Fall 2004 Handout 3

Server and client

In a conversation between two programs, the one that sends the first packet is the *client*; the other one is the *server*.

The *server* starts running first, and waits for a client to talk to it. In fact, the server probably runs 24 hours per day, and continues to run after the conversation is over. Moreover, a server probably talks to many clients simultaneously.

Either the server or the client can terminate the conversation. In the following examples, the **daytime** server at port 13 and the **finger** server at port 79 terminate the conversation. On the other hand, the client who talks to the **echo** server at port 7 decides when it's time to terminate.

/etc/services

See pp. 47−48; **services**(4); and

http://www.iana.org/assignments/port-numbers

sed outputs a copy of the **/etc/services** file with the comments stripped away. Inside the argument of **awk**, a regular expression must be surrounded by slashes. If the regular expresion contains a slash, it must therefore be preceded by a backslash.

The following caret ^ means ''start of the second field'', not ''start of the entire line''.

```
1$ sed 's/#.*//' /etc/services | awk '$2 ~ /^(7|13|21|23|53|79|520)\//' | more
echo    7/tcp
echo    7/udp
daytime 13/tcp
daytime 13/udp
ftp     21/tcp
telnet  23/tcp
domain  53/udp
domain  53/tcp
finger  79/tcp
route   520/udp router routed
```

What telnet really does

telnet is a client that lets you say whatever you want to any server that speaks TCP. When your **telnet** terminates the conversation, it says **Connection closed**. When the server terminates the conversation, your **telnet** says (in iambic tetrameter) **Connection closed by foreign host**.

If you get no response from **spider.let.uu.nl**, try **smail.let.uu.nl** (131.211.194.45).

```
1$ telnet spider.let.uu.nl 7                    What you type is in italics.
Trying 131.211.194.47...
Connected to spider.let.uu.nl.
Escape character is '^]'.
"And this also," said Marlow suddenly,
"And this also," said Marlow suddenly,
"has been one of the dark places of the earth."
"has been one of the dark places of the earth."
control-]
telnet> help
telnet> quit
Connection closed.
2$
```

The following server (**daytime**) requires no input from the client. When the server terminates your conversation, **telnet** says **Connection closed by foreign host**:

```
3$ telnet www.urz.uni-heidelberg.de 13
Trying 129.206.218.89...
Connected to www.urz.uni-heidelberg.de.
Escape character is '^]'.
Fri Jan  9 06:00:57 2004
Connection closed by foreign host.
4$

5$ date
Fri Jan  9 00:00:57 EST 2004
```

The following server requires a newline, optionally preceded by a loginname:

```
6$ telnet www.urz.uni-heidelberg.de 79
Trying 129.206.218.89...
Connected to www.urz.uni-heidelberg.de.
Escape character is '^]'.
RETURN
  User     Real Name          What    Idle  TTY  Host      Console Location
aaghajan Anoush Aghajani-Tl pine      0:02   *5 aixterm5   (128.84.46.130)
abender  Andreas Bender              0:05 dtre aixterm4   (xog02.urz.uni-he)
ahillers Annette Hillers             0:05 dtre aixterm4   (xterm02.psi.uni-)
etc.
Connection closed by foreign host.
7$

8$ telnet www.urz.uni-heidelberg.de 79
Trying 129.206.218.89...
Connected to www.urz.uni-heidelberg.de.
Escape character is '^]'.
aaghajanRETURN
  User     Real Name          What    Idle  TTY  Host      Console Location
aaghajan Anoush Aghajani-Tl pine      0:05   *5 aixterm5   (128.84.46.130)
Connection closed by foreign host.
9$
```

Sockets: Curry p. 391

Each computer that is connected to the Internet is called a *host*. A *socket* is like a pipe, except that (1) the same socket can be used for both input and output, and (2) the two processes can run on different hosts. The process that runs on the host where you're logged in is called the *local* process; the one on the other host is the *remote* process.

The *address family* of a socket tells how far apart the two processes are allowed to be. A socket whose address family is **AF_UNIX** (Curry p. 367) can be used only for communication between two processes that are running on the same Unix machine. An **AF_UNIX** socket is given a name and is stored in a directory, so you can see it with **ls -l**. It's similar to a named pipe:

```
1$ cd /tmp
2$ ls -l | grep '^s' | more
srwxrwxrwx   1 mysqlu   mysqlg         0 Nov  7 21:29 mysql.sock
srwxr-xr-x   1 root     root           0 Nov  7 21:29 psb_back_socket
srwxr-xr-x   1 root     root           0 Nov  7 21:29 psb_front_socket
```

A socket whose address family is **AF_INET** (Curry p. 399) can be used for communication between two processes that are running on different hosts. An **AF_INET** socket is given a *port number* instead of a name and directory.

Communicate via a socket: Bach pp. 383−388; Curry pp. 398−399, 401−404, 405−407

Every host on the Internet has an *IP address*; Curry calls it an *host address* on pp. 393−394. For example, the IP address of the host **smail.let.uu.nl** (Universiteit Utrecht in the Netherlands) is

```
1$ /usr/sbin/nslookup smail.let.uu.nl
Server:  CMCL2.NYU.EDU
Address:  128.122.253.92
```

written with three dots separating the four octets.

There is a server named **echo** at port 7 on the host **smail.let.uu.nl** that is waiting for you to send it data. It will send a copy of the data back to you.

```
 1 /* Excerpts from /usr/include/sys/socket.h. */
 2
 3 /* Address family: the first argument of socket, and the sin_family field
 4 of struct sockaddr_in. */
 5 #define AF_UNIX 1             /* two processes on the same Unix machine */
 6 #define AF_INET 2             /* two processes on different hosts, IPv4 */
 7 #define AF_INET6 26           /* two process on different hosts, IPv6 */
 8
 9 /* Socket type: the second argument of the socket function. */
10 #define SOCK_STREAM 1         /* stream socket (TCP) */
11 #define SOCK_DGRAM 2          /* datagram socket (UDP) */
12
13 /* The second argument of the shutdown function. */
14 #define SHUT_RD   0           /* Disables further receive operations */
15 #define SHUT_WR   1           /* Disables further send operations */
16 #define SHUT_RDWR 2           /* Disables further send and receive operations
17
18 /* Maximum queue length specifiable by listen. */
19 #define SOMAXCONN 1024

20 /* Excerpts from /usr/include/sys/types.h. */
21 typedef unsigned short u_short;
```

```
22 typedef unsigned int u_int;

23 /* Excerpts from /usr/include/netinet/in.h. */
24
25 /* Only the superuser can bind to a port whose number is smaller than this.
26 Curry pp. 412-413. */
27 #define IPPORT_RESERVED 1024
28
29 /* Ports > IPPORT_USERRESERVED are reserved for servers, not necessarily
30 run by the superuser. */
31 #define IPPORT_USERRESERVED 5000
32
33 /* Tell server to accept a connection on any interface, Curry pp. 400, 405. */
34 #define INADDR_ANY 0x00000000
35
36 typedef uint32_t in_addr_t;
37
38 struct in_addr {
39     union {
40         struct { uint8_t s_b1, s_b2, s_b3, s_b4; } _S_un_b;
41         struct { uint16_t s_w1, s_w2; } _S_un_w;
42         in_addr_t _S_addr;
43     } _S_un;
44 };
45
46 #define  s_addr  _S_un._S_addr
47
48 struct sockaddr_in {          /* Internet socket address */
49     u_short sin_family;       /* address family: AF_UNIX or AF_INET */
50     u_short sin_port;         /* port number from /etc/services */
51     struct in_addr sin_addr;  /* 32-bit IP address from nslookup */
52     char sin_zero[8];         /* padding */
53 };
```

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/socket.c**

```
 1 /* Write one character to the program at port 7 (echo) of smail.let.uu.nl
 2 Then read one character back. */
 3
 4 #include <stdio.h>
 5 #include <stdlib.h>
 6 #include <sys/types.h>
 7 #include <sys/socket.h>
 8
 9 int main(int argc, char **argv)
10 {
11     const int s = socket(AF_INET, SOCK_STREAM, 0); /* file descriptor */
12     struct sockaddr_in echo;
13     char buffer[INET6_ADDRSTRLEN];          /* for inet_ntop */
14     int i;                                  /* return value of inet_pton */
15     char c = 'A';
16
17     if (s < 0) {
18         perror(argv[0]);
19         return 1;
```

```
20      }
21
22      bzero((char *)&echo, sizeof echo);   /* fill the echo structure with zeroes */
23      echo.sin_family = AF_INET;           /* Internet address family: IPv4 */
24      echo.sin_port = htons(7);            /* echo is at port number 7 */
25
26      /* IP address of smail.let.uu.nl: */
27      i = inet_pton(AF_INET, "131.211.194.40", &echo.sin_addr);
28      if (i == 0) {
29          fprintf(stderr, "%s: bad dotted string to IP address\n", argv[0]);
30          return 2;
31      }
32
33      if (i != 1) {
34          perror(argv[0]);   /* bad address family */
35          return 3;
36      }
37
38      if (inet_ntop(AF_INET, &echo.sin_addr, buffer, sizeof buffer) == NULL) {
39          perror(argv[0]);
40          return 4;
41      }
42      printf("Trying %s...\n", buffer);
43
44      if (connect(s, (struct sockaddr *)&echo, sizeof echo) != 0) {
45          perror(argv[0]);
46          return 5;
47      }
48
49      printf("Connected to smail.let.uu.nl.\n");
50
51      if (write(s, &c, 1) != 1) {
52          perror(argv[0]);
53          return 6;
54      }
55
56      c = '\0';   /* Prove that the same character comes back from the server. */
57
58      if (read(s, &c, 1) != 1) {
59          perror(argv[0]);
60          return 7;
61      }
62
63      printf("%c\n", c);
64
65      if (shutdown(s, SHUT_RDWR) != 0) {    /* Curry pp. 128-409 */
66          perror(argv[0]);
67          return 8;
68      }
69
70      printf("Connection closed.\n");
71      return EXIT_SUCCESS;
72  }
```

```
2$ gcc -o ~/bin/socket socket.c -lsocket -lnsl "Networking Services Library"
3$ ls -l ~/bin/socket
4$ socket
Trying 131.211.194.40...
Connected to smail.let.uu.nl.
A
Connection closed.
5$
```

Network byte order: Curry pp. 397–398; `htons`(3), `htonl`(3)

Use use *type punning* to print the contents of **`s`** byte-by-byte:

—Source code on the Web at **`http://i5.nyu.edu/~mm64/x52.9544/src/byteorder.c`**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <sys/types.h>
 4 #include <netinet/in.h>
 5
 6 int main(int argc, char **argv)
 7 {
 8     unsigned short s = 0x1234;   /* decimal 4660, binary 00010010 00110100 */
 9     char *p = (char *)&s;        /* Let the value of p be the address of s. */
10
11     printf("The value of s is hexadecimal %04X.\n", s);
12     printf("The address of s is hexadecimal %08X.\n", p);
13     printf("The number of bytes in s is %d.\n\n", sizeof s);
14
15     printf("The contents of address %08X is hexadecimal %02X.\n", p, p[0]);
16     printf("The contents of address %08X is hexadecimal %02X.\n\n", p + 1, p[1]);
17
18     printf("htons returns %04X.\n", htons(s));    /* "host to network short" */
19     return EXIT_SUCCESS;
20 }
```

You could split line 9 to

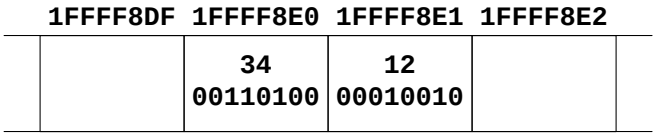
```
 9     char *p;
10     p = (char *)&s;      /* Let the value of p be the address of s. */
```

—but why would you want to?

```
1$ gcc -o ~/bin/byteorder byteorder.c
2$ ls -l ~/bin/byteorder
3$ byteorder
The value of s is hexadecimal 1234.
The address of s is hexadecimal 1FFFF8E0.
The number of bytes in s is 2.

The contents of address 1FFFF8E0 is hexadecimal 34.
The contents of address 1FFFF8E1 is hexadecimal 12.

htons returns 3412.
```

▼ Homework 3.1: find the byte order on your machine at work

What is the **sizeof** and byte order on your machine of these three variables:

```
 1     short s = 0x1234;
 2     int i;
 3     long l = 0x12345678;
 4
 5     if (sizeof(int) == 2) {
 6         i = 0x1234;
 7     } else if (sizeof(int) == 4) {
 8         i = 0x12345678;
 9     } else {
10         fprintf(stderr, "sizeof(int) == %d\n", sizeof(int));
11     }
```

▲

socket in Perl: pp. 217, 348−355, 498−500 in O'Reilly Perl book

We saw **inet_aton**, **sockaddr_in**, and **connect** in X52.9547 Handout 1, p. 25.

http://i5.nyu.edu/~mm64/x52.9544/src/socket.pl

```
#!/bin/perl
#Output one line to the echo server at port 7 of smail.let.uu.nl
#(131.211.194.40).  Then input the line back from the server.

use Socket;       #for AF_INET, etc.
use FileHandle;   #for autoflush

socket(S, AF_INET, SOCK_STREAM, 0) or die "$0: $!";
S->autoflush();      #pp. 130, 444 in O'Reilly Perl book

$ip = inet_aton('131.211.194.40')  or die "$0: inet_aton failed";
$address = sockaddr_in(7, $ip)     or die "$0: sockaddr_in failed";

print "Trying 131.211.194.40...\n";
connect(S, $address)               or die "$0: $!";
print "Connected to smail.let.uu.nl.\n";

print S "hello\n";   #Write one line to the socket.
$line = <S>;         #Read one line from the socket.
print $line;

shutdown(S, SHUT_RDWR)             or die "$0: $!";
print "Connection closed.\n";
exit 0;
```



```
1$ chmod 755 socket.pl
2$ mv socket.pl ~/bin
3$ ls -l ~/bin/socket.pl
4$ socket
Trying 131.211.194.40...
```

Make the perlscript executable: **rwxr-xr-x**
if the perlscript is not already in **~/bin**

socket in Java:

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/Echo.java**

```
 1 import java.io.*;
 2 import java.net.*;
 3
 4 class Echo {
 5     static public void main(String[] argv) {
 6         try {
 7             System.out.println("Trying 131.211.194.40...");
 8             Socket echo = new Socket("131.211.194.40", 7);
 9             System.out.println("Connected to smail.let.uu.nl.");
10
11             OutputStream out = echo.getOutputStream();
12             InputStream in = echo.getInputStream();
13
14             byte[] buffer = new byte[100];
15             buffer[0] = 'A';
16             out.write(buffer, 0, 1);
17
18             buffer[0] = '\0';
19             int len = in.read(buffer);
20             if (len > 0) {
21                 System.out.write(buffer, 0, len);
22             }
23             System.out.print('\n');
24
25             echo.close();
26         }
27
28         catch (UnknownHostException e) {
29             e.printStackTrace(System.err);
30         }
31
32         catch (IOException e) {
33             e.printStackTrace(System.err);
34         }
35
36         System.out.println("Connection closed.");
37         System.exit(0);
38     }
39 }
```

```
1$ javac Echo.java
2$ ls -l Echo.class
3$ java Echo
Trying 131.211.194.40...
Connection closed.
```

Run the java compiler.

Run the java interpreter (a.k.a the Java Virtual Machine).

telnet

▼ Homework 3.2: write your own version of telnet

Write a rudimentary version of **`telnet`** named **`mytelnet`**. Hand in only the last version; no credit otherwise.

In C, start by changing the name of **`socket.c`** to **`mytelnet.c`**. In Perl, start by changing the name of **`socket`** to **`mytelnet`**.

(1) Instead of hardwiring the IP address **`131.211.194.47`** into the program, hardwire the fully qualified domain name **`spider.let.uu.nl`**. After all, human beings prefer names.

```
 1 /* Excerpts from the file /usr/include/netdb.h ("net database"),
 2 Curry pp. 393-396 */
 3
 4 struct hostent {              /* excerpts from the fields of this structure */
 5     int h_length;             /* number of bytes in the address */
 6     char **h_addr_list;       /* ptr to ptr to first byte in the address */
 7 };
 8
 9 /* If gethostbyname fails, it will put one of the following values into
10 the variable h_errno: */
11
12 #define HOST_NOT_FOUND  1  /* Authoritative Answer Host not found */
13 #define TRY_AGAIN       2  /* Non-Authoritative Host not found, or SERVERFAIL */
14 #define NO_RECOVERY     3  /* Non recoverable errors, FORMERR, REFUSED, NOTIMP */
15 #define NO_DATA         4  /* Valid name, no data record of requested type */
16 #define NO_ADDRESS      NO_DATA   /* no address, look for MX record */
```

The following program can be modified to output every IP address of the host **`spider.let.uu.nl`**, but you should use only the first IP address in your homework. See K&R p. 52 for the following use of **`?:`**.

—Source code on the Web at **`http://i5.nyu.edu/~mm64/x52.9544/src/gethostbyname.c`**

```
 1 /* Output the IP address of www.uu.nl. */
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <netdb.h>   /* network data base */
 5
 6 extern int h_errno;
 7
 8 int main(int argc, char **argv)
 9 {
10     struct hostent *p = gethostbyname("www.uu.nl");
11     int a;  /* loop through the host's IP addresses */
12     int b;  /* loop through the octets of each IP address */
13
14     if (p == NULL) {
15         fprintf(stderr, "%s: gethostbyname error number %d\n", argv[0], h_errno);
```



```
16          return EXIT_FAILURE;
17      }
18
19      printf("Each address consists of %d octets.\n", p->h_length);
20
21      for (a = 0; p->h_addr_list[a] != NULL; ++a) {
22          for (b = 0; b < p->h_length; ++b) {
23              printf("%d%c", p->h_addr_list[a][b] & 0xFF,
24                  b < p->h_length - 1 ? '.' : '\n');
25          }
26      }
27
28      return EXIT_SUCCESS;
29  }
```

```
    1$ prog
    Each address consists of 4 octets.
    131.211.194.47
```

A pointer to a pointer

In the above program, **`p->h_addr_list`** can be treated as a two-dimensional array, i.e., it can be given two subscripts. This is true of any **`char **`** in C:

```
char **h_addr_list;
char **argv;
```

`argv[0][0]`	*first character of the first word on the command line*
`argv[0][1]`	*second character of the first word on the command line*
`argv[0][2]`	*third character of the first word on the command line*
`argv[1][0]`	*first character of the second word on the command line*
`argv[1][1]`	*second character of the second word on the command line*
`argv[1][2]`	*third character of the second word on the command line*
`argv[2][0]`	*first character of the third word on the command line*
`argv[2][1]`	*second character of the third word on the command line*
`argv[2][2]`	*third character of the third word on the command line*

Remove the sign extension with &: K&R pp. 44−45, 48−49

```
23              printf("%d%c", p->h_addr_list[a][b] & 0xFF,
```

	binary
`p->h_addr_list[0][0]`	**`10000011`**

```
  p->h_addr_list[0][0] with sign extension   11111111 11111111 11111111 10000011
&                                     0xFF   00000000 00000000 00000000 11111111
                 p->h_addr_list[0][0] & 0xFF   00000000 00000000 00000000 10000011
```

Perl **`gethostbyname`** is now unnecessary, since both of the following do the same thing:

```
inet_aton('spider.let.uu.nl')
inet_aton('131.211.194.47')
```

(2) Instead of hardwiring the fully qualified domain name **spider.let.uu.nl** into the program, let the user specify it as the first command line argument:

```
2$ gcc -o ~/bin/mytelnet mytelnet.c -lsocket -lnsl
3$ ls -l ~/bin/mytelnet
4$ mytelnet spider.let.uu.nl
Trying 131.211.194.47...
Connected to spider.let.uu.nl.
A
Connection closed.
5$
```

First, if there is not exactly one command line argument, output an error message to **stderr** and exit with a non-zero exit status. In C,

```
1     if (argc != 2) {
2         fprintf(stderr, "%s: requires exactly one command line argument.\n",
3             argv[0]);
4         return 1;
5     }
```

In Perl,

```
if (@ARGV != 1) {                    #@ARGV is in a scalar context
    die "$0: requires exactly one command line argument.";
}
```

Now that we've verified that there is an argument, pass it to **gethostbyname** in C, or to **inet_aton** in Perl. In C, the first command line argument is **argv[1]**; in Perl, the first command line argument is **$ARGV[0]**:

```
1     entry = gethostbyname(argv[1]);                              /* C */
2     $ip = inet_aton($ARGV[0]) or die "$0: inet_aton failed";   #Perl
```

(3) Instead of hardwiring the port number **7** into the program, let the user specify it as the second command line argument:

```
6$ gcc -o ~/bin/mytelnet mytelnet.c -lsocket -lnsl
7$ ls -l ~/bin/mytelnet
8$ mytelnet spider.let.uu.nl 7
Trying 131.211.194.47...
Connected to spider.let.uu.nl.
A
Connection closed.
9$
```

In C, the second command line argument is **argv[2]**; In Perl, the second command line argument is **$ARGV[1]**. But first, output an error message to **stderr**, and **exit** with a non-zero exit status if there are not exactly two command line arguments. In C,

```
1     if (argc != 3) {
2         fprintf(stderr, "%s: requires exactly 2 command line arguments.\n",
3             argv[0]);
4         return 1;
5     }
```

In Perl,


```
if (@ARGV != 2) {                    #@ARGV is in a scalar context
    die "$0: requires exactly 2 command line arguments.";
}
```

Then make sure that the second argument is not the null string. Then make sure that every character in the second argument is a digit. Then convert the second argument from string to integer. In C,

```
 1 #include <string.h>  /* for strlen, strspn, and size_t */
 2
 3     size_t length;
 4
 5     if (argc != 3) {
 6         error message and exit;
 7     }
 8
 9     length = strlen(argv[2]);
10     if (length <= 0) {
11         error message and exit;
12     }
13
14     if (strspn(argv[2], "0123456789") != length) {    /* K&R p. 250 */
15         error message and exit;
16     }
17
18     echo.sin_port = htons(atoi(argv[2]));             /* K&R p. 251 */
```

In Perl, it's much easier to validate the second argument. Write a regular expression in **/**slashes**/** (as in **awk**). **\d** is a wildcard whose meaning is the same as **[0-9]**, and **+** means ''one or more''. The operator **!~** means ''does not match''.

```
    if ($ARGV[1] !~ /^\d+$/) {
        die "$0: second argument $ARGV[1] must be a port number\n";
    }
```

(4) If the user specifies only one command line argument, use the port number of the **telnet** server by default. Call **getservbyname** (Curry pp. 396−397) to find the port number of the **telnet** server, and store it in the **sin_port** field.

```
1 /* Excerpt from the file /usr/include/netdb.h. */
2 struct servent {
3     int s_port;               /* port number */
4 };
```

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/getservbyname.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <netdb.h>
 4
 5 int main(int argc, char **argv)
 6 {
 7     struct servent *p = getservbyname("telnet", "tcp");
 8
 9     if (p == NULL) {
10         perror(argv[0]);
11         return EXIT_FAILURE;
12     }
```



```
13
14      printf("decimal\t%d\n", p->s_port);
15      printf(  "hex\t%04X\n", p->s_port);
16
17      return EXIT_SUCCESS;
18  }
```

```
10$ gcc -o ~/bin/getservbyname getservbyname.c
11$ ls -l ~/bin/getservbyname getservbyname
12$ getservbyname
decimal     5888
hex         1700
```

17 *in hex is* **23** *in decimal.*

You don't need to apply **htons** in C or C++ to the return value of **getservbyname**: the bytes that **getservbyname** returns are already in network order.

```
#!/bin/perl

$port = getservbyname('telnet', 'tcp') or die "$0: couldn't find telnet";

print "$port\n";
exit 0;
```

```
13$ getservbyname
23
```

Unfortunately, you won't be able to use **mytelnet** to connect to the **telnet** server at port 23: it speaks a complicated protocol that we won't cover in class.

(5) In C, use **fputc** and **fgetc** (K&R pp. 246−247; see **fputc**(3) and **fgetc**(3)) instead of **write** and **read**. Call **fdopen** (Handout 2, pp. 18−19; Curry p. 101) only once. The **"r+"** argument of **fdopen** will let you input and output (K&R p. 242). The **setbuf** will make you flush automatically every time you output to the socket; see K&R p. 243.

fclose normally does two things for you: it flushes the buffer and then severs the connection between your program and a file. But there is no need to call **fclose** here, because the buffer never needs flushing (thanks to **setbuf**), and because we're using **shutdown** to sever the connection with the server.

★ **putchar**, **fputc**, and **putc** can output either a **char** or an **int**, but the return value of **getchar**, **fgetc**, and **getc** must be stored in an **int**; see K&R p. 16. Therefore you should use an **int** variable to hold the character going to and coming from the socket.

(For your information, the **%c** format of **printf** or **fprintf** can output either a **char** or an **int**, but the **%c** format of **scanf** or **fscanf** can input only a **char**.)

```
 1      int s = socket(...;
 2      FILE *fp;
 3
 4      printf Trying in line 42 of socket.c;
 5      Call connect(s, ... in line 44 of socket.c;
 6      printf Connected to in line 49 of socket.c;
 7
 8      fp = fdopen(s, "r+");
 9      if (fp == NULL) {
10          perror(argv[0]);
11          return whatever exit number you're up to;
12      }
13      setbuf(fp, NULL);               /* like Perl autoflush */
```



```
14
15      Call fprintf(fp, "%c", c) instead of write(s, &c, 1) in line 51 of socket.c;
16      Call fscanf(fp, "%c", &c) instead of read(s, &c, 1) in line 58 of socket.c;
17      printf the character;
18      Call shutdown;
19      printf Connection closed.
```

(6) So far, **mytelnet** outputs and inputs exactly one character (an uppercase **A** *à la* Nathaniel Hawthorne) to and from the socket. Change it to do this to every character read from the standard input.

Output the message

```
Escape character is '^d'.
```

immediately after the **Connected to** message, and then write a traditional **while-getchar** loop to input characters one at a time from the standard input. As in the previous section of this Homework, use an **int** variable to hold each character. During each loop, output the character to the socket with one **fputc**. Then to make sure that the socket really works, put a **'\0'** into the variable that held the character. Then input one character from the socket into the same variable with **fgetc**. Finally, output the character in the variable to the standard output with **putchar**.

When you run **mytelnet** with its standard input and output connected to the terminal keyboard and screen, everything you type will be echoed on the screen via **spider.let.uu.nl**. The characters you type are in *italics*:

```
14$ mytelnet spider.let.uu.nl 7
Trying 131.211.194.47...
Connected to spider.let.uu.nl.
Escape character is '^d'.
"And this also," said Marlow suddenly,
"And this also," said Marlow suddenly,
"has been one of the dark places of the earth."
"has been one of the dark places of the earth."
control-d
Connection closed by foreign host.
My child's exit status was 0.
15$
```

(7) The server at the remote end of our socket (in this case **echo** on **spider.let.uu.nl**) performs input and output in a predictable order and quantity. It always begins by inputting one byte, not outputting. And every time it inputs one byte, it outputs exactly one byte immediately afterwards.

Not all servers will be so tame. For example, some produce output but accept no input. Others perform input and output in an unpredictable order and quantity.

When conducting a dialog with such a server, it may be impossible for **mytelnet** to know whether it should input or output at any given moment. The easiest way to attempt both, simultaneously, is to split **mytelnet** into a parent and a child. Let one of them continuously attempt output, the other continuously attempt input:

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/mytelnet.c**

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8 #include <unistd.h>
```



```
 9
10 int main(int argc, char **argv)
11 {
12     int s = socket(AF_INET, SOCK_STREAM, 0);
13     struct sockaddr_in address;
14     char buffer[INET6_ADDRSTRLEN];
15     pid_t pid;
16     int status;
17     int i;    /* getchar requires an int: KR p. 16 */
18     char c;   /* write requires a char: KR p. 170 */
19
20     if (s < 0) {
21         perror(argv[0]);
22         return 1;
23     }
24
25     bzero((char *)&address, sizeof address);
26     address.sin_family = AF_INET;
27     address.sin_port = htons(7);
28
29     /* IP address for spider.let.uu.nl: */
30     i = inet_pton(AF_INET, "131.211.194.47", &address.sin_addr);
31     if (i == 0) {
32         fprintf(stderr, "%s: bad dotted string to IP address\n", argv[0]);
33         return 2;
34     }
35
36     if (i != 1) {
37         perror(argv[0]);
38         return 3;
39     }
40
41     if (inet_ntop(AF_INET, &address.sin_addr, buffer, sizeof buffer) == NULL) {
42         perror(argv[0]);
43         return 4;
44     }
45     printf("Trying %s...\n", buffer);
46
47     if (connect(s, (struct sockaddr *)&address, sizeof address) != 0) {
48         perror(argv[0]);
49         return 5;
50     }
51
52     printf("Connected to spider.let.uu.nl.\n"
53         "Escape character is '^d'.\n");
54
55     pid = fork();
56     if (pid < 0) {
57         perror(argv[0]);
58         return 6;
59     }
60
61     if (pid == 0) {
62         /* The child will copy from the server to the stdout. */
```



```
63          while (read(s, &c, 1) == 1) {
64              putchar(c);
65          }
66
67          fprintf(stderr, "Connection closed by foreign host.\n");
68          return EXIT_SUCCESS;
69      }
70
71      /* The parent will copy from the stdin to the server. */
72      while ((i = getchar()) != EOF) {
73          c = i;
74          write(s, &c, 1);
75      }
76
77      if (shutdown(s, SHUT_RDWR) != 0) {   /* terminate the while loop in line 63 */
78          perror(argv[0]);
79          return 7;
80      }
81
82      wait(&status);
83      if (WIFEXITED(status)) {
84          printf("My child's exit status was %d.\n", WEXITSTATUS(status));
85      }
86      return EXIT_SUCCESS;
87  }
```

```
16$ gcc -o ~/bin/mytelnet mytelnet.c -lsocket -lnsl
17$ ls -l ~/bin/mytelnet
18$ mytelnet
```

Retrofit features (1) through (6) into the above program. Call **fdopen** (with the **"r+"** argument) and **setbuf** before the **fork**. As in the previous section, the parent should call **shutdown** but not **fclose**. The child should call neither: only one process has to call **shutdown**. The characters you type are in *italics*.

19$ *mytelnet spider.let.uu.nl 7*
Trying 131.211.194.47...
Connected to spider.let.uu.nl.
Escape character is '^d'.
"And this also," said Marlow suddenly,
"And this also," said Marlow suddenly,
"has been one of the dark places of the earth."
"has been one of the dark places of the earth."
control-d
Connection closed by foreign host.
My child's exit status was 0.
20$

▲

A future improvement to mytelnet

The above child will loop as long as the parent does, because the **echo** server inputs and outputs equal numbers of bytes. Other servers, however, may input and output unequal numbers of bytes.

An example of this is the **daytime** server at port 13: it inputs nothing, but outputs one line. In this case, the child would encounter the end-of-input from the socket and voluntarily exit while the parent is

still alive and waiting for standard input. The user must then type a **control-d** to the parent to make the parent exit:

```
1$ mytelnet spider.let.uu.nl 13
Friday, January 09, 2004 12:08:35 AM-EST
Connection closed by foreign host.      It types this.
control-d                               You have to type this.
My child's exit status was 0.           It types this.
2$
```

To eliminate the need for the **control-d**, later in the course the parent will check for the death-of-child signal (**SIGCHLD** in Curry pp. 243−244 and **signal**(3); **SIGCLD** in Bach p. 449 and p. 212, figure 7.14) during each loop, and terminate the loop as soon as the child exits.

Specify port number 79 for the **finger** server. You must **write** one line of data to the remote **finger** before it will **write** data back to you. **write** a **\n** to get a list of everyone who is logged in; **write abc1234\n** to get information about one person.

```
3$ mytelnet acf2.nyu.edu 79
RETURN                                  just press RETURN to see everyone
control-d

4$ mytelnet acf2.nyu.edu 79
root RETURN                             or type any other login name
control-d
```

Chicken out with fdopen

You can **fprintf** a file only if you **fopen** it first. But suppose you **open** the file because you plan to use **write** instead of **fprintf**. Then immediately after the **open**, you chicken out and wish you could use **fprintf** instead.

It's not too late:

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/fdopen.c**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #include <fcntl.h>
 5 #include <sys/types.h>
 6 #include <sys/stat.h>
 7 #include <unistd.h>
 8
 9 int main(int argc, char **argv)
10 {
11     int fd = open("/home/a/abc1234/outfile",
12         O_CREAT | O_TRUNC | O_WRONLY,
13         S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
14     FILE *fp;
15
16     if (fd < 0) {
17         perror(argv[0]);
18         return 1;
19     }
20
21     fp = fdopen(fd, "w");
22     if (fp == NULL) {
```



```
23          perror(argv[0]);
24          return 2;
25      }
26
27      fprintf(stderr, "fd == %d, fp->_file == %d\n", fd, fp->_file);
28      fprintf(fp, "hello\n");
29
30      if (fclose(fp) != 0) {
31          perror(argv[0]);
32          return 3;
33      }
34
35      return EXIT_SUCCESS;
36  }
```

```
fd == 3, fp->_file == 3
```

fdopen is just like **fopen**, except that **fdopen** does not call **open**:

```
 1  FILE *fopen(char *filename, char *mode)
 2  {
 3      int arg2;
 4      mode_t arg3;
 5
 6      if (mode[0] == 'r') {
 7          arg2 = O_RDONLY;
 8          arg3 = 0;
 9      } else if (mode[0] == 'w') {
10          arg2 = O_CREAT | O_TRUNC | O_WRONLY;
11          arg3 = S_IRUSR | S_IWUSR;
12      } else {
13          fprintf(stderr, "illegal mode %s\n", mode);
14      }
15
16      return fdopen(open(filename, arg2, arg3), mode);
17  }
```

▼ Homework 3.3: fdopen an input file

open an input file. Then **fdopen** it and **fscanf** some data from it. Echo the data to the standard output to verify that the **fscanf** worked.

▲

—Source code on the Web at **http://i5.nyu.edu/~mm64/x52.9544/src/strerror.c**

```
 1  #include <stdio.h>   /* for printf */
 2  #include <stdlib.h>  /* for EXIT_SUCCESS */
 3  #include <string.h>  /* for strerror */
 4
 5  int main()
 6  {
 7      int i;
 8      const char *p;   /* p is a read-only pointer */
 9
10      for (i = 0; (p = strerror(i)) != NULL; ++i) {
11          printf("%3d: %s\n", i, p);
12      }
```

```
13
14      return EXIT_SUCCESS;
15  }
```

```
  0: Error 0
  1: Not owner
  2: No such file or directory
  3: No such process
  4: Interrupted system call

147: Host is down
148: No route to host
149: Operation already in progress
150: Operation now in progress
151: Stale NFS file handle
```

□

