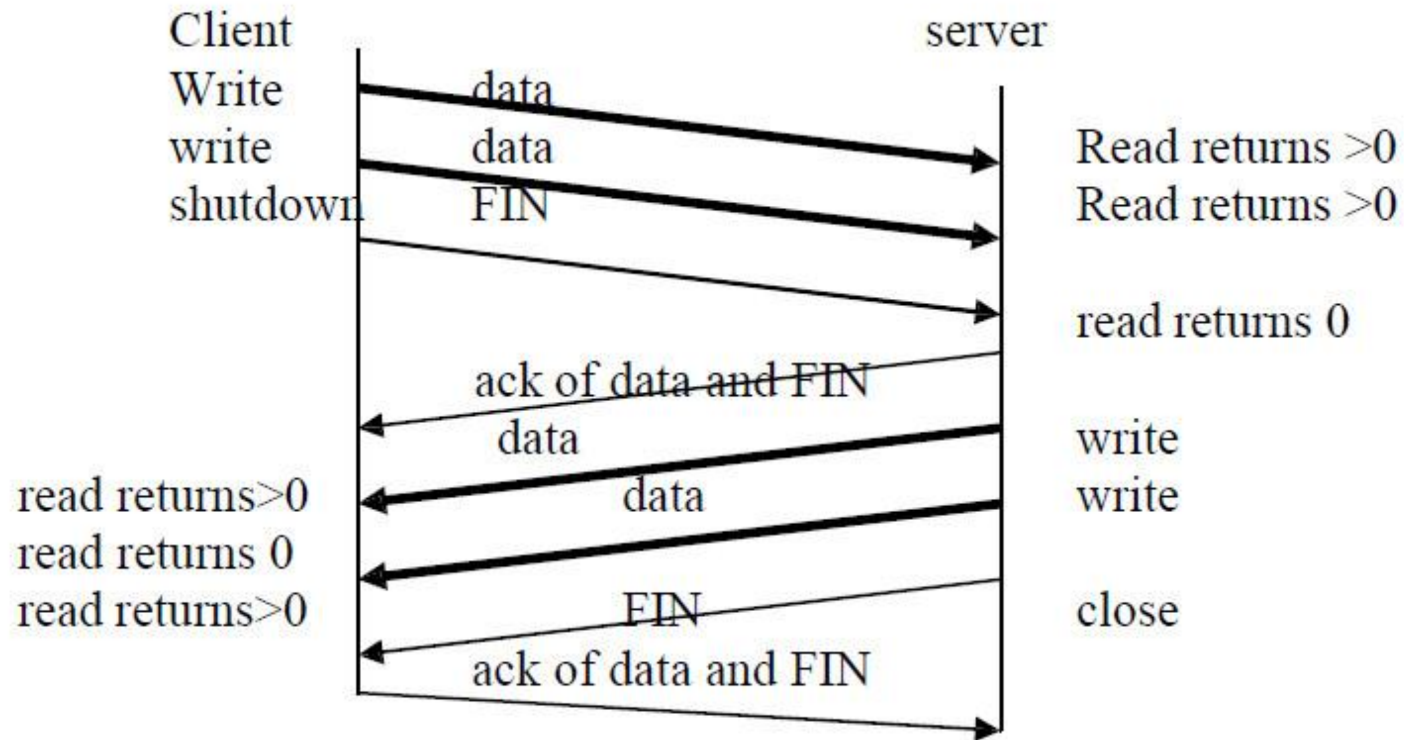
- After the handling of an end-of-file on input, the send function returns to the main function, that is, the program is terminated.
- However, in *batch mode*, there are still other requests and replies in the pipe.

We need a way to close one-half of the TCP connection

- send a **FIN** to the server, telling it we have finished sending data, but leave the socket descriptor open for reading
  - ✓ **shutdown** function

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

SHUTDOWN FUNCTION



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

SHUTDOWN FUNCTION 1/3

Close one half of the TCP connection

- send **FIN** to server, but leave the socket descriptor open for reading

Limitations with **close** function

- decrements the descriptor's reference count and closes the socket only if the count reaches 0
  - ✓ With **shutdown**, can initiate TCP normal connection termination regardless of the reference count
- terminates both directions (reading and writing)
  - ✓ With **shutdown**, we can tell other end that we are done sending, although that end might have more data to send us

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Shutdown function

3/3

```
#include<sys/socket.h>
  int shutdown ( int sockfd, int howto );
  /* return : 0 if OK, -1 on error */
```

howto argument

- **SHUT_RD**
  - read-half of the connection closed
  - Any data in receive buffer is discarded
  - Any data received after this call is ACKed and then discarded
- **SHUT_WR**
  - write-half of the connection closed (half-close)
  - Data in socket send buffer sent, followed by connection termination
- **SHUT_RDWR**
  - both closed

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

STR_CLI FUNCTION (REVISITED AGAIN)

```
 1 #include    "unp.h"

 2 void
 3 str_cli(FILE *fp, int sockfd)
 4 {
 5     int     maxfdp1, stdineof;
 6     fd_set  rset;
 7     char    buf[MAXLINE];
 8     int     n;

 9     stdineof = 0;
10     FD_ZERO(&rset);
11     for ( ; ; ) {
12         if (stdineof == 0)
13             FD_SET(fileno(fp), &rset);
14         FD_SET(sockfd, &rset);
15         maxfdp1 = max(fileno(fp), sockfd) + 1;
16         Select(maxfdp1, &rset, NULL, NULL, NULL);

17         if (FD_ISSET(sockfd, &rset)) {  /* socket is readable */
18             if ( (n = Read(sockfd, buf, MAXLINE)) == 0) {
19                 if (stdineof == 1)
20                     return;        /* normal termination */
21                 else
22                     err_quit("str_cli: server terminated prematurely");
23             }
24             Write(fileno(stdout), buf, n);
25         }
26         if (FD_ISSET(fileno(fp), &rset)) {  /* input is readable */
27             if ( (n = Read(fileno(fp), buf, MAXLINE)) == 0) {
28                 stdineof = 1;
29                 Shutdown(sockfd, SHUT_WR);  /* send FIN */
30                 FD_CLR(fileno(fp), &rset);
31                 continue;
32             }
33             Writen(sockfd, buf, n);
34         }
35     }
36 }
```

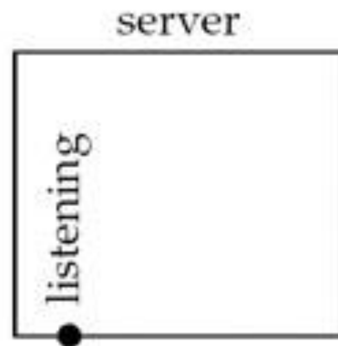
stdineof is a new flag that is initialized to 0.

Normal Termination else error

Shutdown function

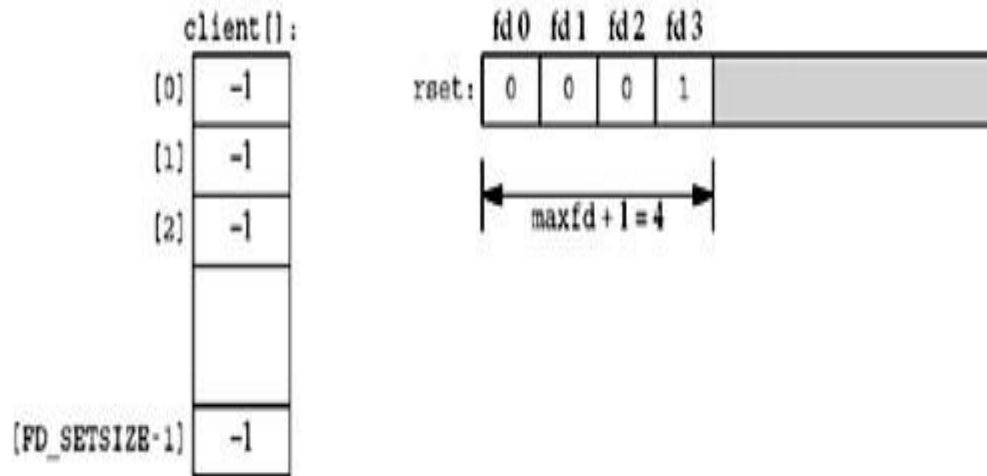
TCP ECHO SERVER WITH IO MULTIPLEXING:

TCP server before first client has established a connection.



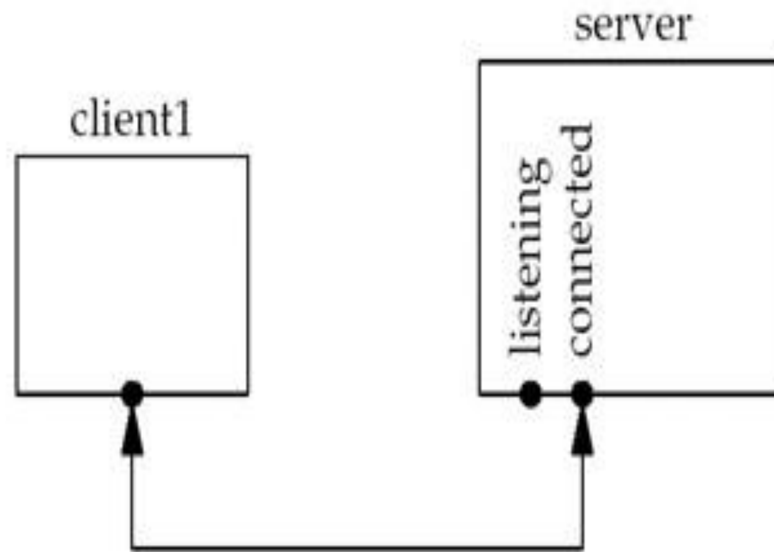
©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Figure 6.15. Data structures for TCP server with just a listening socket.



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Figure 6.16. TCP server after first client establishes connection.



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Figure 6.17. Data structures after first client connection is established.

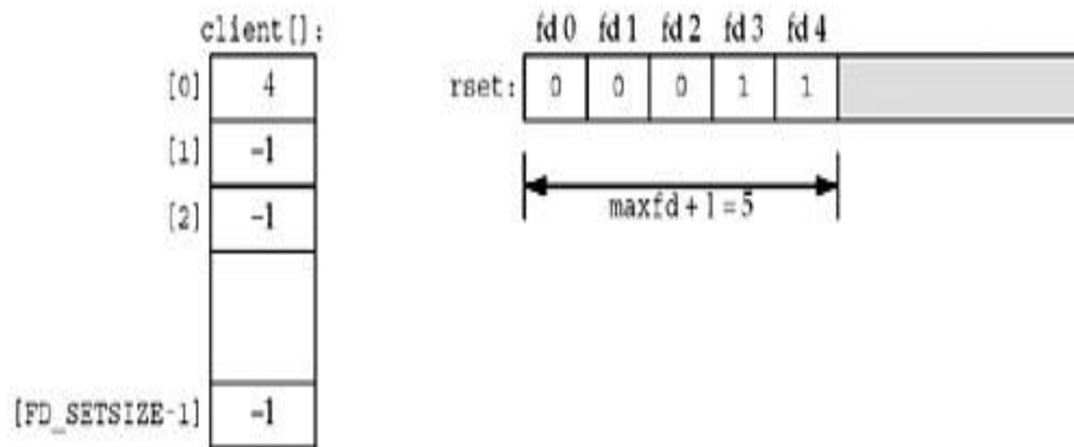
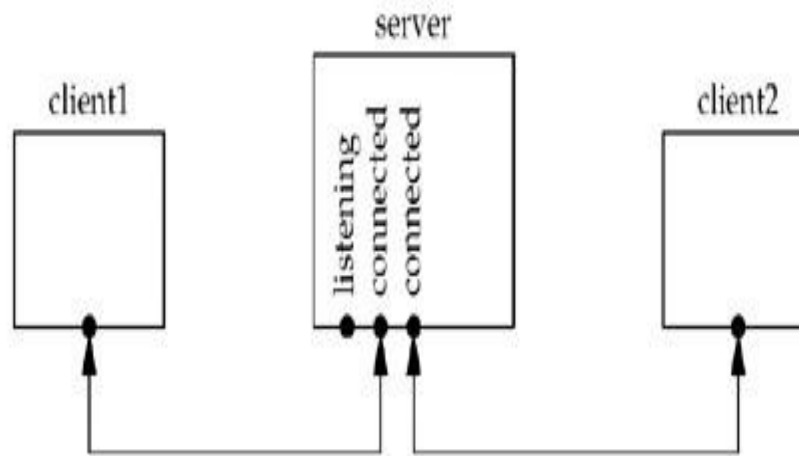


Figure 6.18. TCP server after second client connection is established.



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

Figure 6.19. Data structures after second client connection is established.

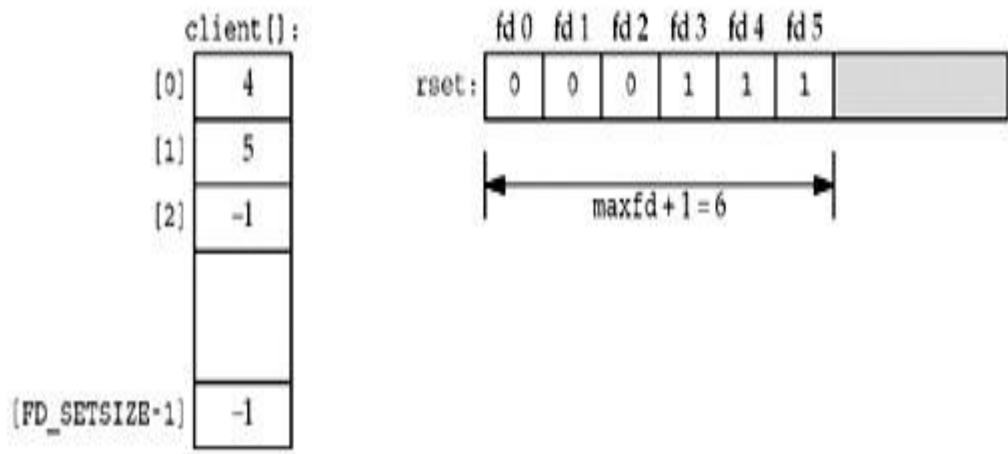
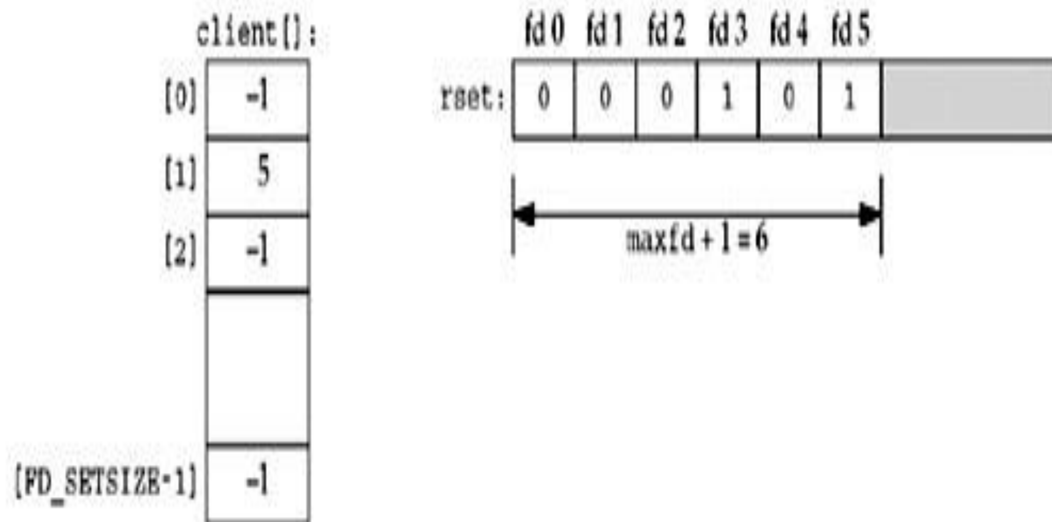


Figure 6.20. Data structures after first client terminates its connection.



©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

TCP ECHO SERVER USING *SELECT* 5/5

As clients arrive, record connected socket descriptor in first available entry in client array (first entry = -1)

Add connected socket to read descriptor set

Keep track of

- Highest index in client array that is currently in use
- Maxfd +1

The limit on number of clients to be served

Min (FD_SETSIZE, **Max** (Number of descriptors allowed for this process by the kernel))

Source code in **tcpcliserv/tcpservselect01.c**

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

PSELECT FUNCTION

```
#include <sys/select.h>
#include <signal.h>
#include <time.h>
int pselect (int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timespec *timeout,
const sigset_t *sigmask);
```

Returns: count of ready descriptors, 0 on timeout, –1 on error

Pselect function

pselect contains two changes from the normal select function:

1.pselect uses the timespec structure, another POSIX invention, instead of the timeval structure.

```
struct timespec
{
time_t tv_sec; /* seconds */
long tv_nsec; /* nanoseconds */
};
```

2. pselect adds a sixth argument: a pointer to a signal mask. This allows the program to disable the delivery of certain signals, test some global variables that are set by the handlers for these now-disabled signals, and then call pselect, telling it to reset the signal mask.

Poll function

```
#include <poll.h>
int poll (struct pollfd *fdarray, unsigned long nfds, int timeout);
```

Returns: count of ready descriptors, 0 on timeout, –1 on error

```
struct pollfd { int fd; /* descriptor to check */
short events; /* events of interest on fd */
short revents; /* events that occurred on fd
*/ };
```

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad

TCP AND UDP ECHO SERVER USING *SELECT*

Section 8.15

Combine concurrent TCP echo server with iterative UDP server into a single server that uses **select** to multiplex a TCP and UDP socket

Source code in **udpcliserv/udpservselect01.c**

Source code for **sig_chld** function (signal handler) is in **udpcliserv/sigchldpidwait.c**

- Handles termination of a child TCP server
- See sections 5.8, 5.9, and 5.10

©Prof.Prasad Sawant ,Assitiant Professor ,Dept. Of CS PCCCS Chichwad