```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

void
typefile (char *filename)
{
  int fd, nread;
  char buf[1024];

  fd = open (filename, O_RDONLY);
  if (fd == -1) {
    perror (filename);
    return;
  }

  while ((nread = read (fd, buf, sizeof (buf))) > 0)
    write (1, buf, nread);

  close (fd);
}

int
main (int argc, char **argv)
{
  int argno;
  for (argno = 1; argno < argc; argno++)
    typefile (argv[argno]);
  exit (0);
}
```

```c
/*
** showip.c -- show IP addresses for a host given on the command line
*/

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    struct addrinfo hints, *res, *p;
    int status;
    char ipstr[INET6_ADDRSTRLEN];

    if (argc != 2) {
        fprintf(stderr,"usage: showip hostname\n");
        return 1;
    }

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC; // AF_INET or AF_INET6 to force version
    hints.ai_socktype = SOCK_STREAM;

    if ((status = getaddrinfo(argv[1], NULL, &hints, &res)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(status));
        return 2;
    }

    printf("IP addresses for %s:\n\n", argv[1]);

    for(p = res;p != NULL; p = p->ai_next) {
        void *addr;
        char *ipver;

        // get the pointer to the address itself,
        // different fields in IPv4 and IPv6:
        if (p->ai_family == AF_INET) { // IPv4
            struct sockaddr_in *ipv4 = (struct sockaddr_in *)p->ai_addr;
            addr = &(ipv4->sin_addr);
            ipver = "IPv4";
        } else { // IPv6
            struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)p->ai_addr;
            addr = &(ipv6->sin6_addr);
            ipver = "IPv6";
        }

        // convert the IP to a string and print it:
        inet_ntop(p->ai_family, addr, ipstr, sizeof ipstr);
        printf(" %s: %s\n", ipver, ipstr);
    }

    freeaddrinfo(res); // free the linked list

    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

typedef enum op_t { ERROR, FETCH, STORE } op_t;
const char *file;
char *tmp_file;

void
do_fetch (int csock)
{
  int fd;
  int nread;
  char buf[1024];

  fd = open (file, O_RDONLY);
  if (fd < 0) {
    perror (file);
    return;
  }

  while ((nread = read (fd, buf, sizeof (buf))) > 0)
    write (csock, buf, nread);
}

void
do_store (int csock)
{
  int fd;
  int nread;
  char buf[1024];

  fd = open (tmp_file, O_CREAT|O_TRUNC|O_WRONLY, 0666);
  if (fd < 0) {
    perror (tmp_file);
    write (csock, "ERROR\n", 6);
    return;
  }

  while ((nread = read (csock, buf, sizeof (buf))) > 0)
    write (fd, buf, nread);
  fsync (fd);
  close (fd);
  rename (tmp_file, file);
  write (csock, "OK\n", 3);
}

int
fullread (int fd, char *buf, int len)
{
  while (len > 0) {
    int nread = read (fd, buf, len);
    if (nread < 0) {
      perror ("read");
      return -1;
    }
    if (nread == 0) {
      fprintf (stderr, "premature EOF\n");
      return -1;
    }
    len -= nread;
    buf += nread;
```

```c
  }
  return 0;
}

op_t
get_command (int csock)
{
  char cmd[7];
  if (fullread (csock, cmd, 6) < 0)
    return ERROR;
  cmd[6] = '\0';
  if (!strncmp (cmd, "fetch", 5))
    return FETCH;
  if (!strncmp (cmd, "store", 5))
    return STORE;
  return ERROR;
}

void
service_client (int csock)
{
  switch (get_command (csock)) {
  case FETCH:
    do_fetch (csock);
    break;
  case STORE:
    do_store (csock);
    break;
  default:
    fprintf (stderr, "bad command\n");
    break;
  }
}

void
server_loop (int lsock)
{
  struct sockaddr_in sin;
  socklen_t sinlen;
  int csock;

  if (listen (lsock, 5) < 0) {
    perror ("listen");
    exit (1);
  }

  for (;;) {
    bzero (&sin, sizeof (sin));
    sinlen = sizeof (sin);
    csock = accept (lsock, (struct sockaddr *) &sin, &sinlen);
    if (csock < 0) {
      perror ("accept");
      exit (1);
    }

    printf ("servicing client from IP address %s, TCP port %d\n",
            inet_ntoa (sin.sin_addr), ntohs (sin.sin_port));
    service_client (csock);
    close (csock);
  }
}

int
tcpsocket (char* port)
{
  int fd;
  struct addrinfo hints, *res, *rp;
  int n, err;
  int nport = atoi(port);

  if (nport < 1 || nport > 0xffff) {
    fprintf (stderr, "bad port number\n");
    exit (1);
  }

  memset(&hints, 0, sizeof(hints));
  hints.ai_family = AF_UNSPEC;
  hints.ai_socktype = SOCK_STREAM;
  hints.ai_flags = AI_PASSIVE;

  err = getaddrinfo(NULL, port, &hints, &res);
  if (err != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(err));
    exit(1);
  }

  /* getaddrinfo returns a list of struct addrinfo. Try each address
   * until we successfully bind. */
  for (rp = res; rp != NULL; rp = rp->ai_next) {
    fd = socket (res->ai_family, res->ai_socktype, res->ai_protocol);
    if (fd < 0) {
        perror ("socket");
        continue;
    }

    /* The following allows you to bind a TCP port when old connections
     * are still in TIME_WAIT.   */
    n = 1;
    setsockopt (fd, SOL_SOCKET, SO_REUSEADDR, (char *) &n, sizeof (n));

    if (bind (fd, rp->ai_addr, rp->ai_addrlen) == 0) {
      break;    /* Success */
    }
    close(fd);
  }

  if (rp == NULL) {
    fprintf(stderr, "Could not bind\n");
    exit(1);
  }
  freeaddrinfo(res);
  return fd;
}

int
main (int argc, char **argv)
{
  int lsock;

  if (argc != 3) {
    fprintf (stderr, "usage: %s port file\n", argv[0]);
    exit (1);
  }

  file = argv[2];
  tmp_file = malloc (strlen (file) + 1);
  sprintf (tmp_file, "%s~", file);

  lsock = tcpsocket ((argv[1]));
  server_loop (lsock);

  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int
udpsocket (int port)
{
  int fd;
  struct sockaddr_in sin;

  if (port < 1 || port > 0xffff) {
    fprintf (stderr, "bad port number\n");
    exit (1);
  }

  fd = socket (AF_INET, SOCK_DGRAM, 0);
  if (fd < 0) {
    perror ("socket");
    exit (1);
  }

  bzero (&sin, sizeof (sin));
  sin.sin_family = AF_INET;
  sin.sin_port = htons (port);
  sin.sin_addr.s_addr = htonl (INADDR_ANY);
  if (bind (fd, (struct sockaddr *) &sin, sizeof (sin)) < 0) {
    perror ("bind");
    exit (1);
  }

  return fd;
}

int
main (int argc, char **argv)
{
  int fd;
  struct sockaddr_in sin;
  socklen_t sinlen;
  char buf[512];
  int n;

  if (argc != 2) {
    fprintf (stderr, "usage: %s port\n", argv[0]);
    exit (1);
  }

  fd = udpsocket (atoi (argv[1]));
  for (;;) {
    bzero (&sin, sizeof (sin));
    sinlen = sizeof (sin);
    n = recvfrom (fd, buf, sizeof (buf), 0, (struct sockaddr *) &sin, &sinlen);
    if (n < 0) {
      perror ("recvfrom");
      exit (1);
    }
    fprintf (stderr, "got %d bytes from IP address %s, UDP port %d\n",
             n, inet_ntoa (sin.sin_addr), ntohs (sin.sin_port));
    sendto (fd, buf, n, 0, (struct sockaddr *) &sin, sizeof (sin));
  }
}
```