CSCI 330
The UNIX System

Socket Programming Detail

Unit Overview

- TCP programming
- socket behavior
  - blocking vs. non-blocking
- signal handler

Socket system calls

server

client

Primitive	Meaning
socket	Create a new communication endpoint
bind	Attach a local address to a socket
listen	Announce willingness to accept connections
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
write	Send(write) some data over the connection
read	Receive(read) some data over the connection
close	Release the connection

TCP socket programming

- sockets can operate in one of two modes:

  <u>blocking or non-blocking</u>

- default: blocking mode
  - calls may block, i.e. wait for something before they return
  - blocking is a special state in which a process is waiting for I/O
  - process is removed from the scheduler queue
  - when the I/O completes, the process is woken up
- non-blocking mode
  - calls returns immediately, producing full, partial or no result

Blocking mode: calls that can block

- accept
  - blocks until a connection is present
- connect
  - blocks until connection is established
- read
  - blocks if no data is available
- write
  - blocks when data does not fit into send buffer of the socket

Illustration: blocking mode

```
// loop to write a single character
char c = 'x';
while (true) {
   if (write(sock, &c, 1) < 1) {
      perror("write");
      exit(EXIT_FAILURE);
   }
}
```

Illustration: dummy server

dummyTCPServer.cxx - /home/student/Desktop/Unit 20 Programs - Geany

File Edit Search View Document Project Build Tools Help

dummyTCPServer.cxx

```
60
61      // Run until cancelled
62      while (true) {
63          connSock=accept(sock, NULL, NULL);
64          if (connSock < 0) {
65              perror("accept failed");
66              exit(EXIT_FAILURE);
67          }
68          // fork
69          if (fork()) {   // parent process
70              close(connSock);
71          } else {            // child process
72              cerr << "Waiting here: ";
73              cin >> buffer;
74
75              close(connSock);
76              exit(EXIT_SUCCESS);
77          }
78      }
79
80      close(sock);
81      return 0;
```

line: 73 / 83 col: 0 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: main

Illustration: blocking mode

writeBlock.cxx - /home/student/Desktop/Unit 20 Programs - Geany

File Edit Search View Document Project Build Tools Help

writeBlock.cxx

```
36
37      // Construct the server sockaddr_in structure
38      memset(&echoserver, 0, sizeof(echoserver));       /* Clear struct */
39      echoserver.sin_family = AF_INET;                  /* Internet/IP */
40      echoserver.sin_addr.s_addr = inet_addr(argv[1]);  /* IP address */
41      echoserver.sin_port = htons(atoi(argv[2]));       /* server port */
42
43      // connect to server
44      if (connect(sock, (struct sockaddr *) &echoserver, sizeof(echoserver)) < 0) {
45          perror("cannot connect");
46          exit(EXIT_FAILURE);
47      }
48
49      // loop to write a single character
50      char c = 'x';
51      for (count=1; ; count++) {
52
53          if (write(sock, &c, 1) < 1) {
54              perror("Mismatch in number of bytes sent");
55              exit(EXIT_FAILURE);
56          }
57
58          if (!(count % 1024))
59              cerr << count << " characters sent\n";
60      }
61      return 0;
```

line: 1 / 63 col: 0 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: unknown

What to do when call blocks ?

- Answer: <u>wait</u>

  or: <u>set alarm</u>

  - i.e. schedule signal to sent in future

- how:
  - set alarm timeout
  - program and install signal handler

System call: alarm

Terminal

File Edit View Search Terminal Help

```
ALARM(2)                    Linux Programmer's Manual                    ALARM(2)

NAME
       alarm - set an alarm clock for delivery of a signal

SYNOPSIS
       #include <unistd.h>

       unsigned int alarm(unsigned int seconds);

DESCRIPTION
       alarm()  arranges  for a SIGALRM signal to be delivered to the calling
       process in seconds seconds.

       If seconds is zero, no new alarm() is scheduled.

       In any event any previously set alarm() is canceled.

RETURN VALUE
       alarm() returns the number of seconds remaining until  any  previously
       scheduled  alarm was due to be delivered, or zero if there was no pre-
       viously scheduled alarm.

 Manual page alarm(2) line 1 (press h for help or q to quit)
```

System call: alarm

```
unsigned int alarm(unsigned int seconds)
```

- arranges for a SIGALRM signal to be delivered to the calling process in **`seconds`** seconds
  - if seconds is zero, no new alarm() is scheduled
  - any previously set alarm() is canceled
- returns the number of seconds remaining until any previously scheduled alarm was due to be delivered
- zero if there was no previously scheduled alarm

Illustration: blocking mode

writeBlockAlarm.cxx - /home/student/Desktop/Unit 20 Programs - Geany

File Edit Search View Document Project Build Tools Help

writeBlockAlarm.cxx

```
37      // Construct the server sockaddr_in structure
38      memset(&echoserver, 0, sizeof(echoserver));       /* Clear struct */
39      echoserver.sin_family = AF_INET;                  /* Internet/IP */
40      echoserver.sin_addr.s_addr = inet_addr(argv[1]);  /* IP address */
41      echoserver.sin_port = htons(atoi(argv[2]));       /* server port */
42
43      // connect to server
44      if (connect(sock, (struct sockaddr *) &echoserver, sizeof(echoserver)) < 0) {
45          perror("cannot connect");
46          exit(EXIT_FAILURE);
47      }
48
49      // loop to write a single character
50      char c = 'x';
51      for (count=1; ; count++) {
52          alarm(5);
53          if (write(sock, &c, 1) < 1) {
54              perror("Mismatch in number of bytes sent");
55              exit(EXIT_FAILURE);
56          }
57          alarm(0);
58          if (!(count % 1024))
59              cerr << count << " characters sent\n";
60      }
61      return 0;
62  }
```

line: 61 / 63 col: 13 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: main

System call: signal

Terminal

File Edit View Search Terminal Help

```
SIGNAL(2)                    Linux Programmer's Manual                    SIGNAL(2)

NAME
       signal - ANSI C signal handling

SYNOPSIS
       #include <signal.h>

       typedef void (*sighandler_t)(int);

       sighandler_t signal(int signum, sighandler_t handler);

DESCRIPTION
       The  behavior  of  signal()  varies across UNIX versions, and has also
       varied historically across different versions  of  Linux.   Avoid  its
       use: use sigaction(2) instead.  See Portability below.

       signal()  sets  the disposition of the signal signum to handler, which
       is either SIG_IGN, SIG_DFL, or the  address  of  a  programmer-defined
       function (a "signal handler").

       If the signal signum is delivered to the process, then one of the fol-
       lowing happens:
 Manual page signal(2) line 1 (press h for help or q to quit)
```

System call: signal

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler)
```

- sets the disposition of the signal **`signum`** to **`handler`**
- **`handler`** can be:
  - **`SIG_IGN`** to ignore signal
  - **`SIG_DFL`** to restore default
  - address of a programmer-defined function
- returns the previous value of the signal handler

Signal handler: detail

```
void alarm_action(int s) {
    cerr << "write blocked after " << count << " characters\n";
    exit(EXIT_SUCCESS);
}

// set signal handler on ALARM
signal(SIGALRM, alarm_action);
```

Illustration: blocking mode w/signal

writeBlockSignal.cxx - /home/student/Desktop/Unit 20 Programs - Geany

File Edit Search View Document Project Build Tools Help

writeBlockSignal.cxx

```
46      echoserver.sin_addr.s_addr = inet_addr(argv[1]);   /* IP address */
47      echoserver.sin_port = htons(atoi(argv[2]));        /* server port */
48
49      // connect to server
50      if (connect(sock, (struct sockaddr *) &echoserver, sizeof(echoserver)) < 0) {
51          perror("cannot connect");
52          exit(EXIT_FAILURE);
53      }
54
55      // set signal handler on ALARM
56      signal(SIGALRM, alarm_action);
57
58      // loop to write a single character
59      char c = 'x';
60      for (count=1; ; count++) {
61          alarm(5);
62          if (write(sock, &c, 1) < 1) {
63              perror("Mismatch in number of bytes sent");
64              exit(EXIT_FAILURE);
65          }
66          alarm(0);
67          if (!(count % 1024))
68              cerr << count << " characters sent\n";
69      }
70      return 0;
71  }
```

line: 56 / 72 col: 0 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: main

TCP socket: non-blocking mode

- calls returns immediately, producing full, partial or no result

- socket can be in blocking or non-blocking mode
- determined by flag associated with socket descriptor

- flag can be set via `fcntl` system call

Non-Blocking mode: call behavior

- accept
  - when there are connections to accept, accept behaves as usual
  - if there are no pending connections, returns −1 with errno set to EWOULDBLOCK
- connect
  - when connection to a listening server can be established, connect behaves as usual
  - If connection cannot be established, connect returns −1 with errno set to EINPROGRESS

Non-Blocking mode: call behavior

- read
  - when there is data to read, read behaves as usual
  - if there is no data, read returns −1 with errno EWOULDBLOCK
- write
  - when buffer space is available, write behaves as usual
  - if buffer becomes full, write returns number of bytes written
  - if buffer is full, write returns -1 with errno EWOULDBLOCK

System call: fcntl

Terminal

File Edit View Search Terminal Help

```
FCNTL(2)                    Linux Programmer's Manual                    FCNTL(2)

NAME
       fcntl - manipulate file descriptor

SYNOPSIS
       #include <unistd.h>
       #include <fcntl.h>

       int fcntl(int fd, int cmd, ... /* arg */ );

DESCRIPTION
       fcntl()  performs  one  of  the operations described below on the open
       file descriptor fd.  The operation is determined by cmd.

       fcntl() can take an optional third  argument.  Whether  or  not  this
       argument is required is determined by cmd.  The required argument type
       is indicated in parentheses after each cmd name (in  most  cases,  the
       required  type  is  long,  and we identify the argument using the name
       arg), or void is specified if the argument is not required.

   Duplicating a file descriptor
       F_DUPFD (long)
 Manual page fcntl(2) line 1 (press h for help or q to quit)
```

System call: fcntl

```
int fcntl(int fd, int cmd, ... /* arg */ )
```

- performs operation **`cmd`** on the open descriptor **`fd`**
- **`cmd`** to set non-blocking mode for socket:

  `F_SETFL, O_NONBLOCK`

- returns -1 on error

Illustration: non-blocking write

writeNonBlock.cxx - /home/student/Desktop/Unit 20 Programs - Geany

File Edit Search View Document Project Build Tools Help

writeNonBlock.cxx

```
51      // put socket into non-blocking mode
52      if (fcntl(sock, F_SETFL, O_NONBLOCK) < 0) {
53          perror("fcntl");
54          exit(EXIT_FAILURE);
55      }
56
57      // loop to write a single character
58      char c = 'x';
59      for (count=1; ; count++) {
60
61          if (write(sock, &c, 1) < 1) {
62              if (errno == EWOULDBLOCK) {
63                  cerr << "buffer full after: " << count << " characters sent\n";
64                  break;
65              } else {
66                  perror("write");
67                  exit(EXIT_FAILURE);
68              }
69          }
70          if (!(count % 1024))
71              cerr << count << " characters sent\n";
72      }
73      return 0;
```

line: 52 / 75 col: 0 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: main

Summary

- TCP programming
- socket behavior
  - blocking vs. non-blocking
- signal handler