Fall 2006 Handout 8

A client that reads and writes at the same time

—On the Web at
http://i5.nyu.edu/˜mm64/x52.9547/src/mytelnet.c

```
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>   /* for getservbyname */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>

int main(int argc, char **argv)
{
    int s;                        /* file descriptor */
    struct sockaddr_in address;
    char buffer[INET_ADDRSTRLEN]; /* for inet_pton */
    struct hostent *entry;        /* for gethostbyname, gethostbyaddr */
    pid_t pid;
    int c;                        /* return value of getchar is int, not char */
    int status;

    bzero((char *)&address, sizeof address);
    address.sin_family = AF_INET;

    if (argc == 2) {   /* one command line argument */
        const struct servent *const p = getservbyname("telnet", "tcp");
        if (p == NULL) {
            fprintf(stderr, "%s: couldn’t find telnet port number\n",
                argv[0]);
            return 1;
        }
        address.sin_port = p->s_port; /* doesn’t need htons */
    }

    else if (argc == 3) {   /* two command line arguments */
        /* %hd for "short decimal": a port number is 16 bits */
        if (sscanf(argv[2], "%hd", &address.sin_port) != 1) {
            fprintf(stderr, "%s: %s is not a legal port number\n",
                argv[0], argv[2]);
            return 2;
        }
        address.sin_port = htons(address.sin_port);
    }

    else {
        fprintf(stderr, "%s: usage hostname [port]\n", argv[0]);
```

```
            return 3;
        }

        entry = gethostbyname(argv[1]);
        if (entry == NULL) {
            fprintf(stderr, "%s: gethostbyname, h_errno == %d\n",
                argv[0], h_errno);
            return 4;
        }
        address.sin_addr.s_addr = *(in_addr_t *)entry->h_addr_list[0];

        s = socket(AF_INET, SOCK_STREAM, 0);
        if (s < 0) {
            perror(argv[0]);
            return 5;
        }

        if (inet_ntop(AF_INET, &address.sin_addr, buffer, sizeof buffer) == NULL) {
            perror(argv[0]);
            return 6;
        }
        printf("Trying %s...\n", buffer);

        if (connect(s, (struct sockaddr *)&address, sizeof address) != 0) {
            perror(argv[0]);
            return 7;
        }

        entry = gethostbyaddr((char *)&address.sin_addr, sizeof address.sin_addr,
            AF_INET);
        if (entry == NULL) {
            fprintf(stderr, "%s: gethostbyaddr, h_errno == %d\n",
                argv[0], h_errno);
            return 8;
        }
        printf("Connected to %s.\n", entry->h_name);

        pid = fork();
        if (pid < 0) {
            perror(argv[0]);
            return 9;
        }

        if (pid == 0) {
            /* Arrive here if I am the child.
            Copy from the server to the stdout. */

            char c;
            int n;
            while ((n = read(s, &c, 1)) == 1) {
                putchar(c);
            }
            if (n < 0) {
                perror(argv[0]);
            return 1;
        }
        return EXIT_SUCCESS;
    }

    /* Arrive here if I am the parent.
    Copy from the stdin to the server. */

    while ((c = getchar()) != EOF) {
        char ch = c;
        if (write(s, &ch, 1) != 1) {
            perror(argv[0]);
            return 10;
        }
    }

    if (shutdown(s, SHUT_RDWR) != 0) {
        perror(argv[0]);
        return 11;
    }

    pid = wait(&status);
    if (!WIFEXITED(status) || WEXITSTATUS(status) != EXIT_SUCCESS) {
        fprintf(stderr, "%s: child did not return exit status 0.\n",
            argv[0]);
        return 12;
    }

    return EXIT_SUCCESS;
}
```

```
1$ gcc -o ~/bin/mytelnet mytelnet.c -lsocket -lnsl
2$ ls -l ~/bin/mytelnet
-rwx--x--x   1 mm64     users      10172 Jan  4 10:30 /home1/a/abc1234/bin/mytelnet
```

```
3$ mytelnet labinfu.unipv.it 7
Trying 193.204.35.58...
Connected to labinfu.unipv.it.
"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                 to make the parent break out of its while loop
Connection closed.        Graceful termination.

4$ echo $?                exit status
0
```

—On the Web at
http://i5.nyu.edu/˜mm64/x52.9547/src/mytelnet.pl

```
#!/bin/perl
#Output one line to the echo server at port 7 of labinfu.unipv.it
#(193.204.35.58).  Then input the line back from the server.

```

```
use Socket;
use FileHandle;  #for autoflush
use POSIX;      #for WIFEXITED and WEXITSTATUS

if (@ARGV == 1) {   #This @ARGV is in a scalar context
    $port = getservbyname(’telnet’, ’tcp’) #look up telnet in /etc/services
        or die "$0: couldn’t find port number for telnet";
} elsif (@ARGV == 2) {
    $port = $ARGV[1];
} else {
    die "$0: usage: hostname [port]";
}

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

$ip = inet_aton($ARGV[0])          or die "$0: inet_aton failed";
$address = sockaddr_in($port, $ip) or die "$0: sockaddr_in failed";

$dotted = inet_ntoa($ip);
print "Trying $dotted...\n";
connect(S, $address)               or die "$0: $!";
$name = gethostbyaddr($ip, AF_INET) or $name = $dotted;
print "Connected to $name.\n";

$pid = fork();
defined $pid or die "$0: $!";

if ($pid == 0) {
    #Arrive here if I am the child.
    #Copy from the server to the stdout.

    while (($_ = <S>) ne ’’ ) {  #Read one line from the socket
        print STDOUT $_;     #and write it to the stdout.
    }

    exit 0;
}

#Arrive here if I am the parent.
#Copy from the stdin to the server.

while (($_ = <STDIN>) ne ’’) {       #Read one line from the stdin
    print S $_;                  #and write it to the server.
}

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

$pid = wait();
if ($pid < 0) {
    die "$0: $!";
}
```

```
59 if (!WIFEXITED($?) || WEXITSTATUS($?) != 0) {
60     print STDERR "child did not return exit status 0.\n";
61 }
62 exit 0;
```

The above lines 37 and 47 may be simplified to

```
37     while (<S>) {
47     while (<STDIN>) {
```

```
5$ mytelnet.pl labinfu.unipv.it 7
Trying 193.204.35.58...
Connected to labinfu.unipv.it.
"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                to make the parent break out of its while loop
Connection closed.       Graceful termination.

6$ echo $?               exit status
0
```

Catch the death-of-child signal

In a dialog with the **echo** server, the client terminates the conversation. But in a dialog with the **daytime** server, the server terminates the conversation:

```
1$ telnet labinfu.unipv.it 13
Trying 193.204.35.58...
Connected to labinfu.unipv.it.
Escape character is '^]'.
Wed Jan  4 15:49:23 2006
Connection to labinfu.unipv.it closed by foreign host.
2$
```

As shown by the above, the real **telnet** client turns itself off when the server closes the TCP connection. But our **telnet** requires a **control-d** to make the parent break out of its **while** loop:

```
3$ mytelnet.pl labinfu.unipv.it 13
Trying 193.204.35.58...
Connected to labinfu.unipv.it.
Wed Jan  4 15:49:23 2006
control-d                Ungraceful: I wish I didn't have to type this control-d.
Connection closed.
4$
```

To see the signal numbers,

```
5$ kill -l | more             minus lowercase L for "list"
6$ man -s 3head signal        read it at http://i5.nyu.edu/~mm64/man/
```

```
7$ awk '$1 == "#define" && $2 ~ /^SIG(HUP|FPE|KILL|BUS|PIPE|ALRM|CH?LD|TSTP|CONT)$/' \
    /usr/include/sys/iso/signal_iso.h
#define SIGHUP  1   /* hangup */
#define SIGFPE  8   /* floating point exception */
#define SIGKILL 9   /* kill (cannot be caught or ignored) */
#define SIGBUS  10  /* bus error */
#define SIGPIPE 13  /* write on a pipe with no one to read it */
#define SIGALRM 14  /* alarm clock */
#define SIGCLD  18  /* child status change */
#define SIGCHLD 18  /* child status change alias (POSIX) */
#define SIGTSTP 24  /* user stop requested from tty */
#define SIGCONT 25  /* stopped process has been continued */
```

—On the Web at
`http://i5.nyu.edu/~mm64/x52.9547/src/mytelnet2.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>   /* for getservbyname */
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <sys/wait.h>

char *progname;
int s;                              /* file descriptor */
void death_of_child(int sig);

int main(int argc, char **argv)
{
    struct sockaddr_in address;
    char buffer[INET_ADDRSTRLEN];   /* for inet_pton */
    struct hostent *entry;          /* for gethostbyname, gethostbyaddr */
    pid_t pid;
    int c;

    progname = argv[0];
    bzero((char *)&address, sizeof address);
    address.sin_family = AF_INET;

    if (argc == 2) {   /* one command line argument */
        const struct servent *const p = getservbyname("telnet", "tcp");
        if (p == NULL) {
            fprintf(stderr, "%s: couldn't find telnet port number\n",
                progname);
            return 1;
        }
        address.sin_port = p->s_port;
    }

    else if (argc == 3) {   /* two command line arguments */
        if (sscanf(argv[2], "%hd", &address.sin_port) != 1) {
            fprintf(stderr, "%s: %s is not a legal port number\n",
                progname, argv[2]);
            return 2;
        }
        address.sin_port = htons(address.sin_port);
    }

    else {
        fprintf(stderr, "%s: usage hostname [port]\n", progname);
        return 3;
    }

    entry = gethostbyname(argv[1]);
    if (entry == NULL) {
        fprintf(stderr, "%s: gethostbyname, h_errno == %d\n",
            progname, h_errno);
        return 4;
    }
    address.sin_addr.s_addr = *(in_addr_t *)entry->h_addr_list[0];

    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror(progname);
        return 5;
    }

    if (inet_ntop(AF_INET, &address.sin_addr, buffer, sizeof buffer) == NULL) {
        perror(progname);
        return 6;
    }
    printf("Trying %s...\n", buffer);

    if (connect(s, (struct sockaddr *)&address, sizeof address) != 0) {
        perror(progname);
        return 7;
    }

    entry = gethostbyaddr((char *)&address.sin_addr, sizeof address.sin_addr,
        AF_INET);
    if (entry == NULL) {
        fprintf(stderr, "%s: gethostbyaddr, h_errno == %d\n",
            progname, h_errno);
        return 8;
    }
    printf("Connected to %s.\n", entry->h_name);

    if (signal(SIGCHLD, death_of_child) == SIG_ERR) {
        perror(progname);
        return 9;
    }

    pid = fork();
    if (pid < 0) {
        perror(progname);
        return 10;
    }

    if (pid == 0) {
        /* Arrive here if I am the child.
        Copy from the server to the stdout. */

        char c;
        while (read(s, &c, 1) == 1) {
            putchar(c);
        }
        return EXIT_SUCCESS;
    }

    /* Arrive here if I am the parent.
    Copy from the stdin to the server. */

    while((c = getchar()) != EOF) {
        char ch = c;
        if (write(s, &ch, 1) != 1) {
            perror(progname);
            return 11;
        }
    }

    if (shutdown(s, SHUT_RDWR) != 0) {
        perror(progname);
        return 12;
    }

    printf("Connection closed.\n");
    return EXIT_SUCCESS;
}

void death_of_child(int sig)
{
    int status;

    if (wait(&status) < 0) {
        perror(progname);
        exit(13);
    }

    if (!WIFEXITED(status) || WEXITSTATUS(status) != EXIT_SUCCESS) {
        fprintf(stderr, "%s: child did not return exit status %d.\n",
            progname, EXIT_SUCCESS);
    }

    if (shutdown(s, SHUT_RDWR) != 0) {
        perror(progname);
        exit(14);
    }

    printf("Connection closed by foreign host.\n");
    exit(EXIT_SUCCESS);
}
```

—On the Web at
http://i5.nyu.edu/~mm64/x52.9547/src/mytelnet2.pl

```
#!/bin/perl
#Telnet client.

use Socket;
use FileHandle;   #for autoflush
use POSIX;        #for WIFEXITED and WEXITSTATUS

if (@ARGV == 1) {   #This @ARGV is in a scalar context
    $port = getservbyname(’telnet’, ’tcp’) #look up telnet in /etc/services
        or die "$0: couldn’t find port number for telnet";
} elsif (@ARGV == 2) {
    $port = $ARGV[1];
} else {
    die "$0: requires 1 or 2 command line arguments";
}

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

$ip = inet_aton($ARGV[0])          or die "$0: inet_aton failed";
$address = sockaddr_in($port, $ip) or die "$0: sockaddr_in failed";

$dotted = inet_ntoa($ip);
print "Trying $dotted...\n";
connect(S, $address)               or die "$0: $!";
$name = gethostbyaddr($ip, AF_INET) or $name = $dotted;
print "Connected to $name.\n";

$SIG{CHLD} = ’death_of_child’;   #Must be before the fork.
$pid = fork();
defined $pid or die "$0: $!";

if ($pid == 0) {
    #Arrive here if I am the child.
    #Copy from the server to the stdout.

    while (<S>) {            #Read one line from the socket
        print STDOUT $_;     #and write it to the stdout.
    }

    exit 0;
}

#Arrive here if I am the parent.
#Copy from the stdin to the server.
while (<STDIN>) {                #Read one line from the stdin
    print S $_;                  #and write it to the server.
}

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

```

sub death_of_child {
    $pid = wait();
    if (!WIFEXITED($?) || WEXITSTATUS($?) != 0) {
        print STDERR "Child did not return exit status 0.\n";
    }

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

Now it terminates gracefully:

```
8$ mytelnet2.pl labinfu.unipv.it 13
Trying 193.204.35.58...
Connected to labinfu.unipv.it.
Wed Jan  4 15:49:33 2006
Connection closed by foreign host.
9$
```

We’ve built a primitive web browser

```
1$ mytelnet2.pl i5.nyu.edu 80
Trying 128.122.253.152...
Connected to i5.nyu.edu.
GET /index.html HTTP/1.1            You type these two lines of HTTP.
Host: i5.nyu.edu                    Press RETURN twice.

HTTP/1.1 200 OK
Date: Wed, 04 Jan 2006 15:31:11 GMT
Server: Apache/1.3.29 (Unix) mod_perl/1.25
Last-Modified: Tue, 27 Sep 2005 18:15:17 GMT
ETag: "1219-196d-43398c35"
Accept-Ranges: bytes
Content-Length: 6509
Content-Type: text/html



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <meta http-equiv="Cache-Control" content="no-cache" />

    <title>NYU&gt; i5 Home Pages</title>
    <style type="text/css">
        @import url("http://www.nyu.edu/v45/css/simplenyu.css");
        #outerBox {width: 504px;}
    </style>

</script>


</head>

</html>
```

Speak the telnet protocol

Will you, won't you, will you, won't you,
will you join the dance?
Will you, won't you, will you, won't you,
won't you join the dance?

—Lewis Carroll, *Alice's Adventures in Wonderland*, chapter 10

A typical telnet protocol command looks like this. Each box is one byte. See Handout 4, p. 19.

IAC 255	**WILL 251**	**option**

A typical reply is

IAC 255	**DONT 254**	**option**

The **==** in lines 14 and 52, and the **!=** in lines 53 and 96, perform numeric comparison. The **eq** in lines 64 and 70, and the **ne** in line 53 perform string comparison.

The **$line = <S>** in Handout 7, p. 28, line 19 will input an entire line. The **sysread** in lines 52, 59, and 60 will read a single byte. It returns the number of bytes read.

See **in.telnetd**(1M) for the telnet options that our telnet server is willing to do. For example, an **$option** value of 36 in line 60 stands for ‘‘environment variables’’. See RFC 1408 (and 1571) at

ftp://ftp.rfc-editor.org/in-notes/rfc1408.txt

—On the Web at
http://i5.nyu.edu/˜mm64/x52.9547/src/mytelnet3.pl

```
 1 #!/bin/perl
 2 #Telnet client.  Speaks the telnet protocol and declines all telnet options.
 3
 4 use Socket;
 5 use POSIX;        #for WIFEXITED and WEXITSTATUS
 6
 7 use FileHandle;   #for autoflush
 8 STDOUT->autoflush();
 9
10 $telnet_port = getservbyname('telnet', 'tcp')   #look up telnet in /etc/services
11     or warn "$0: couldn't find port number for telnet";
12     exit 1;
13
14 if (@ARGV == 1) {   #This @ARGV is in a scalar context
15     $port = $telnet_port;
16 } elsif (@ARGV == 2) {
17     $port = $ARGV[1];
18 } else {
19     warn "$0: requires 1 or 2 command line arguments";
20     exit 2;
21 }
22
23 socket(S, AF_INET, SOCK_STREAM, 0) or die "$0: $!";
24 S->autoflush();      #pp. 130, 444 in O'Reilly Perl book
25
26 $ip = inet_aton($ARGV[0])          or die "$0: inet_aton failed";
27 $address = sockaddr_in($port, $ip) or die "$0: sockaddr_in failed";
28
29 $dotted = inet_ntoa($ip);
30 print "Trying $dotted...\n";
31 connect(S, $address)               or die "$0: $!";
32 $name = gethostbyaddr($ip, AF_INET) or $name = $dotted;
33 print "Connected to $name.\n";
34
35 $SIG{CHLD} = 'death_of_child';
36 $pid = fork();
37 defined $pid or die "$0: $!";
```

```
38
39 if ($pid == 0) {
40     #Arrive here if I am the child.
41
42     #Telnet protocol codes from RFC 854
43
44     $WILL = "\xFB";   #251
45     $WONT = "\xFC";   #252
46
47     $DO   = "\xFD";   #253
48     $DONT = "\xFE";   #254
49
50     $IAC  = "\xFF";   #255: "interpret as command"
51
52     while (sysread(S, $c, 1) == 1) {   #Read one byte from the server.
53         if ($port != $telnet_port || $c ne $IAC) {
54             print $c;
55             next;        #Go back to line 52.
56         }
57
58         #Read a telnet command and the following option.
59         sysread(S, $command, 1);
60         sysread(S, $option, 1);
61
62         #If the server says it’s willing to perform a telnet option,
63         #tell it not to.
64         if ($command eq $WILL || $command eq $WONT) {
65             print S "$IAC$DONT$option";
66         }
67
68         #If the server asks us to perform a telnet option,
69         #tell it we won’t.
70         elsif ($command eq $DO || $command eq $DONT) {
71             print S "$IAC$WONT$option";
72         }
73
74         else {
75             warn "$0: Don’t know telnet command ", ord($command);
76             exit 1;
77         }
78     }
79
80     close(S) or die "$0: $!";
81     exit 0;
82 }
83
84 #Arrive here if I am the parent.
85 #Copy from the stdin to the server.
86 while (<STDIN>) {                #Read one line from the stdin
87     print S $_;                  #and write it to the server.
88 }
89
90 shutdown(S, SHUT_RDWR) or die "$0: $!";
91 print "Connection closed.\n";
```

```
exit 0;

sub death_of_child {
    $p = wait();
    if ($p != $pid) {
        warn "$0: My child is $pid, not $p";
        exit 3;
    }

    if (!WIFEXITED($?)) {
        warn "$0: My child did not return an exit status";
        exit 4;
    }

    $status = WEXITSTATUS($?);
    if ($status != 0) {
        warn "Child returned exit status $status";
        exit 5;
    }

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

Which TCP ports on i5 already have a server bound to them?

```
1$ netstat -an -f inet -P tcp | head

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
      *.*                  *.*                0      0 49152      0 IDLE
      *.111                *.*                0      0 49152      0 LISTEN
      *.*                  *.*                0      0 49152      0 IDLE
      *.13782              *.*                0      0 49152      0 LISTEN
      *.13724              *.*                0      0 49152      0 LISTEN
      *.13783              *.*                0      0 49152      0 LISTEN
```

—On the Web at
http://i5.nyu.edu/˜mm64/x52.9544/src/bound

```
#!/bin/ksh
#Output the ports that already have a server bound to them.

netstat -an -f inet -P tcp |
awk 'NR > 4 {print $1}' |
awk -F. '{print $NF}' |
sort -n |
uniq

exit 0
```

Here's the output of the above shellscript piped through **pr -7 -l3 -i' '1 -s -t** (minus lowercase i blank one, minus lowercase L):

```
    * 80 3306 5988 13782 50252 51761
    22 111 3742 13722 13783 50256 51821
    25 898 5987 13724 50251
```

Bind a server to a port

The **echo**, **daytime**, and **finger** programs on **acf.nyu.edu** are called *servers:* they wait for other programs called *clients* to ask for something, and then respond to the request. The program you write for the previous homework was a client. Now you will write a server. See David Curry's O'Reilly book *Using C on the UNIX System* pp. 400−401, 404−405.

Each server listens to requests that arrive at a given port number. For example, the **echo** server on **www.uu.nl** is *bound* to port number 7.

When run on **i5.nyu.edu**, the following server binds itself to port number 10566 on **i5.nyu.edu**. Use **INADDR_ANY** instead of 128.122.253.152 to bind a server to a port on the host on which it is running; see David Curry's O'Reilly book *Using C on the UNIX System* p. 400 and **tcp**(7). I picked the big port number 10566 to lessen the chance that some other server was using the same port number on **i5.nyu.edu**. Actually, 0566 are the last four digits of my social security number:

```
    long ss;                /* Print a long with percent lowercase LD. */
    printf ("The last four digits are %04ld.\n", ss % 10000);
```

```
    The last four digits are 0566.
```

Only the superuser is allowed bind a server to a port number less than **IPPORT_RESERVED** (David Curry's O'Reilly book *Using C on the UNIX System* pp. 412−413).

The **SO_REUSEADDR** in lines 27 and 34 allows another copy of this server to be started while a previous copy is still running at the same port number. The **L** in **SOL_SOCKET** stands for "socket level"; other levels are the IP and TCP levels.

```
1$ cat -n /usr/include/sys/socket.h | sed -n '99,101p;104p'
    99  /*
   100   * Option flags per-socket.
   101   */
   104  #define  SO_REUSEADDR 0x0004       /* allow local address reuse */
```

If several clients on several hosts try to **connect** to your server simultaneously, they will have to wait in line. The second argument of **listen** (David Curry's O'Reilly book *Using C on the UNIX System* pp. 369−370) gives the maximum length of this queue.

```
2$ cat -n /usr/include/sys/socket.h | sed -n 287,290p
   287  /*
   288   * Maximum queue length specifiable by listen.
   289   */
   290  #define  SOMAXCONN    5
```

If more than this number of clients attempt to **connect**, the **connect** function in the unlucky clients will return −1. **listen** will deposit the number **ECONNREFUSED** or **ETIMEDOUT** (**#define**'d in the file **/usr/include/errno.h**) into the variable **errno** of the unlucky clients, causing **perror** to print **Connection refused** or **Connection timed out**.

The **accept** system call (David Curry's O'Reilly book *Using C on the UNIX System* p. 370) will make the server go to sleep until a client tries to **connect** to it. Use the return value of **accept** as the file descriptor for all subsequent communication with the client. It can be used as the first argument of either **read** (line 72) or **write** (line 80) For convenience, the server should call **fdopen "r+"** so that it could **fprintf** and **fscanf** instead of **write** and **read**.

At line 89, after finishing its dialog with the client, the server could use the original file descriptor (**`s`**) as the first argument of another call to **`accept`**. This would let the server sleep until another client **`connect`**'s to it. To keep the server simple, we choose not to do this. This server will talk to only the first client that **`connect`**'s to it, and will then **`exit`**.

—On the Web at
`http://i5.nyu.edu/˜mm64/x52.9544/src/myserver.c`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <ctype.h>   /* for tolower */
 4
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7 #include <netinet/in.h>
 8
 9 int main(int argc, char **argv)
10 {
11     const u_short port = 10566;            /* sin_port field of struct sockaddr_in */
12     int s = socket(AF_INET, SOCK_STREAM, 0); /* file descriptor to accept a client */
13     int on = 1;
14     int opt_length = sizeof (int);
15     struct sockaddr_in myaddress;
16     struct sockaddr_in clientaddress;
17     socklen_t length = sizeof clientaddress;
18     int client;   /* file descriptor to talk to client */
19     char dotted[INET6_ADDRSTRLEN];           /* for inet_ntop */
20     char buffer[2];
21
22     if (s < 0) {
23         perror(argv[0]);
24         return 1;
25     }
26
27     if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &on, sizeof on) != 0) {
28         perror(argv[0]);
29         return 2;
30     }
31
32     on = 0;
33
34     if (getsockopt(s, SOL_SOCKET, SO_REUSEADDR, &on, &opt_length) != 0) {
35         perror(argv[0]);
36         return 3;
37     }
38
39     printf("The SO_REUSEADDR option of the socket is %d.\n", on);
40
41     bzero((char *)&myaddress, sizeof myaddress);
42     myaddress.sin_family = AF_INET;
43     myaddress.sin_port = htons(port);
44     myaddress.sin_addr.s_addr = INADDR_ANY;
45
46     if (bind(s, (struct sockaddr *)&myaddress, sizeof myaddress) != 0) {
47         perror(argv[0]);
48         return 4;
```

```
    }

    if (listen(s, SOMAXCONN) != 0) {
        perror(argv[0]);
        return 5;
    }

    client = accept(s, (struct sockaddr *)&clientaddress, &length);
    if (client < 0) {
        perror(argv[0]);
        return 6;
    }

    if (inet_ntop(AF_INET, &clientaddress.sin_addr, dotted, sizeof dotted)
        == NULL) {
        perror(argv[0]);
        return 7;
    }

    printf("I have accepted a client whose IP address is %s.\n", dotted);

    /* The service provided by this server is merely to input one character
    and then output it in lowercase, followed by a newline. */
    if (read(client, buffer, 1) != 1) {
        perror(argv[0]);
        return 8;
    }

    buffer[0] = tolower(buffer[0]);
    buffer[1] = '\n';

    if (write(client, buffer, sizeof buffer) != sizeof buffer) {
        perror(argv[0]);
        return 9;
    }

    if (shutdown(client, SHUT_RDWR) != 0) {
        perror(argv[0]);
        return 10;
    }

    return EXIT_SUCCESS;
}
```

Run the above program like this on its initial voyage:

```
3$ gcc -o ~/bin/myserver myserver.c -lsocket -lnsl
4$ ls -l ~/bin/myserver
5$ myserver
The SO_REUSEADDR option of the socket is 4.
```

Nothing further should happen; prompt should not reappear.

If **myserver** blows up, say

```
6$ echo $?
```

before doing anything else to see its exit status. If **myserver** doesn't blow up during the first 30 seconds, kill it with **control-c** and run it in the background with **nohup** (KP p. 33, 35) so that it will not die when you log out:

```
7$ nohup myserver &
8$ ps -f | more

9$ netstat -an -f inet -P tcp | awk '2 <= NR && NR <= 4 || $1 ˜ /\.10566$/'
TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
      *.10566              *.*                0      0 24576      0 LISTEN
```

Then on any Internet host (including **i5.nyu.edu** itself) you can run **mytelnet**, or simply

```
$ telnet i5.nyu.edu 10566
A                             You type this line.
a                             Your server types this line; telnet types the next.
Connection to i5.nyu.edu closed by foreign host.
```

myserver return's from **main** when it is done with its first and only client. If you want to kill the server before it has talked to a client, use **ps** and **kill -9**.

If you try to **rm** a program while it is running, you may get the **text file busy** error message. For example, you will get this message if you try to recompile a C program while it is running. Simply **kill -9** the process and then try to **rm** again. For a minute or two after the server terminates, the socket will remain in the **TIME_WAIT** state:

```
10$ netstat -an -f inet -P tcp | grep 10566
128.122.253.152.10566 128.122.253.152.52689 49152      0 49152      0 TIME_WAIT
```

Without the **SO_REUSEADDR** in the above line 26, the **bind** in line 31 will give you

```
Address already in use
```

if you try to run the server again while the socket is still in the **TIME_WAIT** state. With the **SO_REUSEADDR**, we can have two servers bound to the same port:

```
128.122.253.152.10566 128.122.253.152.52689 49152      0 49152      0 TIME_WAIT
      *.10566              *.*                0      0 49152      0 LISTEN
```

—On the Web at
http://i5.nyu.edu/˜mm64/x52.9544/src/myserver.pl

```
#!/bin/perl -T
#The same server, in Perl.
use Socket;

use FileHandle;
STDOUT->autoflush();

socket(S, AF_INET, SOCK_STREAM, 0) or die "$0: $!";

setsockopt(S, SOL_SOCKET, SO_REUSEADDR, 1) or die "$0: $!";
$reuse = getsockopt(S, SOL_SOCKET, SO_REUSEADDR);
if (!defined $reuse) {
    die "$0: $!"
}
print 'SO_REUSEADDR == ', SO_REUSEADDR, "\n",
    'Reuse option has been set to ', unpack('I', $reuse), ".\n";

```

```
18 bind(S, sockaddr_in(10566, INADDR_ANY)) or die "$0: $!";
19 state();
20
21 print 'SOMAXCONN == ', SOMAXCONN, "\n";
22 listen(S, SOMAXCONN) or die "$0: $!";
23 state();
24
25 $clientaddress = accept(CLIENT, S) or die "$0: $!";
26 CLIENT->autoflush();
27
28 ($port, $ip) = sockaddr_in($clientaddress);
29 print "I have accepted a client whose IP address is ",
30     inet_ntoa($ip), ".\n";
31 state();
32
33 $_ = <CLIENT>;
34 $_ =~ tr/A-Z/a-z/;   #Can remove the $_ =~
35 print CLIENT $_;     #Can remove the $_
36
37 shutdown(CLIENT, SHUT_RDWR) or die "$0: $!";
38 state();
39 exit 0;
40
41 sub state {
42     print 'State of socket is ',
43         `/bin/netstat -an -f inet -P tcp |\
44         awk 'NR > 4 && \$1 ~ /\.10566\$/ {print \$NF}' |\
45         tail -1`;
46 }
```

In Perl, run in *taint mode* with **-T** (O'Reilly Perl book, p. 356) to prevent outsiders from making mischief. See

```
11$ man -M /usr/perl5/5.6.1/man perlrun         Perl command line options, including -T
12$ man -M /usr/perl5/5.6.1/man perlsec         Perl security
```

The **'I'** in line 16 unpacks an unsigned integer.

```
13$ nohup myserver.pl > myserver.log 2>&1 &

14$ telnet i5.nyu.edu 10566
Hello
hello
Connection to i5.nyu.edu closed by foreign host.

15$ cat myserver.log
SO_REUSEADDR == 4
Reuse option has been set to 4.
State of socket is BOUND            line 19
SOMAXCONN == 5                      line 21
State of socket is LISTEN           line 23
I have accepted a client whose IP address is 128.122.253.152.
State of socket is ESTABLISHED      line 31
State of socket is TIME_WAIT        line 38
```

Serve more than one client simultaneously

No man can serve two masters.

—*Matthew 6:24*

To serve two or more clients, your server must **`read`** from and **`write`** to one of them while also **`accept`**'ing another one. The easiest way to do both, simultaneously, is to have your server give birth immediately after each **`accept`**'ance of a new client. The child will devote itself entirely to its dialog with the new client, and will **`exit`** when it is done talking to the client. The parent, meanwhile, will have no further relations to do with the new client. It will do nothing but **`accept`** additional clients. See Bach pp. 386−387; David Curry's O'Reilly book *Using C on the UNIX System* pp. 350−352.

In the following program, all i/o is performed with the file descriptor **`client`**. The other file descriptor, **`s`**, is only for **`accept`**'ing new clients. If you want to use the standard i/o library, it therefore makes sense to **`fdopen`** only the **`client`** file descriptor, not **`s`**. And since only the child, not the parent, will perform the i/o, you should **`fdopen client`** in the child. The parent should do nothing with the **`client`** file descriptor except **`close`** it immediately after the **`fork`**.

Instead of **`wait`**'ing for dead children in lines 104−118, the parent could be sensitive to the death-of-child signal. A death-of-child signal would awaken the parent from the **`accept`** in line 51. But then we would have to distinguish between the two things that could wake the parent from the **`accept`**: an acceptance of a client, or the death of a child. It seemed simpler to **`wait`**.

—On the Web at
`http://i5.nyu.edu/˜mm64/x52.9544/src/myserver2.c`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <ctype.h>
 4
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7 #include <netinet/in.h>
 8 #include <sys/wait.h>
 9
10 int main(int argc, char **argv)
11 {
12     const u_short port = 10566;  /* sin_port field of struct sockaddr_in */
13     int s = socket(AF_INET, SOCK_STREAM, 0); /* file descriptor to accept a client */
14     const int on = 1;
15     struct sockaddr_in myaddress;
16     struct sockaddr_in clientaddress;
17     size_t length = sizeof clientaddress;
18     int client;  /* file descriptor to talk to client */
19     char dotted[INET6_ADDRSTRLEN];          /* for inet_ntop */
20     pid_t pid;
21     char c;
22     char buffer[2];
23     int status;
24
25     if (s < 0) {
26         perror(argv[0]);
27         return 1;
28     }
29
30     if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR, &on, sizeof on) != 0) {
31         perror(argv[0]);
```

```
        return 2;
    }

    bzero((char *)&myaddress, sizeof myaddress);
    myaddress.sin_family = AF_INET;
    myaddress.sin_port = htons(port);
    myaddress.sin_addr.s_addr = INADDR_ANY;

    if (bind(s, (struct sockaddr *)&myaddress, sizeof myaddress) != 0) {
        perror(argv[0]);
        return 3;
    }

    if (listen(s, SOMAXCONN) != 0) {
        perror(argv[0]);
        return 4;
    }

    for (;;) {
        client = accept(s, (struct sockaddr *)&clientaddress, &length);
        if (client < 0) {
            perror(argv[0]);
            return 5;
        }

        if (inet_ntop(AF_INET, &clientaddress.sin_addr,
            dotted, sizeof dotted) == NULL) {
            perror(argv[0]);
            return 6;
        }

        fprintf(stderr, "I have accepted a client whose IP address is %s.\n",
            dotted);

        pid = fork();
        if (pid < 0) {
            perror(argv[0]);
            return 7;
        }

        if (pid == 0) {
            /* Arrive here if I am the child.  The service provided
            by this server is merely to input one character and then
            output it in lowercase, followed by a newline. */

            if (read(client, &c, 1) != 1) {
                perror(argv[0]);
                return 1;
            }

            buffer[0] = tolower(c);
            buffer[1] = ’\n’;

            if (write(client, buffer, sizeof buffer) != sizeof buffer) {
                    perror(argv[0]);
                    return 2;
                }

                if (shutdown(client, SHUT_RDWR) != 0) {
                    perror(argv[0]);
                    return 3;
                }

                return EXIT_SUCCESS;
            }

            /* Arrive here if I am the parent. */
            if (close(client) != 0) {
                perror(argv[0]);
                return 8;
            }

            /* Harvest any children that are ripe (i.e., zombies). */
            while ((pid = waitpid(-1, &status, WNOHANG)) > 0) {
                printf("child PID %d: ", pid);
                if (WIFEXITED(status)) {
                    printf("exit status was %d.\n", WEXITSTATUS(status));
                } else if (WIFSIGNALED(status)) {
                    printf("terminated by signal number %d.\n", WTERMSIG(status));
                } else if (WIFSTOPPED(status)) {
                    printf("stopped by signal number %d.\n", WSTOPSIG(status));
                } else {
                    fprintf(stderr, "%s: child %d came to unknown end.\n",
                        argv[0], pid);
                    return 9;
                }
            }
        }

        return EXIT_SUCCESS;
}
```

Run this server with **nohup** as shown above. Use **ps** and **kill -9** to kill it when you no longer want it to **accept** clients.

Instead of writing a server that gives birth to server-children, the superuser can have the **inetd** give birth to server-children. See **inetd**(8) and **inetd.conf**(4).

```
1$ ps -A -o user,ruser,pid,comm | awk 'NR == 1 || $1 != $2' | more
    USER    RUSER   PID COMMAND
```

—On the Web at
http://i5.nyu.edu/˜mm64/x52.9544/src/myserver2.pl

```
#!/bin/perl -T
#The same server, in Perl.
use Socket;
use FileHandle;
use POSIX;

socket(S, AF_INET, SOCK_STREAM, 0)         or die "$0: $!";
```

```
setsockopt(S, SOL_SOCKET, SO_REUSEADDR, 1) or die "$0: $!";
bind(S, sockaddr_in(10566, INADDR_ANY))    or die "$0: $!";
listen(S, SOMAXCONN)                       or die "$0: $!";

for (;;) {
    $clientaddress = accept(CLIENT, S) or die "$0: $!";

    ($port, $ip) = sockaddr_in($clientaddress);
    print "I have accepted a client whose IP address is ",
        inet_ntoa($ip), ".\n";

    CLIENT->autoflush();

    $pid = fork();
    defined $pid or die "$0: $!";

    if ($pid == 0) {
        #Arrive here if I am the child.
        $_ = <CLIENT>;
        tr/A-Z/a-z/;
        print CLIENT;
        shutdown(CLIENT, SHUT_RDWR) or die "$0: $!";
        exit 0;
    }

    #Arrive here if I am the parent.
    close(CLIENT) or die "$0: $!";

    while (waitpid(-1, WNOHANG)) {
        if (WIFEXITED($?)) {
            print "My child’s exit status was ",
                WEXITSTATUS($?), "\n";
        } elsif (WIFSIGNALED($?)) {
            print "My child was terminated by signal number ",
                WTERMSIG($?), ".\n";
        } elsif (WIFSTOPPED($?)) {
            print "My child was stopped by signal number ",
                WSTOPSIG($?), ".\n";
        } else {
            print STDERR "$0: couldn’t find out how child $pid ended up.\n";
        }
    }
}

exit 0;
```

▼ Homework 8.1: write a server that can serve two or more clients simultaneously

Do not attempt this assignment until we discuss it in class. Write a server named **`myserver`** that will simultaneously serve two or more clients running on any hosts as shown above. If **`myserver`** runs on the host **`i5.nyu.edu`**, it should bind itself to port number 10000 plus the last four digits of your social security number. **`nohup`** the server on the date when this assignment is due, and **`kill -9`** it one week later.

Hand in the **myserver.c**, the hostname of the host on which it is running (e.g., **i5.nyu.edu**), the host's IP address (e.g., **128.122.253.152**), and the port number (e.g., **10566**). Also put a note in your World Wide Web home page giving the hostname and IP address of the machine where server runs, the port number of server, Describe what it does and how to use it, i.e., what input, if any, the client must send through the socket.

myserver can provide each client any service you wish, but you must use **fscanf**, **fgetc**, or **getc** instead of **read**, and you must use **fprintf**, **fputc**, or **putc** instead of **write**. For example:

(1) **myserver** can merely output to each client a brief message such as **You are client number n.** In this case, the server performs output but no input.

(2) Write an interactive server (performing both input and output) providing the service shown in **$S45/prog.c** (on the web at **http://i5.nyu.edu/˜mm64/x52.9545/public_html/prog.c**)

(3) If **myserver** is running on **i5.nyu.edu**, it can output to each client a picture of the moon. Have the child **popen** the program **/home1/m/mm64/bin/moon**, and write a **while-getc** loop to input bytes from **moon** and output them one-by-one through the socket to the client. This will make the output of my **moon** program available to the whole Internet.

(4) Write an interactive server that will translate its input into Pig Latin.

Test your server with either **telnet** or **mytelnet**.

▲

A SOCK_DGRAM client

—On the Web at
http://i5.nyu.edu/˜mm64/x52.9547/src/udp.c

```
/*
Send one UDP datagram to port 7 of the host labinfu.unipv.it
*/

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <netdb.h>    /* for gethostbyname */

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/uio.h>
#include <netinet/in.h>

void handle_sigalrm(int s);

int main(int argc, char **argv)
{
    const int s = socket(AF_INET, SOCK_DGRAM, 0);
    struct hostent *entry;

    struct sockaddr_in address;
    size_t length = sizeof address;

    struct sockaddr_in client_address;
    socklen_t client_length = sizeof client_address;
    u_short client_port; /* ephemeral port number, assigned by operating system */

    char buffer[100];
    ssize_t n;   /* socket size */

    if (s < 0) {
        perror(argv[0]);
        return 1;
    }

    entry = gethostbyname("labinfu.unipv.it");
    if (entry == NULL) {
        fprintf(stderr, "%s: h_errno == %d\n", argv[0], h_errno);
        return 2;
    }

    bzero(&address, length);
    address.sin_family = AF_INET;
    address.sin_port = htons(7);
    address.sin_addr.s_addr = *(in_addr_t *)entry->h_addr_list[0];

    /* Send a datagram. */
    if (sendto(s, "Hello", 5, 0,
        (const struct sockaddr *)&address, sizeof address) < 0) {
        perror(argv[0]);
        return 3;
    }

    /* getsockname must come after sendto. */
    if (getsockname(s, (struct sockaddr *)&client_address, &client_length) != 0) {
        perror(argv[0]);
        return 4;
    }

    client_port = ntohs(client_address.sin_port);
    printf("My port number is %u.  (I am the client.)\n", client_port);

    sprintf(buffer, "/bin/netstat -a -f inet -P udp |"
        " awk ’2 <= NR && NR <= 4 || $1 ~ /\\.%d/’\n", client_port);
    system(buffer);

    /* Remain in recvfrom for no more than 30 seconds. */
    if (signal(SIGALRM, handle_sigalrm) == SIG_ERR) {
        perror(argv[0]);
        return 5;
    }
    alarm(30);

    n = recvfrom(s, buffer, sizeof buffer, 0, (struct sockaddr *)&address, &length);
    if (n < 0) {
        perror(argv[0]);
        return 6;
    }

    printf ("I received a datagram containing the %d bytes \"%.*s\"\n"
        "from port number %u of ",
        n, n, buffer, ntohs(address.sin_port));

    if (inet_ntop(AF_INET, &address.sin_addr, buffer, sizeof buffer) == NULL) {
        perror(argv[0]);
        return 7;
    }

    printf("%s.\n", buffer);
    return EXIT_SUCCESS;
}

void handle_sigalrm(int s)
{
    printf("The datagram I was waiting for never arrived.\n");
    exit(8);
}
```

```
1$ gcc -o ~/bin/udp udp.c -lsocket -lnsl
2$ ls -l ~/bin/udp
3$ udp
UDP: IPv4
   Local Address         Remote Address     State
-------------------- -------------------- -------
      *.48992                               Idle
My port number is 48992.  (I am the client.)
I received a datagram containing the 5 bytes "Hello"
from port number 7 of 193.204.35.58.
```

—On the Web at
`http://i5.nyu.edu/~mm64/x52.9547/src/udp.pl`

```
#!/bin/perl
#The same program, in perl.
use Socket;

$n = 5;   #number of bytes to send and receive
socket(S, AF_INET, SOCK_DGRAM, 0) or die "$0: $!";

$i = send(S, ’Hello’, 0, sockaddr_in(7, inet_aton(’labinfu.unipv.it’)));
defined $i or die "$0: $!";

if ($i != $n) {
    die "$0: only $i bytes were sent; should have been $n.\n";
}

$address = getsockname(S);
defined $address or die "$0: $!";

($port, $ip) = sockaddr_in($address);   #ephemeral port number
print "My port number is $port.  (I am the client.)\n";

print ‘/bin/netstat -a -f inet -P udp | awk ’2 <= NR && NR <= 4 || \$1 ~ /\\.$port/’‘;

$SIG{ALRM} = ’handle_sigalrm’;
alarm(15);

```

```
$address = recv(S, $buffer, 5, 0);
defined $address or die "$0: $!";
$i = length($buffer);

if ($i != $n) {
    die "$0: only $i bytes received; should have been $n.\n";
}

($port, $ip) = sockaddr_in($address);
print "I received a datagram containing the ", length($buffer),
    " bytes \"$buffer\"\n",
    "from port $port of ", inet_ntoa($ip), ".\n";
exit 0;

sub handle_sigalrm {
    print "The datagram I was waiting for never arrived.\n";
    exit 1;
}
```

```
4$ udp.pl
My port number is 48994.  (I am the client.)
UDP: IPv4
   Local Address         Remote Address     State
-------------------- -------------------- -------
      *.48994                               Idle
I received a datagram containing the 5 bytes "Hello"
from port 7 of 193.204.35.58.
```

▼ **Homework 8.2: send a datagram to Tokyo and Pisa**

Send an empty datagram to the `daytime` server at port 13/udp of `spider.let.uu.nl`. Does the server respond? If not, does the alarm wake up your client?

▲

▼ **Homework 8.3: how many bytes can a datagram hold?**

The above programs send and receive a five-byte datagram. Discover by trial and error the size of the largest possible datagram you can `sendto` and `recvfrom` `echo`. It will be a power of 2.

▲

□