```
// TollServer.cpp - TOLL GATE SERVER
//
// MODULE INDEX
// NAME                 CONTENTS
// ErrSys               Process system error
// ErrApp               Process application error
// main                 Main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 01-10-04     JS      Original
//
//-----------------------------------------------------------------------------

#include <sys/types.h>          // System type declarations
#include <string.h>             // String manipulation functions
#include <signal.h>             // Signal codes
#include <unistd.h>             // Operating system standard functions
#include <termios.h>            // Serial port parameter structure
#include <fcntl.h>              // File control options
#include <errno.h>              // Operating system error codes
#include <sys/socket.h>         // Socket declarations
#include <netinet/in.h>         // Internet type conversion functions
#include <arpa/inet.h>          // More Internet type conversion functions
#include <sys/wait.h>           // Wait for process completion
#include <iostream>             // C++ input/output streams
#include <iomanip>              // C++ input/output manipulators
#include <list>                 // C++ lists
using namespace std;            // Expand standard namespace to global scope

//-----------------------------------------------------------------------------

// DEFINITIONS

const size_t            STOCK_CODE_LEN = 15;
                                    // Stock code length
const unsigned short    STOCK_PRICE_REQ = 1486;
                                    // Stock price request code
const unsigned short    STOCK_PRICE_RESP = 1487;
                                    // Stock price response code
const unsigned short    SERV_PORT = 1492;
                                    // Server port number

//-----------------------------------------------------------------------------

// SERVER REQUEST STRUCTURE

struct ServReq_t {
    unsigned short      servReqCode;            // Request code
    unsigned short      servReqBodyLen;         // Request body length
    char                servReqStockCode[STOCK_CODE_LEN+1]; // Stock code
};

//-----------------------------------------------------------------------------

// SERVER RESPONSE STRUCTURE

struct ServResp_t {
    unsigned short      servRespCode;           // Response code
    unsigned short      servRespBodyLen;        // Response body length
    unsigned long       servRespPrice;          // Stock price in cents
};

//-----------------------------------------------------------------------------

// SERVER COMMUNICATIONS FAULT EXCEPTION

class ServFault_c {
public:
    const char          *servFaultMsg;          // Server fault message
    int                 servFaultErrNo;         // Server fault error number
    ServFault_c (const char *msg)
        : servFaultMsg (msg), servFaultErrNo (errno) {}
                                                // Constructor
};

//-----------------------------------------------------------------------------

// PROCESS SYSTEM ERROR

void
ErrSys (
```

```
    const char          *msg)           // Error message
{
    int                 savedErrNo;     // Saved error number

    // Report the error

    savedErrNo = errno;
    cerr << "Error: " << msg << ": " << strerror (savedErrNo) << '\n';

    // Terminate the process

    exit (1);
}

//------------------------------------------------------------------------------

// PROCESS APPLICATION ERROR

void
ErrApp (
    const char          *msg)           // Error message
{
    // Report the error

    cerr << "Error: " << msg << '\n';

    // Terminate the process

    exit (1);
}

//------------------------------------------------------------------------------

// MAIN LINE

int
main ()
{
    int                 servSockFd;     // Server socket file descriptor
    sockaddr_in         servInAddr;     // Server's Internet address
    ServReq_t           servReq;        // Server request
    ServResp_t          servResp;       // Server request
    size_t              reqLen;         // Request length
    ssize_t             rdLen;          // Read length
    int                 cliSockFd;      // Client socket file descriptor
    sockaddr_in         cliInAddr;      // Client's Internet address
    socklen_t           cliInAddrLen;   // Client's Internet addr struct length

    // Create an endpoint for communication

    servSockFd = socket (PF_INET, SOCK_STREAM, 0);
    if (servSockFd == -1)
        ErrSys ("open server socket");

    // Bind the endpoint to the server port

    memset (&servInAddr, 0, sizeof(servInAddr));
    servInAddr.sin_family = AF_INET;
    servInAddr.sin_port = htons(SERV_PORT);
    servInAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind (servSockFd, reinterpret_cast<sockaddr*>(&servInAddr),
      sizeof(servInAddr)) == -1)
        ErrSys ("bind server socket");

    // Listen for connections

    if (listen (servSockFd, 5) == -1)
        ErrSys ("listen on server socket");

    // Loop processing consecutive connections to the server

    for (;;) {

        // Initialise the client socket file descriptor

        cliSockFd = -1;

        // Catch server communications faults

        try {

            // Accept a client connection
```

```
            cliInAddrLen = sizeof(cliInAddr);
            cliSockFd = accept (servSockFd,
                reinterpret_cast<sockaddr*>(&cliInAddr), &cliInAddrLen);
            if (cliSockFd == -1)
                throw ServFault_c ("connect to server");

            // Receive requests from the client until the connection
            // is broken.

            for (;;) {

                // Read a request from the client

                reqLen = 0;
                do {
                    errno = 0;
                    rdLen = read (
                        cliSockFd,
                        static_cast<char*>(static_cast<void*>(&servReq))
                                + reqLen,
                        sizeof(servReq) - reqLen
                    );
                    if (rdLen == 0)
                        goto ConnectionClosed;
                    if (
                        rdLen < 0 ||
                        rdLen > static_cast<ssize_t>(sizeof(servReq) - reqLen)
                    )
                        throw ServFault_c ("read failed");
                    reqLen += rdLen;
                } while (reqLen < sizeof(servReq));

                // Validate the client request

                errno = 0;
                if (ntohs(servReq.servReqCode) != STOCK_PRICE_REQ)
                    throw ServFault_c ("wrong request code");
                if (ntohs(servReq.servReqBodyLen)
                  != sizeof(servReq.servReqStockCode))
                    throw ServFault_c ("wrong request length");

                // Log the request

                cout << "Request for price of " << servReq.servReqStockCode
                    << '\n';

                // Send the response to the client

                servResp.servRespCode = htons(STOCK_PRICE_RESP);
                servResp.servRespBodyLen = htons(4);
                servResp.servRespPrice = lrand48() % 10000;
                if (write (cliSockFd, &servResp, sizeof(servResp))
                  != sizeof(servResp))
                    throw ServFault_c ("write to client");
            }
        }

        // Process server communications faults

        catch (ServFault_c servFault) {

            // Log the communications fault

            cout << "Warning: client fault: " << servFault.servFaultMsg;
            if (servFault.servFaultErrNo != 0)
                cout << ": " << strerror (servFault.servFaultErrNo);
            cout << '\n';
        }
    ConnectionClosed:

        // Close the client's socket file descriptor

        if (cliSockFd != -1) close (cliSockFd);
    }
}
```