```
// TollSensor.cpp - TOLL GATE SENSOR EMULATOR
//
// 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 <stdlib.h>             // Standard library
#include <iostream>             // C++ input/output streams
#include <list>                 // C++ lists
using namespace std;            // Expand standard namespace to global scope

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

// DEFINITIONS

const size_t            VEHICLE_ID_LEN = 16;
                                // Vehicle identifier length
const size_t            MAX_SENSOR_MSG_LEN = 32;
                                // Maximum sensor message length
const static char       SENSOR_DEV_NAME[] = "/dev/tty2a";
                                // Sensor network device name
const static unsigned char SENSOR_ADR_ARR[] = {0x04, 0x05, 0x08, 0x09};
                                // Sensor address array
const size_t            SENSOR_ADR_CNT = sizeof(SENSOR_ADR_ARR)
                                / sizeof(SENSOR_ADR_ARR[0]);
                                // Sensor address count
const unsigned char     POLLER_ADR = 0x00;
                                // Poller address

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

// SERIAL MESSAGE TYPES

const unsigned char     SENSOR_MT_POLL = 0x01;
                                // Poll command
const unsigned char     SENSOR_MT_VEHICLE_DETECTED = 0x02;
                                // Vehicle detected reply
const unsigned char     SENSOR_MT_NO_VEHICLE = 0x03;
                                // No vehicle reply

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

// VEHICLE IDENTIFIER STRUCTURE

struct VehicleId_t {
    unsigned char       vehicleIdBytes[VEHICLE_ID_LEN];
                                // Vehicle identifier bytes
};

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

// POLLING COMMUNICATIONS FAULT EXCEPTION

class PollFault_c {
public:
    bool                pollFaultResync;        // Resynchronisation needed
    const char          *pollFaultMsg;          // Polling fault message
    PollFault_c (bool resync, const char *msg)
        : pollFaultResync (resync), pollFaultMsg (msg) {}
                                                // 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                 serFd;          // Serial i/f file descriptor
    termios             serTermios;     // Serial i/f parameters
    unsigned char       pollBuf[MAX_SENSOR_MSG_LEN]; // Poll buffer
    size_t              pollLen;        // Poll message length
    unsigned char       respBuf[MAX_SENSOR_MSG_LEN]; // Response buffer
    size_t              respLen;        // Response length
    ssize_t             rdLen;          // Read length
    size_t              sensorInd;      // Sensor index
    unsigned char       lrc;            // LRC register
    size_t              i;              // General purpose index
    VehicleId_t         vehicleIdArr[SENSOR_ADR_CNT]; // Vehicles in the plaza
    long                r;              // Random number

    // Initialise the vehicle identifiers

    for (sensorInd = 0; sensorInd < SENSOR_ADR_CNT; sensorInd ++) {
        for (i = 0; i < VEHICLE_ID_LEN; i++) {
            vehicleIdArr[sensorInd].vehicleIdBytes[i] =
                static_cast<unsigned char>(lrand48());
        }
    }

    // Open the serial interface

    if ((serFd = open (SENSOR_DEV_NAME, O_RDWR)) == -1)
        ErrSys ("open serial port");

    // Configure the serial interface

    if (tcgetattr (serFd, &serTermios) == -1)
        ErrSys ("tcgetattr");
    serTermios.c_iflag = INPCK;
    serTermios.c_oflag = 0;
    serTermios.c_cflag = CS8 | CREAD | PARENB | PARODD | CLOCAL;
    serTermios.c_lflag = 0;
    serTermios.c_cc[VMIN] = 0;
    serTermios.c_cc[VTIME] = 2;
    cfsetispeed (&serTermios, B9600);
    cfsetospeed (&serTermios, B9600);
```

```
if (tcsetattr (serFd, TCSANOW, &serTermios) == -1)
    ErrSys ("tcsetattr");

// Main read loop

for (;;) {

    // Catch polling faults

    try {
        // Receive a poll

        pollLen = 0;
        do {
            rdLen=read (serFd,pollBuf+pollLen,MAX_SENSOR_MSG_LEN-pollLen);
            if (rdLen < 0 || rdLen > MAX_SENSOR_MSG_LEN - pollLen)
                throw PollFault_c (1, "read failed");
            if (rdLen == 0)
                pollLen = 0;
            else
                pollLen += rdLen;
            if (pollLen >= 4 && pollBuf[3] > MAX_SENSOR_MSG_LEN-5)
                throw PollFault_c (1, "message too long");
        } while (pollLen < 4 || pollLen < pollBuf[3]+5);

        // Validate the LRC of the received poll

        lrc = 0;
        for (i = 0; i < pollLen; i++) lrc ^= pollBuf[i];
        if (lrc != 0)
            throw PollFault_c (1, "LRC error");

        // Validate the source address of the poll

        if (pollBuf[1] != POLLER_ADR)
            throw PollFault_c (0, "wrong source address");

        // Validate the destination address

        sensorInd = 0;
        while (
            sensorInd < SENSOR_ADR_CNT &&
            SENSOR_ADR_ARR[sensorInd] != pollBuf[0]
        )
            sensorInd ++ ;
        if (sensorInd >= SENSOR_ADR_CNT)
            throw PollFault_c (0, "unrecognised destination address");

        // Validate the message type

        if (pollBuf[2] != SENSOR_MT_POLL)
            throw PollFault_c (0, "unrecognised message type");

        // Initialise the response buffer

        respLen = 0;
        respBuf[respLen++] = POLLER_ADR;
        respBuf[respLen++] = SENSOR_ADR_ARR[sensorInd];

        // Half the time, on the average, return no-response

        r = lrand48 () % 100;
        if (r < 50) {
            respBuf[respLen++] = SENSOR_MT_NO_VEHICLE;
            respBuf[respLen++] = 0; // Payload length
        }

        // The other half return a response

        else {

            // 1/10th of the time, generate a new vehicle identifier

            if (r < 60) {
                for (i = 0; i < VEHICLE_ID_LEN; i++)
                    vehicleIdArr[sensorInd].vehicleIdBytes[i] =
                        static_cast<unsigned char>(lrand48());
            }

            // Construct the response

            respBuf[respLen++] = SENSOR_MT_VEHICLE_DETECTED;
```

```
                respBuf[respLen++] = VEHICLE_ID_LEN;
                for (i = 0; i < VEHICLE_ID_LEN; i++)
                    respBuf[respLen++] =
                        vehicleIdArr[sensorInd].vehicleIdBytes[i];
            }

            // Append the LRC

            lrc = 0;
            for (i = 0; i < respLen; i++) lrc ^= respBuf[i];
            respBuf[respLen++] = lrc;

            // Send the response

            if (write (serFd, respBuf, respLen) != respLen)
                ErrSys ("write to serial port");
        }

        // Catch communications faults

        catch (PollFault_c pollFault) {
            cerr << "Warning: " << pollFault.pollFaultMsg << '\n';
        }
    }
}
```