```
 1: // $Id: sockets.h,v 1.3 2020-11-19 14:18:58-08 - - $
 2:
 3: #ifndef __SOCKET_H__
 4: #define __SOCKET_H__
 5:
 6: #include <cstring>
 7: #include <stdexcept>
 8: #include <string>
 9: #include <vector>
10: using namespace std;
11:
12: #include <arpa/inet.h>
13: #include <netdb.h>
14: #include <netinet/in.h>
15: #include <string>
16: #include <sys/socket.h>
17: #include <sys/types.h>
18: #include <unistd.h>
19:
20: //
21: // class base_socket:
22: // mostly protected and not used by applications
23: //
24:
25: class base_socket {
26:    private:
27:       static constexpr size_t MAXRECV = 0xFFFF;
28:       static constexpr int CLOSED_FD = -1;
29:       int socket_fd {CLOSED_FD};
30:       sockaddr_in socket_addr;
31:       base_socket (const base_socket&) = delete; // prevent copying
32:       base_socket& operator= (const base_socket&) = delete;
33:    protected:
34:       base_socket(); // only derived classes may construct
35:       ~base_socket();
36:       void close();
37:
38:       // Server initialization
39:       void create();
40:       void bind (const in_port_t port);
41:       void listen() const;
42:       void accept (base_socket&) const;
43:
44:       // Client initialization
45:       void connect (const string host, const in_port_t port);
46:
47:    public:
48:       // Data Transimission
49:       ssize_t send (const string&);
50:       ssize_t recv (string&);
51:       void set_non_blocking (const bool);
52:       friend string to_string (const base_socket& sock);
53:
54: };
55:
```

```
56:
57: //
58: // class accepted_socket
59: // used by server when a client connects
60: //
61:
62: class accepted_socket: public base_socket {
63:    public:
64:       accepted_socket() {}
65: };
66:
67: //
68: // class client_socket
69: // used by client application to connect to server
70: //
71:
72: class client_socket: public base_socket {
73:    public:
74:       client_socket (string host, in_port_t port);
75: };
76:
77: //
78: // class server_socket
79: // single use class by server application
80: //
81:
82: class server_socket: public base_socket {
83:    public:
84:       server_socket (in_port_t port);
85:       void accept (accepted_socket& sock) {
86:          base_socket::accept (sock);
87:       }
88: };
89:
```

```
 90:
 91: //
 92: // class socket_error
 93: // base class for throwing socket errors
 94: //
 95:
 96: class socket_error: public runtime_error {
 97:    public:
 98:       explicit socket_error (const string& what): runtime_error(what){}
 99: };
100:
101: //
102: // class socket_sys_error
103: // subclass to record status of extern int errno variable
104: //
105:
106: class socket_sys_error: public socket_error {
107:    public:
108:       int sys_errno;
109:       explicit socket_sys_error (const string& what):
110:                socket_error(what + ": " + strerror (errno)),
111:                sys_errno(errno) {}
112: };
113:
114: //
115: // class socket_h_error
116: // subclass to record status of extern int h_errno variable
117: //
118:
119: class socket_h_error: public socket_error {
120:    public:
121:       int host_errno;
122:       explicit socket_h_error (const string& what):
123:                socket_error(what + ": " + hstrerror (h_errno)),
124:                host_errno(h_errno) {}
125: };
126:
```

```
127:
128: //
129: // class hostinfo
130: // information about a host given hostname or IPv4 address
131: //
132:
133: class hostinfo {
134:    public:
135:       const string hostname;
136:       const vector<string> aliases;
137:       const vector<in_addr> addresses;
138:       hostinfo (); // localhost
139:       hostinfo (hostent*);
140:       hostinfo (const string& hostname);
141:       hostinfo (const in_addr& ipv4_addr);
142:       friend string to_string (const hostinfo&);
143: };
144:
145: string localhost();
146: string to_string (const in_addr& ipv4_addr);
147:
148: #endif
149:
```

```
 1: // $Id: logstream.h,v 1.5 2020-11-19 13:52:35-08 - - $
 2:
 3: //
 4: // class logstream
 5: // replacement for initial cout so that each call to a logstream
 6: // will prefix the line of output with an identification string
 7: // and a process id.  Template functions must be in header files
 8: // and the others are trivial.
 9: //
10:
11: #ifndef __LOGSTREAM_H__
12: #define __LOGSTREAM_H__
13:
14: #include <cassert>
15: #include <iostream>
16: #include <string>
17: #include <vector>
18: using namespace std;
19:
20: #include <sys/types.h>
21: #include <unistd.h>
22:
23: class logstream {
24:    private:
25:       ostream& out_;
26:       string execname_;
27:    public:
28:
29:       // Constructor may or may not have the execname available.
30:       logstream (ostream& out, const string& execname = ""):
31:                  out_ (out), execname_ (execname) {
32:       }
33:
34:       // First line of main should set execname if logstream is global.
35:       void execname (const string& name) { execname_ = name; }
36:       string execname() { return execname_; }
37:
38:       // First call should be the logstream, not cout.
39:       // Then forward result to the standard ostream.
40:       template <typename T>
41:       ostream& operator<< (const T& obj) {
42:          assert (execname_.size() > 0);
43:          out_ << execname_ << "(" << getpid() << "): " << obj;
44:          return out_;
45:       }
46:
47: };
48:
49: #endif
50:
```

```
// $Id: sockets.cpp,v 1.5 2020-11-19 14:18:58-08 - - $

#include <cerrno>
#include <cstring>
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

#include <fcntl.h>
#include <limits.h>

#include "sockets.h"

base_socket::base_socket() {
   memset (&socket_addr, 0, sizeof (socket_addr));
}

base_socket::~base_socket() {
   if (socket_fd != CLOSED_FD) close();
}

void base_socket::close() {
   int status = ::close (socket_fd);
   if (status < 0) throw socket_sys_error ("close("
                         + to_string(socket_fd) + ")");
   socket_fd = CLOSED_FD;
}

void base_socket::create() {
   socket_fd = ::socket (AF_INET, SOCK_STREAM, 0);
   if (socket_fd < 0) throw socket_sys_error ("socket");
   int on = 1;
   int status = ::setsockopt (socket_fd, SOL_SOCKET, SO_REUSEADDR,
                              &on, sizeof on);
   if (status < 0) throw socket_sys_error ("setsockopt");
}

void base_socket::bind (const in_port_t port) {
   socket_addr.sin_family = AF_INET;
   socket_addr.sin_addr.s_addr = INADDR_ANY;
   socket_addr.sin_port = htons (port);
   int status = ::bind (socket_fd,
                        reinterpret_cast<sockaddr*> (&socket_addr),
                        sizeof socket_addr);
   if (status < 0) throw socket_sys_error ("bind(" + to_string (port)
                                           + ")");
}

void base_socket::listen() const {
   int status = ::listen (socket_fd, SOMAXCONN);
   if (status < 0) throw socket_sys_error ("listen");
}

```

```
 55:
 56: void base_socket::accept (base_socket& new_socket) const {
 57:    int addr_length = sizeof new_socket.socket_addr;
 58:    new_socket.socket_fd = ::accept (socket_fd,
 59:                reinterpret_cast<sockaddr*> (&new_socket.socket_addr),
 60:                reinterpret_cast<socklen_t*> (&addr_length));
 61:    if (new_socket.socket_fd < 0) throw socket_sys_error ("accept");
 62: }
 63:
 64: ssize_t base_socket::send (const string& message) {
 65:    int nbytes = ::send (socket_fd, message.c_str(), message.size(),
 66:                         MSG_NOSIGNAL);
 67:    if (nbytes < 0) throw socket_sys_error ("send");
 68:    return nbytes;
 69: }
 70:
 71: ssize_t base_socket::recv (string& message) {
 72:    char buffer [MAXRECV + 1];
 73:    message = "";
 74:    memset (buffer, 0, MAXRECV + 1);
 75:    ssize_t nbytes = ::recv (socket_fd, buffer, MAXRECV, 0);
 76:    if (nbytes < 0) throw socket_sys_error ("recv");
 77:    if (nbytes > 0) message = buffer;
 78:    return nbytes;
 79: }
 80:
 81: void base_socket::connect (const string host, const in_port_t port) {
 82:    struct hostent *hostp = ::gethostbyname (host.c_str());
 83:    if (hostp == NULL) throw socket_h_error ("gethostbyname("
 84:                               + host + ")");
 85:    socket_addr.sin_family = AF_INET;
 86:    socket_addr.sin_port = htons (port);
 87:    socket_addr.sin_addr = *reinterpret_cast<in_addr*> (hostp->h_addr);
 88:    int status = ::connect (socket_fd,
 89:                reinterpret_cast<sockaddr*> (&socket_addr),
 90:                sizeof (socket_addr));
 91:    if (status < 0) throw socket_sys_error ("connect(" + host + ":"
 92:                           + to_string (port) + ")");
 93: }
 94:
 95: void base_socket::set_non_blocking (const bool blocking) {
 96:    int opts = ::fcntl (socket_fd, F_GETFL);
 97:    if (opts < 0) throw socket_sys_error ("fcntl");
 98:    if (blocking) opts |= O_NONBLOCK;
 99:             else opts &= ~ O_NONBLOCK;
100:    opts = ::fcntl (socket_fd, F_SETFL, opts);
101:    if (opts < 0) throw socket_sys_error ("fcntl");
102: }
103:
```

```
104:
105: client_socket::client_socket (string host, in_port_t port) {
106:    base_socket::create();
107:    base_socket::connect (host, port);
108: }
109:
110: server_socket::server_socket (in_port_t port) {
111:    base_socket::create();
112:    base_socket::bind (port);
113:    base_socket::listen();
114: }
115:
116: string to_string (const hostinfo& info) {
117:    return info.hostname + " (" + to_string (info.addresses[0]) + ")";
118: }
119:
120: string to_string (const in_addr& ipv4_addr) {
121:    char buffer[INET_ADDRSTRLEN];
122:    const char *result = ::inet_ntop (AF_INET, &ipv4_addr,
123:                                      buffer, sizeof buffer);
124:    if (result == NULL) throw socket_sys_error ("inet_ntop");
125:    return result;
126: }
127:
128: string to_string (const base_socket& sock) {
129:    hostinfo info (sock.socket_addr.sin_addr);
130:    return info.hostname + " (" + to_string (info.addresses[0])
131:           + ") port " + to_string (ntohs (sock.socket_addr.sin_port));
132: }
133:
```

```
134:
135: string init_hostname (hostent* host) {
136:    if (host == nullptr) throw socket_h_error ("gethostbyname");
137:    return host->h_name;
138: }
139:
140: vector<string> init_aliases (hostent* host) {
141:    if (host == nullptr) throw socket_h_error ("gethostbyname");
142:    vector<string> init_aliases;
143:    for (char** alias = host->h_aliases; *alias != nullptr; ++alias) {
144:       init_aliases.push_back (*alias);
145:    }
146:    return init_aliases;
147: }
148:
149: vector<in_addr> init_addresses (hostent* host) {
150:    vector<in_addr> init_addresses;
151:    if (host == nullptr) throw socket_h_error ("gethostbyname");
152:    for (in_addr** addr =
153:                   reinterpret_cast<in_addr**> (host->h_addr_list);
154:         *addr != nullptr; ++addr) {
155:       init_addresses.push_back (**addr);
156:    }
157:    return init_addresses;
158: }
159:
160: hostinfo::hostinfo (hostent* host):
161:    hostname (init_hostname (host)),
162:    aliases (init_aliases (host)),
163:    addresses (init_addresses (host)) {
164: }
165:
166: hostinfo::hostinfo(): hostinfo (localhost()) {
167: }
168:
169: hostinfo::hostinfo (const string& hostname_):
170:           hostinfo (::gethostbyname (hostname_.c_str())) {
171: }
172:
173: hostinfo::hostinfo (const in_addr& ipv4_addr):
174:           hostinfo (::gethostbyaddr (&ipv4_addr, sizeof ipv4_addr,
175:                                      AF_INET)) {
176: }
177:
178: string localhost() {
179:    char hostname[HOST_NAME_MAX] {};
180:    int rc = gethostname (hostname, sizeof hostname);
181:    if (rc < 0) throw socket_sys_error ("gethostname");
182:    return hostname;
183: }
184:
```

```
 1: // $Id: client.cpp,v 1.6 2014-05-23 13:25:17-07 - - $
 2:
 3: #include <iostream>
 4: #include <sstream>
 5: #include <string>
 6: #include <vector>
 7: using namespace std;
 8:
 9: #include <libgen.h>
10: #include <sys/types.h>
11: #include <unistd.h>
12:
13: #include "logstream.h"
14: #include "sockets.h"
15:
16: string progname;
17:
18: int main (int argc, char** argv) {
19:    logstream clog (cout, basename (argv[0]));
20:    vector<string> args (&argv[1], &argv[argc]);
21:    string host = args.size() < 1 ? "localhost" : args[0];
22:    in_port_t port = args.size() < 2 ? 50000 : stoi (args[1]);
23:    clog << to_string (hostinfo()) << endl;
24:    try {
25:       clog << "connecting to " << host << " port " << port << endl;
26:       client_socket server (host, port);
27:       clog << "connected to " << to_string (server) << endl;
28:       for (int count = 0; count < 4; ++count) {
29:          string reply;
30:          try {
31:             ostringstream message;
32:             message << "Message " << count << " from client "
33:                     << getpid();
34:             clog << "to server: \"" << message.str() << "\"" << endl;
35:             server.send (message.str());
36:             server.recv (reply);
37:          }catch (socket_error& error) {
38:             clog << error.what() << endl;
39:          }
40:          clog << "from server: \"" << reply << "\"" << endl;
41:       }
42:    }catch (socket_error& error) {
43:       clog << error.what() << endl;
44:    }
45:    return 0;
46: }
```

```
 1: //$Id: server.cpp,v 1.9 2014-05-23 13:25:17-07 - - $
 2:
 3: #include <iostream>
 4: #include <string>
 5: #include <vector>
 6: using namespace std;
 7:
 8: #include <libgen.h>
 9:
10: #include "logstream.h"
11: #include "sockets.h"
12:
13: int main (int argc, char** argv) {
14:    logstream clog (cout, basename (argv[0]));
15:    vector<string> args (&argv[1], &argv[argc]);
16:    in_port_t port = args.size() < 1 ? 50000 : stoi (args[0]);
17:    try {
18:       if (port <= IPPORT_USERRESERVED) {
19:          throw socket_error ("unprivileged server port ("
20:                + to_string (port) + ") <= IPPORT_USERRESERVED ("
21:                + to_string (IPPORT_USERRESERVED) + ")");
22:       }
23:       // Create the socket
24:       server_socket listener (port);
25:       for (;;) {
26:          clog << to_string (hostinfo())
27:               << " accepting port " << to_string (port) << endl;
28:          accepted_socket client_sock;
29:          listener.accept (client_sock);
30:          clog << "accepted " << to_string (client_sock) << endl;
31:          try {
32:             for (;;) {
33:                string data;
34:                ssize_t nbytes = client_sock.recv (data);
35:                if (nbytes == 0) break;
36:                clog << "received \"" << data << "\"" << endl;
37:                client_sock.send (data);
38:             }
39:          }catch (socket_error& error) {
40:             clog << error.what() << endl;
41:          }
42:          clog << "client is gone" << endl;
43:       }
44:    }catch (socket_error& error) {
45:       clog << error.what() << endl;
46:    }
47:    return 0;
48: }
```

```
# $Id: Makefile,v 1.20 2020-11-13 23:41:20-08 - - $

GPPWARN    = -Wall -Wextra -Werror -Wpedantic -Wshadow -Wold-style-cast
GPPOPTS    = ${GPPWARN} -fdiagnostics-color=never
GPP        = g++ -std=gnu++17 -g -O0 ${GPPOPTS}

DEPFILE    = Makefile.dep
HEADERS    = sockets.h logstream.h
CPPSRCS    = sockets.cpp client.cpp server.cpp
CLIENTOBJS = client.o sockets.o
SERVEROBJS = server.o sockets.o
OBJECTS    = ${CLIENTOBJS} ${SERVEROBJS}
EXECBINS   = client server
LISTING    = Listing.ps
SOURCES    = ${HEADERS} ${CPPSRCS} Makefile

all: ${DEPFILE} ${EXECBINS}

client: ${CLIENTOBJS}
        ${GPP} -o $@ ${CLIENTOBJS}

server: ${SEROBJS}
        ${GPP} -o $@ ${SEROBJS}

%.o: %.cpp
        - cpplint.py.perl $<
        ${GPP} -c $<

ci:
        - checksource ${SOURCES}
        - cid -is ${SOURCES}

lis: all ${SOURCES} ${DEPFILE}
        mkpspdf ${LISTING} ${SOURCES} ${DEPFILE}

clean:
        - rm ${LISTING} ${LISTING:.ps=.pdf} ${OBJECTS}

spotless: clean
        - rm ${EXECBINS}

dep:
        - rm ${DEPFILE}
        make --no-print-directory ${DEPFILE}

${DEPFILE}:
        ${GPP} -MM ${CPPSRCS} >${DEPFILE}

again: ${SOURCES}
        make --no-print-directory spotless ci all lis

include ${DEPFILE}
```

```
1: sockets.o: sockets.cpp sockets.h
2: client.o: client.cpp logstream.h sockets.h
3: server.o: server.cpp logstream.h sockets.h
```