Files and sockets

Here is a program, `d20` (for "decimal to octal"), that demonstrates reading from and writing to files, and using `fprintf` and `fscanf`. The program takes two arguments; the first is the name of an input file, from which it reads a sequence of integers, and the second is the name of an output file, into which it writes those integers in octal. The output file is created if needed.

d20.c

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main(int argc, char **argv)
{
        FILE *in, *out;
        int i;

        if (argc < 3) {
                fprintf(stderr, "Usage: %s IN OUT\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        in = fopen(argv[1], "r");
        if (!in) {
                perror(argv[1]);
                exit(EXIT_FAILURE);
        }

        out = fopen(argv[2], "w");
        if (!out) {
                perror(argv[2]);
                fclose(in);
                exit(EXIT_FAILURE);
        }

        while (fscanf(in, "%d", &i) == 1)
                fprintf(out, "0%o\n", i);
        if (ferror(in))
                perror("fscanf");

        fclose(in);
        fclose(out);
        return EXIT_SUCCESS;
}
```

Thus, with an input file such as:

in.txt

```
1
13
```

```
18
4
2
```

Then running the program as:

```
$ ./d20 in.txt out.txt
```

Produces no output, but creates this file:

out.txt

```
01
015
022
04
02
```

Here are two programs, a client and server for a simple echo protocol. The `client` program takes a hostname, a port, and a message as command-line arguments, connects to the host and port, sends the message, and then reads it back and prints the result.

client.c

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

#define BUF_SIZE 500

int main(int argc, char **argv)
{
        struct addrinfo hints, *result, *rp;
        int sfd, s;
        ssize_t len, nsent, ret;
        char buf[BUF_SIZE];

        if (argc != 4) {
                fprintf(stderr, "Usage: %s HOST PORT MSG\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        memset(&hints, 0, sizeof hints);
        hints.ai_socktype = SOCK_STREAM;

        s = getaddrinfo(argv[1], argv[2], &hints, &result);
```

```
        if (s != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
                exit(EXIT_FAILURE);
        }

        for (rp = result; rp; rp = rp->ai_next) {
                sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
                if (sfd == -1)
                        continue;
                if (connect(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
                        break;
                close(sfd);
        }

        freeaddrinfo(result);
        if (!rp) {
                fprintf(stderr, "Could not bind\n");
                exit(EXIT_FAILURE);
        }

        len = strlen(argv[3]);
        for (nsent = 0; nsent < len; nsent += ret) {
                ret = send(sfd, &argv[3][nsent], len - nsent, 0);
                if (ret <= 0) {
                        if (ret < 0)
                                perror("send");
                        break;
                }
        }

        shutdown(sfd, SHUT_WR);

        while ((ret = recv(sfd, buf, sizeof buf, 0)) > 0)
                fwrite(buf, ret, 1, stdout);
        if (ret < 0)
                perror("send");

        close(sfd);
        return EXIT_SUCCESS;
}
```

The server accepts one connection, reads all of the input, echoes it back, and then exits.

server.c

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
6  #include <sys/socket.h>
7  #include <netdb.h>

8  #define BUF_SIZE 500

9  int main(int argc, char **argv)
10 {
11         struct addrinfo hints, *result, *rp;
12         int sfd, sock, s;
13         ssize_t nread, nsent, ret;
14         char buf[BUF_SIZE];

15         if (argc != 2) {
16                 fprintf(stderr, "Usage: %s PORT\n", argv[0]);
17                 exit(EXIT_FAILURE);
18         }

19         memset(&hints, 0, sizeof hints);
20         hints.ai_socktype = SOCK_STREAM;
21         hints.ai_flags = AI_PASSIVE;

22         s = getaddrinfo(NULL, argv[1], &hints, &result);
23         if (s != 0) {
24                 fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
25                 exit(EXIT_FAILURE);
26         }

27         for (rp = result; rp; rp = rp->ai_next) {
28                 sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
29                 if (sfd == -1)
30                         continue;
31                 if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
32                         break;
33                 close(sfd);
34         }

35         freeaddrinfo(result);
36         if (!rp) {
37                 fprintf(stderr, "Could not bind\n");
38                 exit(EXIT_FAILURE);
39         }

40         if (listen(sfd, 10) == -1) {
41                 perror("listen");
42                 close(sfd);
43                 exit(EXIT_FAILURE);
44         }

45         sock = accept(sfd, NULL, NULL);
46         if (sock == -1) {
```

```
47                  perror("accept");
48                  close(sfd);
49                  exit(EXIT_FAILURE);
50          }

51          while ((nread = recv(sock, buf, sizeof buf, 0)) > 0) {
52                  for (nsent = 0; nsent < nread; nsent += ret) {
53                          ret = send(sock, &buf[nsent], nread - nsent, 0);
54                          if (ret <= 0) {
55                                  if (ret < 0)
56                                          perror("send");
57                                  break;
58                          }
59                  }
60          }

61          close(sock);
62          close(sfd);
63          return EXIT_SUCCESS;
64  }
```