# LEC 07.
# ADVANCED I/O

Bui Trong Tung, SoICT, HUST



## Content

- Non-blocking I/O
- Signal-driven I/O
- Some advanced I/O functions

# NON-BLOCKING I/O

Bui Trong Tung, SoICT, HUST

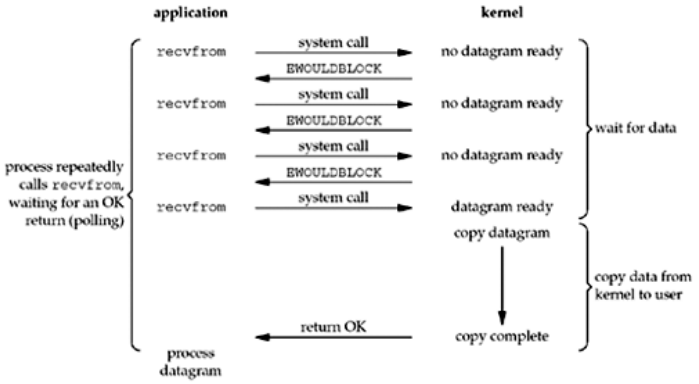


# Blocking I/O

- By default, sockets are blocking: when a socket call cannot be completed immediately, the process is put to sleep, waiting for the condition to be true
- Input functions: `recv()`, `recvfrom()`, etc
  - Blocks until some data arrives
- Output function: `send()`, `sendto()`, etc
  - TCP: blocks until there is free space in sending buffer
  - UDP: block on some systems due to the buffering and flow control
- Accepting incoming connections: `accept()`
  - Blocks until a new connection is available
- Initiating outgoing connections: `connect()`
  - Blocks until the client receives the ACK of its SYN

# Non-blocking I/O Model

- Non-blocking I/O model: I/O function returns immediately
- If there is no data to return, so the kernel immediately returns an error of EWOULDBLOCK instead






# Non-blocking I/O: use `fcntl()`

```
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /* int arg */ );
```

- Perform the file control operations described below on open files
- Parameter:
  - [IN]`fd`: the file descriptor
  - [IN]`cmd`: the control operation
  - The 3rd argument according to `cmd`
- Return:
  - Return -1 on error
  - Otherwise, return others depending on `cmd`

# Non-blocking I/O: use `fcntl()`

- Set non-blocking mode

```
int     flags;
/* Get the file status flags and file access modes */
if  ((flags = fcntl(fd, F_GETFL, 0)) < 0)
    perror("F_GETFL error");
/* Set a socket as nonblocking */
if (fcntl(fd, F_SETFL, flags | O_NONBLOCK) < 0)
    perror("F_SETFL error");
```

- Turn off non-blocking mode

```
int     flags;
/* Get the file status flags and file access modes */
if  ((flags = fcntl(fd, F_GETFL, 0)) < 0)
    perror("F_GETFL error");
/* Turn off non-blocking mode on socket */
if (fcntl(fd, F_SETFL, flags & ~O_NONBLOCK) < 0)
    perror("F_SETFL error");
```

7

# Non-blocking I/O: use `ioctl()`

```
#include <sys/ioctl.h>
int fcntl(int fd, int request, ... /* void arg */ );
```

- Manipulates the underlying device parameters of special files and control operating characteristics of files
- Parameters
  - [IN]`fd`: the file descriptor
  - [IN]`request`: device-dependent request code
  - The 3rd argument according to `request`
- Return:
  - 0 if succeed
  - -1 if error

```
int on = 1;
/* Set a socket as nonblocking */
ioctl(fd, FIONBIO, (char *)&on);
on = 0;
/* Turn off non-blocking mode on socket */
ioctl(fd, FIONBIO, (char *)&on);
```

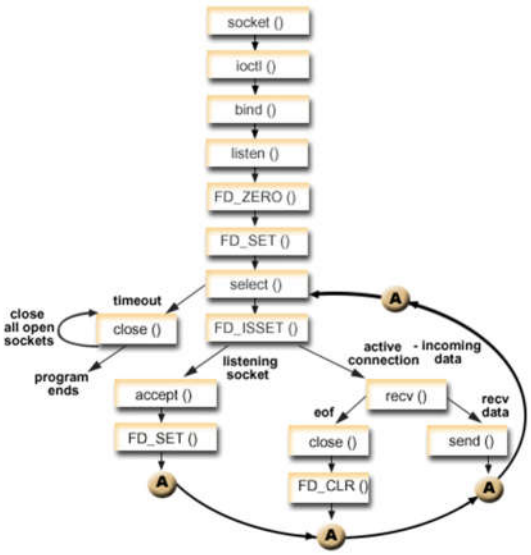
8

# Non-blocking I/O: process return value

```
//call I/O functions
if(ret < 0){
   //Error on I/O operation
   if (errno != EWOULDBLOCK)
}
else if(ret == 0){
   //Connection terminated normally
}
else if{
   //I/O operation is successful
}
```

9

# Non-blocking I/O:Example




10

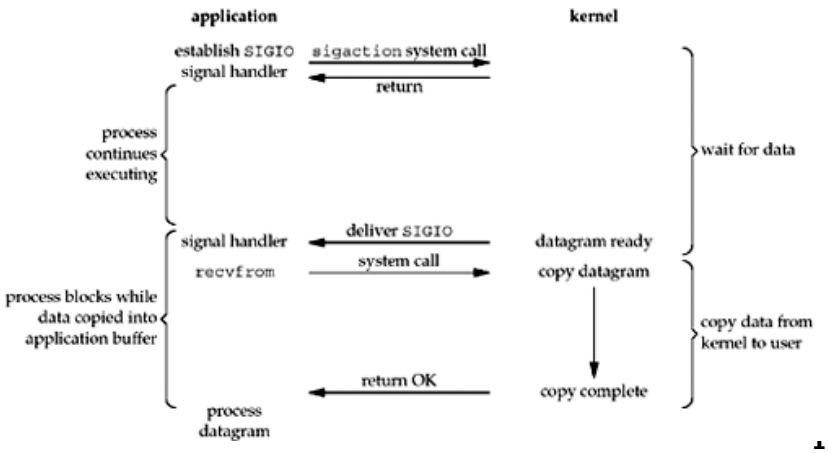
# SIGNAL-DRIVEN I/O

Bui Trong Tung, SoICT, HUST



## Signal-driven I/O

- Use signals, telling the kernel to notify app with the SIGIO signal when the descriptor is ready



# Signal-driven I/O: 3 steps

1. A signal handler must be established for the SIGIO signal.
2. Assign a process to receive the SIGIO signal
   `fcntl(fd, F_SETOWN, process_id)`
3. Enable signal-driven I/O on socket
   - Turn on asynchronous mode
   - Turn on non-blocking mode

- The importance is determining what conditions cause SIGIO to be generated for the socket owner.



# Signal-driven I/O: use `fcntl()`

- Enable signal-driven I/O on socket

```
int     flags;
/* Get the file status flags and file access modes */
if  ( (flags = fcntl (fd, F_GETFL, 0)) < 0)
    err_sys("F_GETFL error");
/* Set a socket as nonblocking */
if (fcntl(fd, F_SETFL, flags | O_ASYNC | O_NONBLOCK) < 0)
    err_sys("F_SETFL error");
```

- Turn off asynchronous I/O mode

```
int     flags;
/* Get the file status flags and file access modes */
if  ( (flags = fcntl (fd, F_GETFL, 0)) < 0)
    err_sys("F_GETFL error");
/* Turn off non-blocking mode on socket */
if (fcntl(fd, F_SETFL, flags & ~O_ASYNC & ~O_NONBLOCK) < 0)
    err_sys("F_SETFL error");
```

## Signal-driven I/O: use `ioctl()`

```
int on = 1;
/* Set a socket as nonblocking */
ioctl(fd, FIOASYNC, (char *)&on);
ioctl(fd, FIONBIO, (char *)&on)

on = 0;
/* Turn off non-blocking mode on socket */
ioctl(fd, FIOASYNC, (char *)&on);
ioctl(fd, FIONBIO, (char *)&on)
```

15

## SIGIO on sockets

- UDP socket: The signal SIGIO is generated whenever
  - A datagram arrives for the socket
  - An asynchronous error occurs on the socket
- TCP socket: the following conditions all cause SIGIO to be generated(very complex)
  - A connection request has completed on a listening socket
  - A disconnect request has been initiated
  - A disconnect request has completed
  - Half of a connection has been shut down
  - Data has arrived on a socket
  - Data has been sent from a socket (i.e., the output buffer has free space)
  - An asynchronous error occurred

16

## Example: signal-driven I/O on UDP socket

- See source code



# ADVANCED I/O FUNCTIONS

Bui Trong Tung, SoICT, HUST

# Socket Timeouts

- There are three ways to place a timeout on an I/O operation involving a socket:
  - Call alarm, which generates the SIGALRM signal when the specified time has expired
  - Block waiting for I/O in *select*
  - Use the newer SO_RCVTIMEO and SO_SNDTIMEO socket options
- Timeout on *connect* operation?

19

# `connect` with a timeout

```
#include <signal.h>
typedef void sigfunc(int)
static void connect_alarm(int);
int connect_timeo(int sockfd, const SA *saptr, socklen_t salen,
                                                    int nsec)
{
   sigfunc *sigfunc;
   int n;
   sigfunc = signal(SIGALRM, connect_alarm);
   if (alarm(nsec) != 0)
        err_msg("connect_timeo: alarm was already set");
   if ((n = connect(sockfd, saptr, salen)) < 0) {
        close(sockfd);
        if(errno == EINTR)
           errno = ETIMEDOUT;
   }
   alarm(0); // turn off the alarm
   signal(SIGALRM, sigfunc); // restore previous signal handler
   return (n);
}
static void connect_alarm(int signo) {return;}
```

20

## `readv()` and `writev()` Functions

```
#include <sys/uio.h>
ssize_t readv(int sockfd, const struct iovec *iov, int iovcnt);
ssize_t writev(int filedes, const struct iovec *iov, int iovcnt);
```

- Arguments:
  - *iov*: a pointer to an array of iovec structures
  - *iovcnt*: number of elements in iov array
- iov structure

```
struct iovec {
        void *iov_base;
        size_t iov_len;
};
```

21

## Example

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
   //...
   ssize_t bytes_read;
   int fd;
   char buf0[20];
   char buf1[30];
   char buf2[40];
   int iovcnt;
   struct iovec iov[3];
   iov[0].iov_base = buf0;
   iov[0].iov_len = sizeof(buf0);
   iov[1].iov_base = buf1;
   iov[1].iov_len = sizeof(buf1);
   iov[2].iov_base = buf2;
   iov[2].iov_len = sizeof(buf2);
   //...
   bytes_read = readv(fd, iov, 3);
   //...
```

22

# recvmsg() and sendmsg()

```
#include <sys/socket.h>
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
ssize_t sendmsg(int sockfd, struct msghdr *msg, int flags);
```

- Arguments:
  - *msg*: pointer to msghdr structures

```
struct msghdr {
    void *msg_name;              /* protocol address */
    socklen_t msg_namelen;       /* size of protocol address */
    struct iovec *msg_iov;       /* scatter/gather array */
    int msg_iovlen;              /* # elements in msg_iov */
    void *msg_control;           /* ancillary data (cmsghdr struct) */
    socklen_t msg_controllen;    /* length of ancillary data */
    int msg_flags;               /* flags returned by recvmsg() */
};
```

23

# Example

```
#include <sys/socket.h>
   struct sockaddr_in dest;
   int rc;
   struct iovec iov[3];
   struct msghdr mh;
   memset(&dest,'\0',sizeof(dest)); dest.sin_family = AF_INET;
   memcpy(&dest.sin_addr,host->h_addr,sizeof(dest.sin_addr));
   dest.sin_port = htons(TRANSACTION_SERVER);
   iov[0] .iov_base = (caddr_t)head; iov[0] .iov_len = sizeof(struct header);
   iov[1] .iov_base = (caddr_t)trans; iov[1] .iov_len = sizeof(struct record);
   iov[2] .iov_base = (caddr_t)trail; iov[2] .iov_len = sizeof(struct trailer);
   mh.msg_name = (caddr_t) &dest; mh.msg_namelen = sizeof(dest);
   mh.msg_iov = iov; mh.msg_iovlen = 3;
   mh.msg_ msg_control = NULL;
   mh.msg_ msg_controllen = 0;
   rc = sendmsg(s, &mh, 0); /* no flags used */
```

24