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);
```

```
27          if (s != 0) {
28                  fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
29                  exit(EXIT_FAILURE);
30          }

32          for (rp = result; rp; rp = rp->ai_next) {
33                  sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
34                  if (sfd == -1)
35                          continue;
36                  if (connect(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
37                          break;
38                  close(sfd);
39          }

41          freeaddrinfo(result);
42          if (!rp) {
43                  fprintf(stderr, "Could not bind\n");
44                  exit(EXIT_FAILURE);
45          }

47          len = strlen(argv[3]);
48          for (nsent = 0; nsent < len; nsent += ret) {
49                  ret = send(sfd, &argv[3][nsent], len - nsent, 0);
50                  if (ret <= 0) {
51                          if (ret < 0)
52                                  perror("send");
53                          break;
54                  }
55          }

57          shutdown(sfd, SHUT_WR);

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

64          close(sfd);
65          return EXIT_SUCCESS;
66  }
```

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

server.c

```
1   #include <sys/types.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <unistd.h>
5   #include <string.h>
```

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

9  #define BUF_SIZE 500

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

18         if (argc != 2) {
19                 fprintf(stderr, "Usage: %s PORT\n", argv[0]);
20                 exit(EXIT_FAILURE);
21         }

23         memset(&hints, 0, sizeof hints);
24         hints.ai_socktype = SOCK_STREAM;
25         hints.ai_flags = AI_PASSIVE;

27         s = getaddrinfo(NULL, argv[1], &hints, &result);
28         if (s != 0) {
29                 fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
30                 exit(EXIT_FAILURE);
31         }

33         for (rp = result; rp; rp = rp->ai_next) {
34                 sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
35                 if (sfd == -1)
36                         continue;
37                 if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
38                         break;
39                 close(sfd);
40         }

42         freeaddrinfo(result);
43         if (!rp) {
44                 fprintf(stderr, "Could not bind\n");
45                 exit(EXIT_FAILURE);
46         }

48         if (listen(sfd, 10) == -1) {
49                 perror("listen");
50                 close(sfd);
51                 exit(EXIT_FAILURE);
52         }

54         sock = accept(sfd, NULL, NULL);
55         if (sock == -1) {
```

```
56                  perror("accept");
57                  close(sfd);
58                  exit(EXIT_FAILURE);
59          }

61          while ((nread = recv(sock, buf, sizeof buf, 0)) > 0) {
62                  for (nsent = 0; nsent < nread; nsent += ret) {
63                          ret = send(sock, &buf[nsent], nread - nsent, 0);
64                          if (ret <= 0) {
65                                  if (ret < 0)
66                                          perror("send");
67                                  break;
68                          }
69                  }
70          }

72          close(sock);
73          close(sfd);
74          return EXIT_SUCCESS;
75  }
```