# Socket Programming IO Multiplexing: select()

CSEE, Handong Univ.

Jong-won Lee

*ljw@handong.edu*

## Introduction

Assume that a process is handling two inputs at the same time, that is, standard input and a socket.

If the process calls to read from a standard input, it is blocked until it receives data from the standard input.

That is, the process can not receive data from a socket because it is blocked to read from the standrad input.

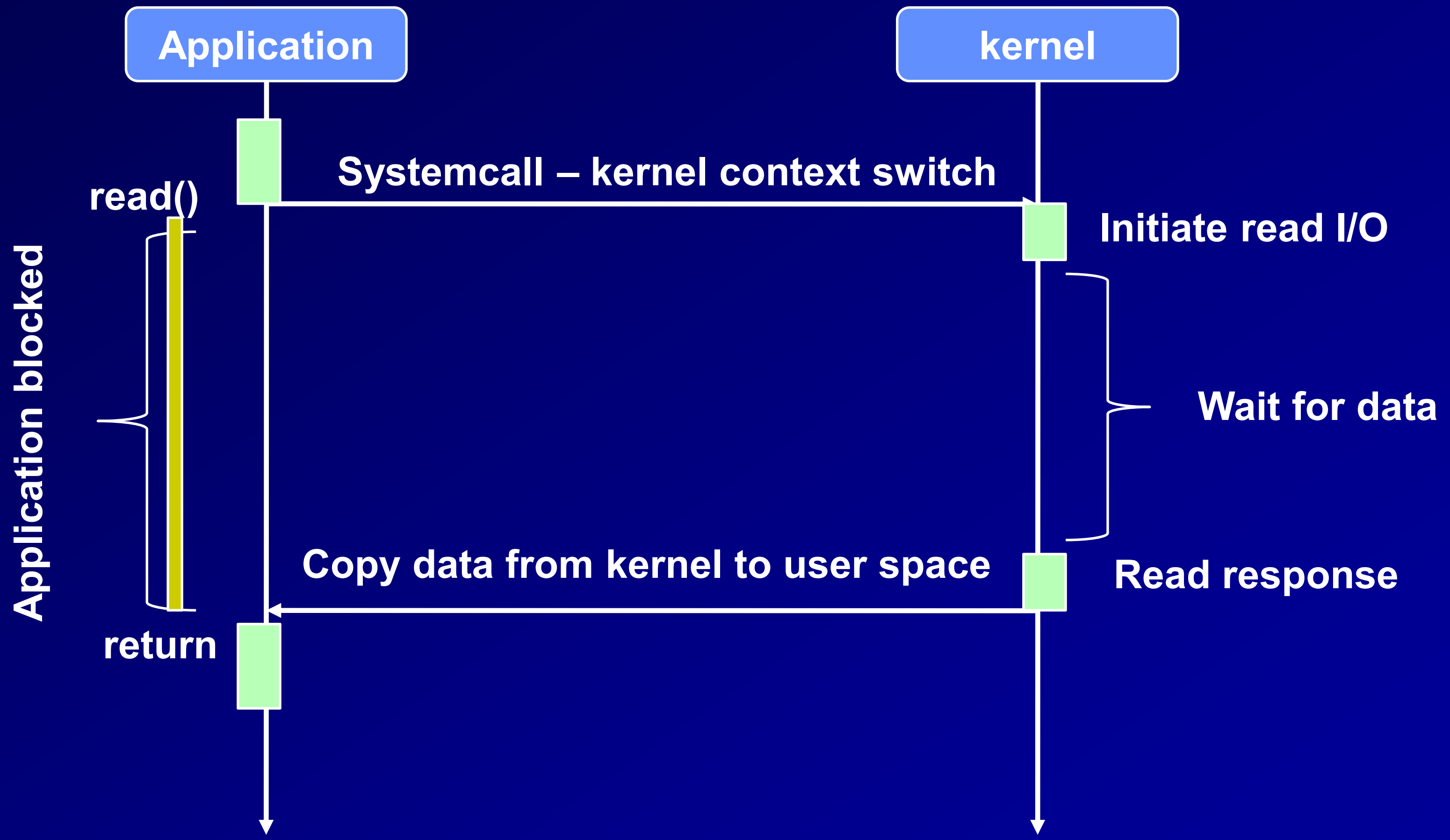
=> A problem in blocking I/O

## I/O Models

Blocking I/O
Nonblocking I/O
I/O multiplexing (select() and poll())
Signal driven I/O
Asynchronous I/O (Posix.1 aio_functions)

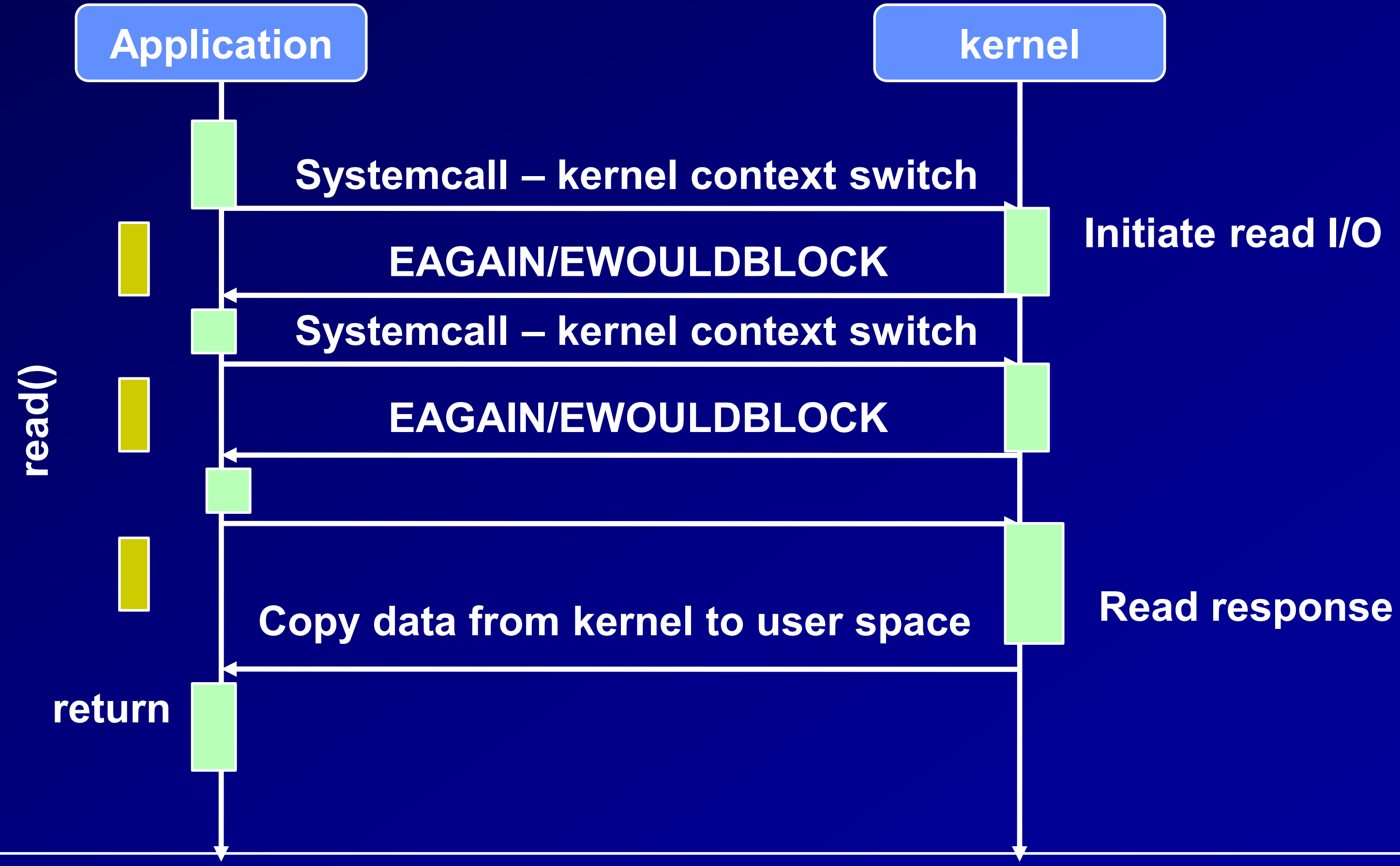
## Blocking I/O

A Procss blocks in a call to read().



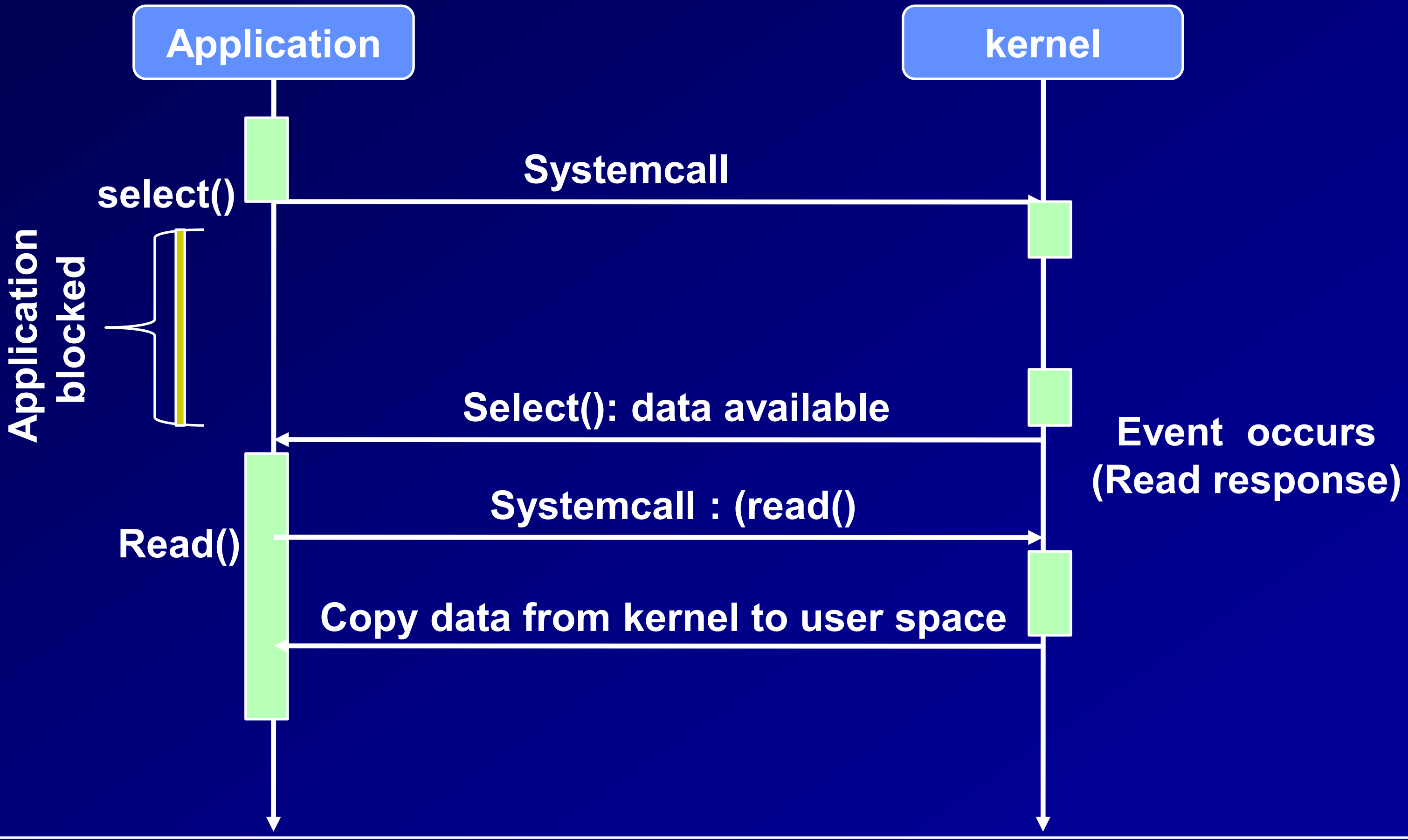
# Non-blocking I/O

A process repeatedly calls read() waiting for an OK return.


Application
kernel
Systemcall – kernel context switch
Initiate read I/O
EAGAIN/EWOULDBLOCK
Systemcall – kernel context switch
EAGAIN/EWOULDBLOCK
read()
Read response
Copy data from kernel to user space
return

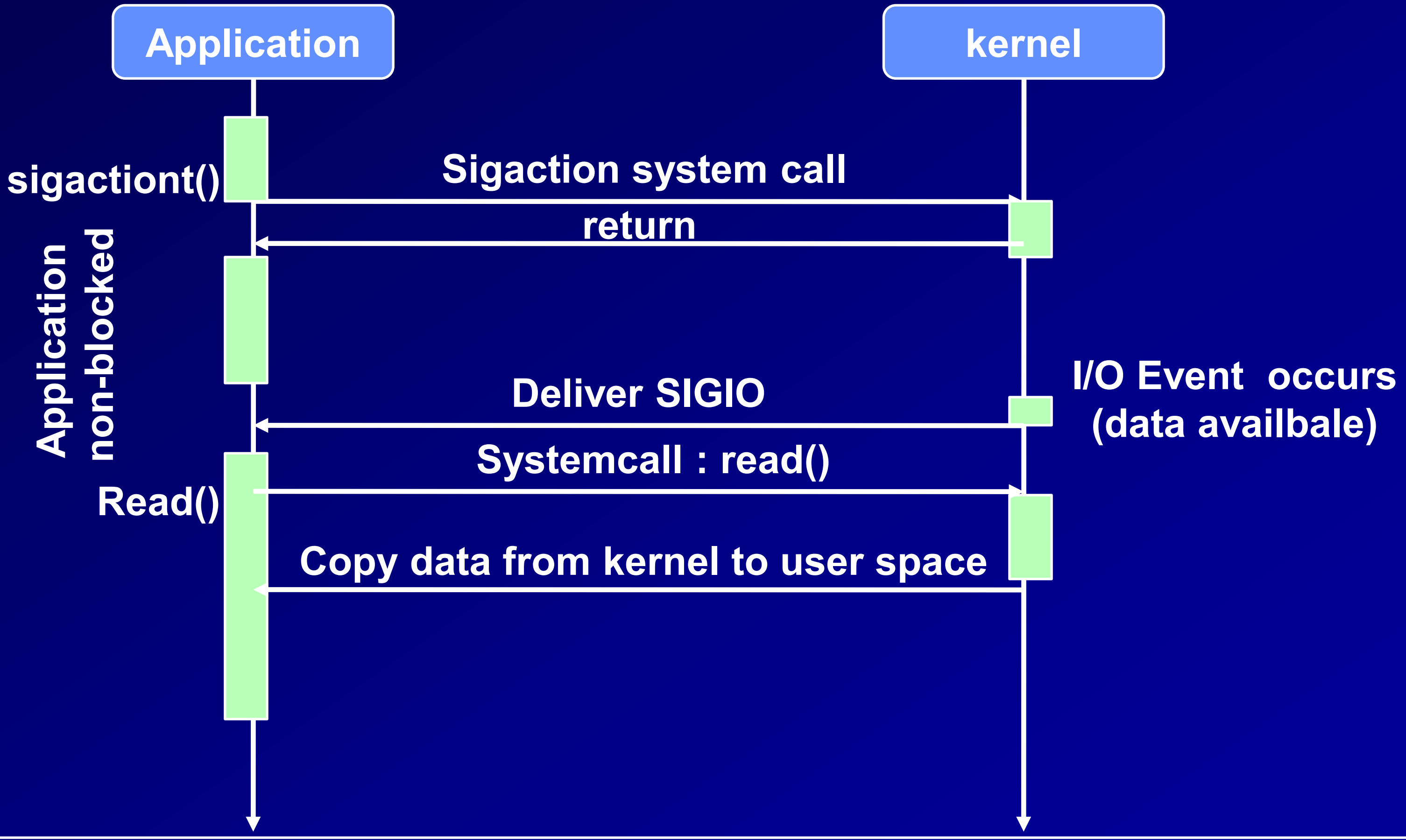
## I/O Multiplexing

A process waits for any one of multiple events to occur during a specified amount of time



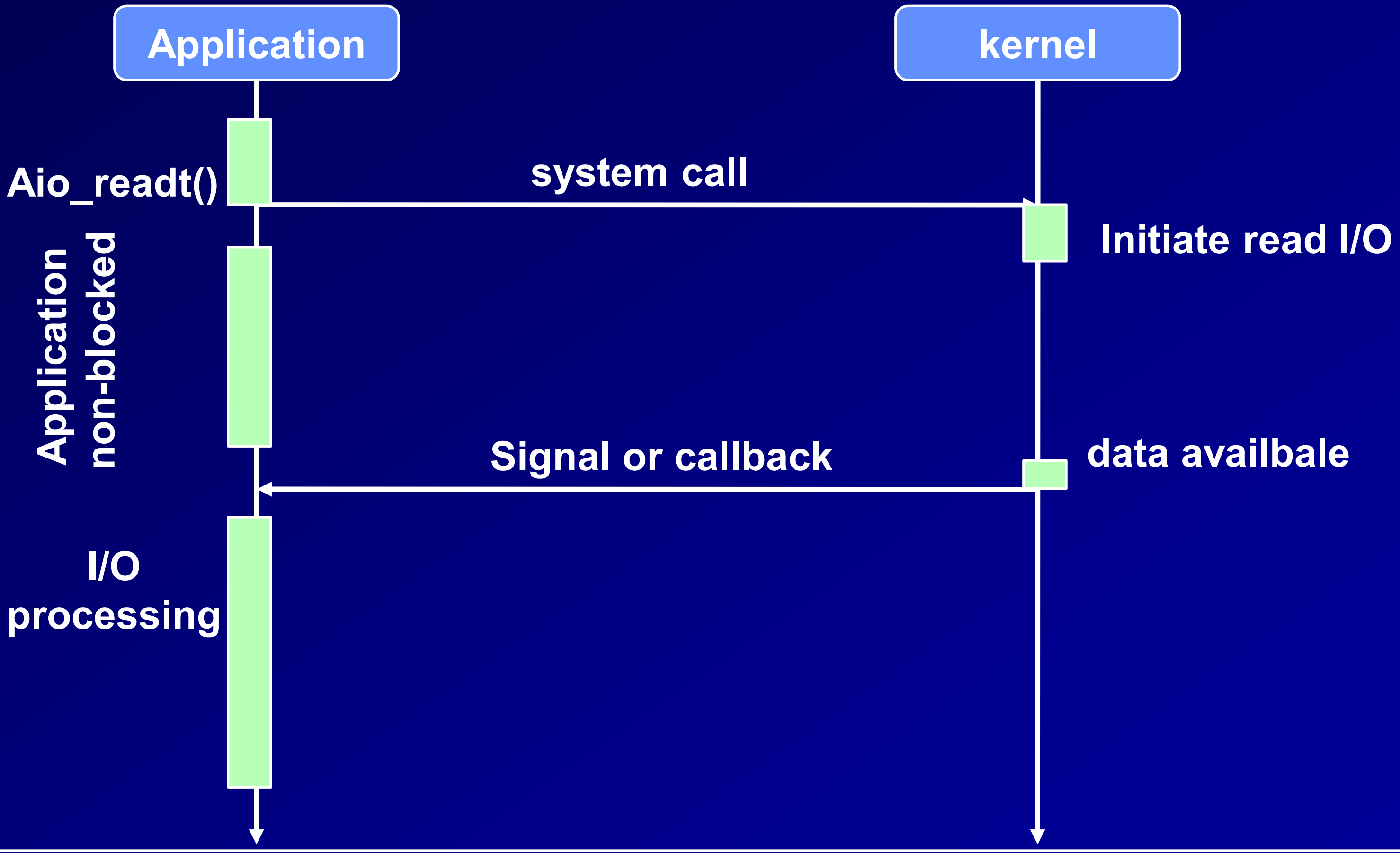
# Signal Driven I/O

## Use a signal handler for SIGIO



# Asynchronous I/O

## Use a signal handler for SIGIO



## Synchronous I/O & Asynchronous /O

Synchronous I/O : cause the requesting process to be blocked until that I/O operation (read()) completes.(blocking, nonblocking, I/O multiplexing)

Asynchronous I/O : does not cause the requesting process to be blocked

## Select function

### Select

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

 int select(int n, fd_set *readfds, fd_set *writefds,
                fd_set *exceptfds, struct timeval* timeout)
```

- waits for any one of multiple events to occur during a specified amount of time
- n: the highest-numbered socket descriptor plus one
  - FD_SETSIZE: the number of descriptor (1024)
- Readfds (writefds) : descriptor for checking readable (writable)
- exceptfds: descriptor for checking exception conditions  (ex.: OOB data)

## Select function

### Select (con't)

- Timeout : specify how long to check these sets.
  - If timeout = NULL, wait forever
  - If timeout=0, do not wait at all.
- Struct timeval

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

- Return value
  - The number of descriptors in the set on success
  - 0 if the timeout was reached
  - -1 on error (errno)

## Descriptor sets

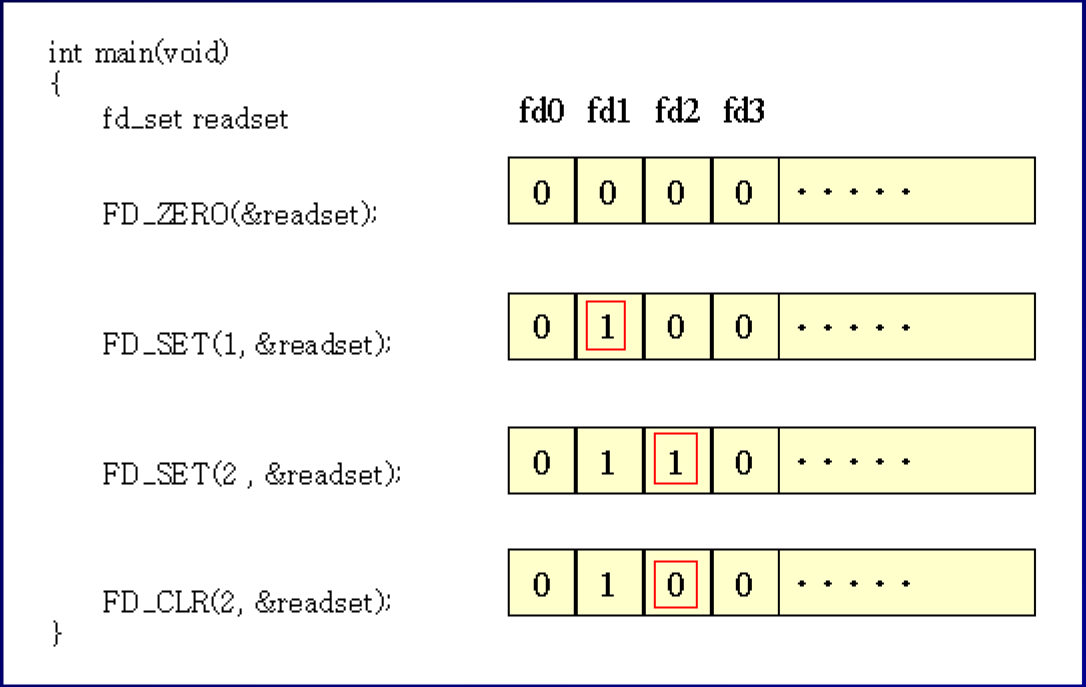
**fd_set : Array of integers**

**each bit in each integer corresponds to a descriptor.**
**The related macro functiuons**

| Macro fucntion | Function description |
|---|---|
| void FD_SET(int fd, fd_set *set) | Add fd to the set |
| void FD_CLR(int fd, fd_set *set) | Remove  fd to the set |
| Int  FD_ISSET(int fd, fd_set *set) | Return true if fd is in the st |
| void  FD_ZERO(fd_set *set) | Clear all entities from the set |

## Descriptor sets

### Example



```
int main(void)
{
    fd_set readset

    FD_ZERO(&readset);

    FD_SET(1, &readset);

    FD_SET(2 , &readset);

    FD_CLR(2, &readset);
}
```

| fd0 | fd1 | fd2 | fd3 | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | ······ |
| 0 | $\boxed{1}$ | 0 | 0 | ······ |
| 0 | 1 | $\boxed{1}$ | 0 | ······ |
| 0 | 1 | $\boxed{0}$ | 0 | ······ |

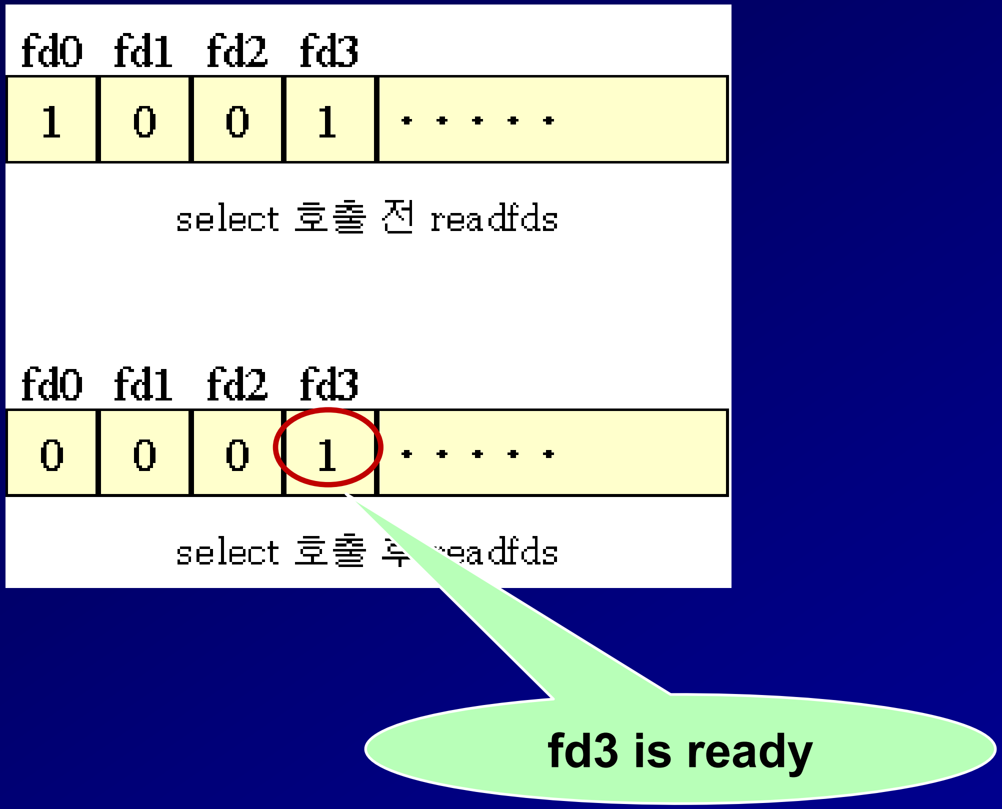
## Select function

### Conditions that cause a socket to be ready for select

| condition | Readable? | Writable? | Exception? |
|---|---|---|---|
| Data to read | o | | |
| New connection ready from listening socket | o | | |
| Read-half closed (FIN received) | o | | |
| Space avaliable for writing | | o | |
| Write-half closed | | o | |
| TCP out-of-band data | | | o |

## Select function

### Example : how to know which fds are ready?



## Select()

```
#define BUFSIZE 30

int main(int argc, char **argv)
{
  fd_set reads, temps;
  int result;

  char message[BUFSIZE];
  int str_len;
  struct timeval timeout;

  FD_ZERO(&reads);
  FD_SET(0, &reads); /* standard input  */

  while(1)
  {
    temps = reads;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;

    result = select(1, &temps, 0, 0, &timeout);
```

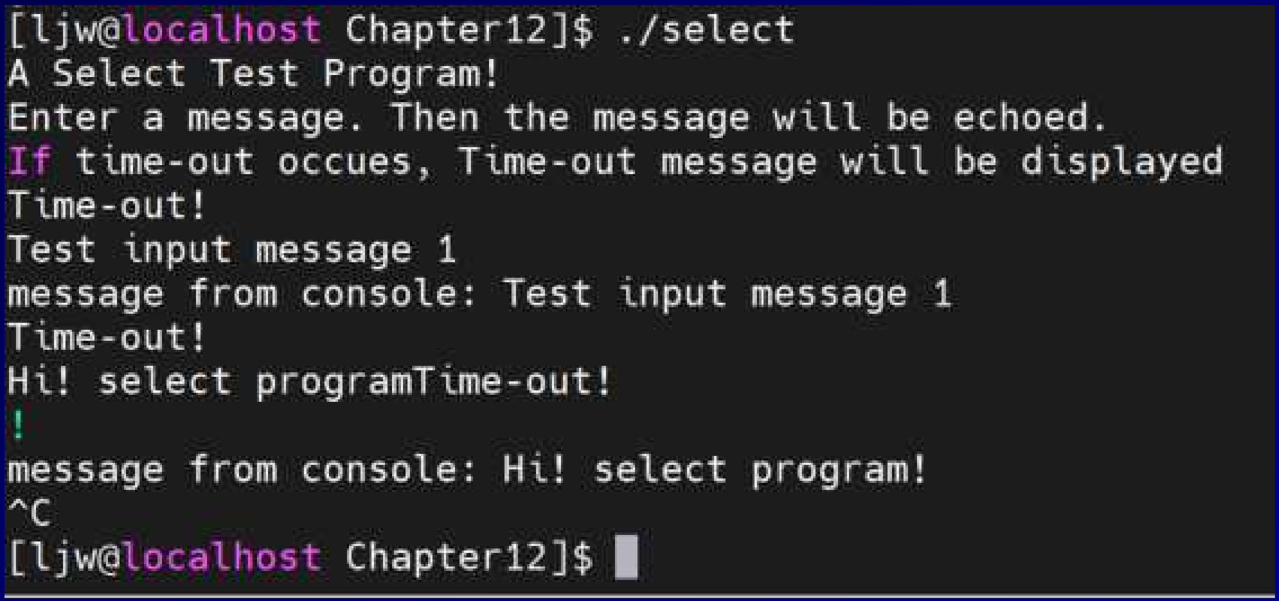
## Select()

### Example 1

```
 if (result == -1) {   /*errors in select */
     puts("select error!");
     exit(1);
   }
   else if (result == 0){   /* time-out */
     puts("time-out! : select ");
   }
   else {  /* change in fd */
       if(FD_ISSET(0, &temps)) {
           str_len = read(0, message, BUFSIZE);
           message[str_len] = '\0';
           fputs(message, stdout);
     }
   }
 } /* while(1) */
}
```

## Select()

### Example 1



```
[ljw@localhost Chapter12]$ ./select
A Select Test Program!
Enter a message. Then the message will be echoed.
If time-out occues, Time-out message will be displayed
Time-out!
Test input message 1
message from console: Test input message 1
Time-out!
Hi! select programTime-out!
!
message from console: Hi! select program!
^C
[ljw@localhost Chapter12]$ 
```

```
int main(int argc, char **argv)
{
  int serv_sock;
  struct sockaddr_in serv_addr;

  fd_set reads, temps;
  int fd_max;

  char message[BUFSIZE];
  int str_len;
  struct timeval timeout;

  if(argc!=2){
    printf("Usage : %s <port>\n", argv[0]);
    exit(1);
  }

  serv_sock = socket(PF_INET, SOCK_STREAM, 0);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  serv_addr.sin_port = htons(atoi(argv[1]));

  if(bind(serv_sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr)))
      error_handling("bind() error");
  if(listen(serv_sock, 5) == -1)
      error_handling("listen() error");
```

## Select()

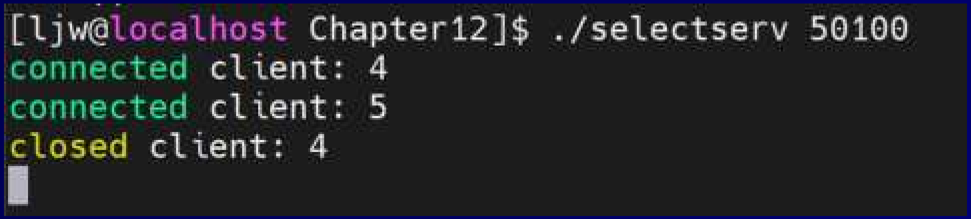
### Example 2

```
FD_ZERO(&reads);
FD_SET(serv_sock, &reads);
fd_max = serv_sock;

while(1)
{
    int fd, str_len;
    int clnt_sock, clnt_len;
    struct sockaddr_in clnt_addr;

    temps = reads;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;

    if (select(fd_max+1, &temps, 0, 0, &timeout) == -1)
        error_handling("select() error");
```

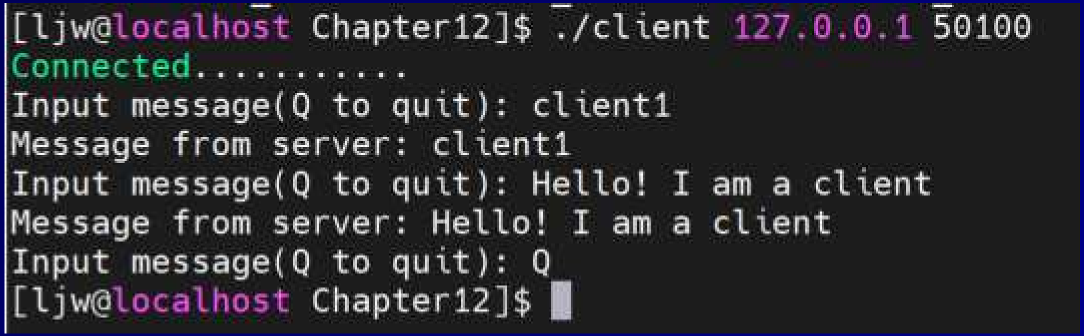
```
   for (fd = 0; fd < fd_max+1; fd++)
   {
      if (FD_ISSET(fd, &temps))
      {
            if (fd == serv_sock) { /* connect request from a client*/
               clnt_len = sizeof(clnt_addr);
               clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &clnt_len);
               FD_SET(clnt_sock, &reads);
               if (fd_max < clnt_sock)
                    fd_max=clnt_sock;
               printf("connected client : %d \n", clnt_sock);
            } else {
               str_len = read(fd, message, BUFSIZE);
               if(str_len == 0) { /* connection close */
                  FD_CLR(fd, &reads);
                  close(fd);
                  printf("closed client: %d \n", fd);
               } else {
                  write (fd, message, str_len);
               }
            }
      } //if(FD_ISSET(fd, &temps))
   } //for(fd=0; fd<fd_max+1; fd++)
  } //while(1)
}
```

## Select()

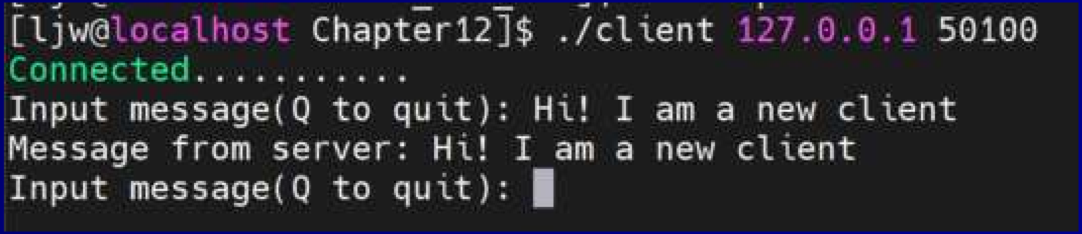
### Example 2



```
[ljw@localhost Chapter12]$ ./selectserv 50100
connected client: 4
connected client: 5
closed client: 4
```



```
[ljw@localhost Chapter12]$ ./client 127.0.0.1 50100
Connected..........
Input message(Q to quit): client1
Message from server: client1
Input message(Q to quit): Hello! I am a client
Message from server: Hello! I am a client
Input message(Q to quit): Q
[ljw@localhost Chapter12]$
```



```
[ljw@localhost Chapter12]$ ./client 127.0.0.1 50100
Connected..........
Input message(Q to quit): Hi! I am a new client
Message from server: Hi! I am a new client
Input message(Q to quit):
```