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

#define BUF_SIZ        4096  // read/write buffer size (in bytes)
#define LINGER_TIMEOUT   30  // length of time to wait for socket data transmission (in seconds)

int main (int argc, char ** argv) {
      struct hostent * he;          // contains information about the host
      struct in_addr addr;          // contains the actual address in network byte order
      struct sockaddr_in sin;
      int host_ip, sock_fd;
      int wr_bytes, wr_bytes_total; // number of bytes read from stdin
      int rd_bytes, rd_bytes_total; // number of bytes written to the socket

      sin.sin_family = AF_INET;

      // argc should be at least 3; if not, complain and abort
      if (argc < 3) {
            fprintf (stderr, "ERROR: Not enough parameters!\n");
            fprintf (stderr, "Syntax: tcp_send hostname port [ <input_stream ]\n");
            fprintf (stderr, "Aborting...\n");
            return -1;
      }

      // get hostname
      fprintf (stderr, "[INFO] Obtaining host name... ");
      if ((host_ip = inet_aton (argv[1], &sin.sin_addr)) == 0) {
            // perform a dns look up
            fprintf (stderr, "Performing DNS look up... ");
            if (!(he = gethostbyname (argv[1]))) {
```

```
            // if the look up fails, exit program
            fprintf (stderr, "FAILURE!\n");
            fprintf (stderr, "ERROR: Invalid host '%s'!\n", argv[1]);
            fprintf (stderr, "Aborting...\n");
            return -1;
      }

      // look up was successful; let's copy relevant info
      memcpy (&sin.sin_addr.s_addr, he->h_addr_list[0], sizeof sin.sin_addr.s_addr);
}
fprintf (stderr, "SUCCESS!\n");

// get port
fprintf (stderr, "[INFO] Parsing port number... ");
if (strtoll (argv[2], NULL, 0) == LLONG_MIN || strtoll (argv[2], NULL, 0) == LLONG_MAX) {
      // something went wrong in the conversion
      fprintf (stderr, "FAILURE!\n");
      fprintf (stderr, "ERROR: Invalid port '%s': %s\n", argv[2], strerror (errno));
      fprintf (stderr, "Aborting...\n");
      return -1;
} else {
      // we should be good to go (copying over the port number)
      sin.sin_port = htons(strtoll (argv[2], NULL, 0));
}
fprintf (stderr, "SUCCESS!\n");

// create socket
fprintf (stderr, "[INFO] Creating socket... ");
if ((sock_fd = socket (AF_INET, SOCK_STREAM, 0)) < 0) {
      fprintf (stderr, "FAILURE!\n");
      fprintf (stderr, "ERROR: socket() failure: %s\n", strerror (errno));
      fprintf (stderr, "Aborting...\n");
      return -1;
}
fprintf (stderr, "SUCCESS!\n");

// set up SO_LINGER structure
struct linger solin;
solin.l_onoff = 1;               // linger active
solin.l_linger = LINGER_TIMEOUT; // linger for LINGER_TIMEOUT seconds
```

```
// set socket option for SO_LINGER (wait for data to be sent after a shutdown)
fprintf (stderr, "[INFO] Preparing socket for lingering... ");
if (setsockopt (sock_fd, SOL_SOCKET, SO_LINGER, &solin, sizeof solin) < 0) {
      fprintf (stderr, "FAILURE!\n");
      fprintf (stderr, "ERROR: setsockopt() failure for SO_LINGER: %s\n", strerror (errno));
      fprintf (stderr, "Aborting...\n");
      return -1;
}
fprintf (stderr, "SUCCESS!\n");

// attempt connection
fprintf (stderr, "[INFO] Connecting to %s... ", inet_ntoa (sin.sin_addr));
if (connect (sock_fd, ((struct sockaddr *) &sin), sizeof sin) < 0) {
      fprintf (stderr, "FAILURE!\n");
      fprintf (stderr, "ERROR: connect() failure: %s\n", strerror (errno));
      fprintf (stderr, "Aborting...\n");
      return -1;
}
fprintf (stderr, "SUCCESS!\n");

// set up data buffer and pointers
char * buf_ptr, * buf_origin;
buf_ptr = buf_origin = (char *) malloc (BUF_SIZ);

// set up time structs for determining transfer rate
struct timeval start, end;
suseconds_t total;

// if successful, read in data from STDIN and then write them to the socket
fprintf (stderr, "[INFO] Now reading from stdin...\n");

// get time start
if (gettimeofday (&start, NULL) < 0) {
      fprintf (stderr, "ERROR: gettimeofday() failure: %s\n", strerror (errno));
      fprintf (stderr, "Aborting...\n");
      return -1;
}

// read from file and transmit data through socket
while ((rd_bytes = read (STDIN_FILENO, buf_ptr, BUF_SIZ)) > 0) {
      rd_bytes_total += rd_bytes;
```

```
		// attempt the write
		while (1) {
			if ((wr_bytes = write (sock_fd, buf_ptr, rd_bytes)) <= 0) {
				fprintf (stderr, "ERROR: write() failure: %s\n", strerror (errno));
				fprintf (stderr, "Aborting...\n");
				return -1;
			} else if (wr_bytes != rd_bytes) {
				wr_bytes_total += wr_bytes;
				buf_ptr += wr_bytes;
				rd_bytes -= wr_bytes;
			} else {
				wr_bytes_total += wr_bytes;
				break;
			}
		}

		// reset the buffer position in case of partial writes
		buf_ptr = buf_origin;
	}

	// get time end
	if (gettimeofday (&end, NULL) < 0) {
		fprintf (stderr, "ERROR: gettimeofday() failure: %s\n", strerror (errno));
		fprintf (stderr, "Aborting...\n");
		return -1;
	}

	// update total time by the difference
	total = ( (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec) );

	fprintf (stderr, "[INFO] EOF reached on stdin!\n");

	// check for read errors
	if (rd_bytes < 0) {
		fprintf (stderr, "ERROR: read() failure: %s\n", strerror (errno));
		fprintf (stderr, "Aborting...\n");
		return -1;
	}

	fprintf (stderr, "[INFO] Data transmission successful!\n");
```

```
	// data transmission statistics (MB/s is just bytes / microseconds)
	if (!(he = gethostbyaddr (&sin.sin_addr.s_addr, sizeof sin.sin_addr.s_addr, AF_INET))) {
		fprintf (stderr, "[INFO] Data transmission statistics for %s on port %s:\n", inet_ntoa
(sin.sin_addr), argv[2]);
	} else {
		fprintf (stderr, "[INFO] Data transmission statistics for %s (%s) on port %s:\n", inet_ntoa
(sin.sin_addr), he->h_name, argv[2]);
	}
	fprintf (stderr, "	>>> Total bytes written: %d bytes\n", wr_bytes_total);
	fprintf (stderr, "	>>> Total time elapsed: %f seconds\n", ((double) total) / 1000000);
	fprintf (stderr, "	>>> Transfer rate @ %f MB/s\n", (((double) wr_bytes_total) / ((double) total)));


	// we're all done; let's shutdown
	fprintf (stderr, "[INFO] Attempting to shutdown the connection... ");
	if (shutdown (sock_fd, SHUT_RDWR) < 0) {
		fprintf (stderr, "FAILURE!\n");
		fprintf (stderr, "ERROR: shutdown() failure: %s\n", strerror (errno));
		fprintf (stderr, "Aborting...\n");
		return -1;
	}
	fprintf (stderr, "SUCCESS!\n");

	return 0;
}
```

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

#define BUF_SIZ 4096  // read/write buffer size (in bytes)

int main (int argc, char ** argv) {
      struct hostent * he;                  // contains information about the host
      struct in_addr addr;                  // contains the actual address in network byte order
      struct sockaddr_in sin;
      int host_ip, sock_fd, sock_fd2, port;
      int wr_bytes, wr_bytes_total;         // number of bytes read from stdin
      int rd_bytes, rd_bytes_total;         // number of bytes written to the socket

      sin.sin_family = AF_INET;
      sin.sin_addr.s_addr = INADDR_ANY;

      // argc should be at least 2; if not, complain and abort
      if (argc < 2) {
            fprintf (stderr, "ERROR: Not enough parameters!\n");
            fprintf (stderr, "Syntax: tcp_recv port [ >output_stream ]\n");
            fprintf (stderr, "Aborting...\n");
            return -1;
      }

      // get port
      fprintf (stderr, "[INFO] Parsing port number... ");
      if (strtoll (argv[1], NULL, 0) == LLONG_MIN || strtoll (argv[1], NULL, 0) == LLONG_MAX) {
            // something went wrong in the conversion
            fprintf (stderr, "FAILURE!\n");
            fprintf (stderr, "ERROR: Invalid port '%s': %s\n", argv[1], strerror (errno));
```

```
        fprintf (stderr, "Aborting...\n");
        return -1;
    } else {
        // we should be good to go (copying over the port number)
        sin.sin_port = htons(port = strtoll (argv[1], NULL, 0));
    }
    fprintf (stderr, "SUCCESS!\n");

    // create the socket
    fprintf (stderr, "[INFO] Creating socket... ");
    if ((sock_fd = socket (AF_INET, SOCK_STREAM, 0)) < 0) {
        fprintf (stderr, "FAILURE!\n");
        fprintf (stderr, "ERROR: socket() failure: %s\n", strerror (errno));
        fprintf (stderr, "Aborting...\n");
        return -1;
    }
    fprintf (stderr, "SUCCESS!\n");

    // bind the socket
    fprintf (stderr, "[INFO] Binding socket to port %d... ", port);
    if (bind (sock_fd, (struct sockaddr *) &sin, sizeof sin) < 0) {
        fprintf (stderr, "FAILURE!\n");
        fprintf (stderr, "ERROR: bind() failure: %s\n", strerror (errno));
        fprintf (stderr, "Aborting...\n");
        return -1;
    }
    fprintf (stderr, "SUCCESS!\n");

    // listen on the port
    fprintf (stderr, "[INFO] Listening on port %d... ", port);
    if (listen (sock_fd, 64) < 0) {
        fprintf (stderr, "FAILURE!\n");
        fprintf (stderr, "ERROR: listen() failure: %s\n", strerror (errno));
        fprintf (stderr, "Aborting...\n");
        return -1;
    }
    fprintf (stderr, "SUCCESS!\n");

    int len;
    len = sizeof sin;
```

```
// await/accept connection
fprintf (stderr, "[INFO] Awaiting incoming connection... ");
if ((sock_fd2 = accept (sock_fd, (struct sockaddr *) &sin, &len)) < 0) {
      fprintf (stderr, "FAILURE!\n");
      fprintf (stderr, "ERROR: accept() failure: %s\n", strerror (errno));
      fprintf (stderr, "Aborting...\n");
      return -1;
}
fprintf (stderr, "SUCCESS!\n");

// set up data buffer and pointers
char * buf_ptr, * buf_origin;
buf_ptr = buf_origin = (char *) malloc (BUF_SIZ);

// set up time structs for determining transfer rate
struct timeval start, end;
suseconds_t total;

// if successful, read in data from STDIN and then write them to the socket
fprintf (stderr, "[INFO] Reading remote data transmission...\n");

// get time start
if (gettimeofday (&start, NULL) < 0) {
      fprintf (stderr, "ERROR: gettimeofday() failure: %s\n", strerror (errno));
      fprintf (stderr, "Aborting...\n");
      return -1;
}

// read from socket and output to file
while ((rd_bytes = read (sock_fd2, buf_ptr, BUF_SIZ)) > 0) {
      rd_bytes_total += rd_bytes;

      // attempt the write
      while (1) {
            if ((wr_bytes = write (STDOUT_FILENO, buf_ptr, rd_bytes)) <= 0) {
                  fprintf (stderr, "ERROR: write() failure: %s\n", strerror (errno));
                  fprintf (stderr, "Aborting...\n");
                  return -1;
            } else if (wr_bytes != rd_bytes) {
                  wr_bytes_total += wr_bytes;
                  buf_ptr += wr_bytes;
```

```
                        rd_bytes -= wr_bytes;
                  } else {
                        wr_bytes_total += wr_bytes;
                        break;
                  }
            }

            // reset the buffer position in case of partial writes
            buf_ptr = buf_origin;
      }

      // get time end
      if (gettimeofday (&end, NULL) < 0) {
            fprintf (stderr, "ERROR: gettimeofday() failure: %s\n", strerror (errno));
            fprintf (stderr, "Aborting...\n");
            return -1;
      }

      // update total time by the difference
      total = ( (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec - start.tv_usec) );

      fprintf (stderr, "[INFO] Remote data transmission ended!\n");

      // check for read errors
      if (rd_bytes < 0) {
            fprintf (stderr, "ERROR: read() failure: %s\n", strerror (errno));
            fprintf (stderr, "Aborting...\n");
            return -1;
      }

      fprintf (stderr, "[INFO] Data transmission successful!\n");

      // data transmission statistics (MB/s is just bytes / microseconds)
      if (!(he = gethostbyaddr ((char *) &sin.sin_addr.s_addr, sizeof sin.sin_addr.s_addr, AF_INET))) {
            fprintf (stderr, "[INFO] Data transmission statistics for %s on port %s:\n", inet_ntoa
(sin.sin_addr), argv[1]);
      } else {
            fprintf (stderr, "[INFO] Data transmission statistics for %s (%s) on port %s:\n", inet_ntoa
(sin.sin_addr), he->h_name, argv[1]);
      }
      fprintf (stderr, "     >>> Total bytes received: %d bytes\n", rd_bytes_total);
```

```
	fprintf (stderr, "     >>> Total time elapsed: %f seconds\n", ((double) total) / 1000000);
	fprintf (stderr, "     >>> Transfer rate @ %f MB/s\n", (((double) rd_bytes_total) / ((double) total)));


	// we're all done; let's shutdown
	fprintf (stderr, "[INFO] Attempting to shutdown the connection... ");
	if (shutdown (sock_fd, SHUT_RDWR) < 0) {
		fprintf (stderr, "FAILURE!\n");
		fprintf (stderr, "ERROR: shutdown() failure: %s\n", strerror (errno));
		fprintf (stderr, "Aborting...\n");
		return -1;
	}
	fprintf (stderr, "SUCCESS!\n");

	return 0;
}
```

// Sample Output