

```
1: // $Id: sockets.h,v 1.2 2014-05-23 22:17:25-07 - - $
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 <sys/wait.h>
19: #include <unistd.h>
20:
21: //
22: // class base_socket:
23: // mostly protected and not used by applications
24: //
25:
26: class base_socket {
27:     private:
28:         static constexpr size_t MAXRECV = 0xFFFF;
29:         static constexpr int CLOSED_FD = -1;
30:         int socket_fd {CLOSED_FD};
31:         sockaddr_in socket_addr;
32:         base_socket (const base_socket&) = delete; // prevent copying
33:         base_socket& operator= (const base_socket&) = delete;
34:     protected:
35:         base_socket(); // only derived classes may construct
36:         ~base_socket();
37:         // server_socket initialization
38:         void create();
39:         void bind (const in_port_t port);
40:         void listen() const;
41:         void accept (base_socket&) const;
42:         // client_socket initialization
43:         void connect (const string host, const in_port_t port);
44:         // accepted_socket initialization
45:         string to_string_socket_fd() { return to_string (socket_fd); }
46:         void set_socket_fd (int fd);
47:     public:
48:         void close();
49:         ssize_t send (const void* buffer, size_t bufsize);
50:         ssize_t recv (void* buffer, size_t bufsize);
51:         void set_non_blocking (const bool);
52:         friend string to_string (const base_socket& sock);
53: };
54:
```

```
55:
56: //
57: // class accepted_socket
58: // used by server when a client connects
59: //
60:
61: class accepted_socket: public base_socket {
62:     public:
63:         accepted_socket() {}
64:         accepted_socket(int fd) { set_socket_fd (fd); }
65:         string to_string_socket_fd() {
66:             return base_socket::to_string_socket_fd();
67:         }
68: };
69:
70: //
71: // class client_socket
72: // used by client application to connect to server
73: //
74:
75: class client_socket: public base_socket {
76:     public:
77:         client_socket (string host, in_port_t port);
78: };
79:
80: //
81: // class server_socket
82: // single use class by server application
83: //
84:
85: class server_socket: public base_socket {
86:     public:
87:         server_socket (in_port_t port);
88:         void accept (accepted_socket& sock) {
89:             base_socket::accept (sock);
90:         }
91: };
92:
```

```
93:
94: //
95: // class socket_error
96: // base class for throwing socket errors
97: //
98:
99: class socket_error: public runtime_error {
100: public:
101:     explicit socket_error (const string& what): runtime_error(what){}
102: };
103:
104: //
105: // class socket_sys_error
106: // subclass to record status of extern int errno variable
107: //
108:
109: class socket_sys_error: public socket_error {
110: public:
111:     int sys_errno;
112:     explicit socket_sys_error (const string& what):
113:         socket_error(what + ": " + strerror (errno)),
114:         sys_errno(errno) {}
115: };
116:
117: //
118: // class socket_h_error
119: // subclass to record status of extern int h_errno variable
120: //
121:
122: class socket_h_error: public socket_error {
123: public:
124:     int host_errno;
125:     explicit socket_h_error (const string& what):
126:         socket_error(what + ": " + hstrerror (h_errno)),
127:         host_errno(h_errno) {}
128: };
129:
```

```
130:
131: //
132: // class hostinfo
133: // information about a host given hostname or IPv4 address
134: //
135:
136: class hostinfo {
137:     public:
138:         const string hostname;
139:         const vector<string> aliases;
140:         const vector<in_addr> addresses;
141:         hostinfo (); // localhost
142:         hostinfo (hostent*);
143:         hostinfo (const string& hostname);
144:         hostinfo (const in_addr& ipv4_addr);
145:         friend string to_string (const hostinfo&);
146: };
147:
148: string localhost();
149: string to_string (const in_addr& ipv4_addr);
150:
151: #endif
152:
```

```
1: // $Id: signal_action.h,v 1.5 2014-05-27 17:16:42-07 - - $
2:
3: #ifndef __SIGNAL_ACTION_H__
4: #define __SIGNAL_ACTION_H__
5:
6: #include <stdexcept>
7: using namespace std;
8:
9: #include <signal.h>
10:
11: class cix_exit: public exception{};
12:
13: class signal_action {
14:     private:
15:         struct sigaction action;
16:     public:
17:         signal_action (int signal, void (*handler) (int));
18: };
19:
20: class signal_error: runtime_error {
21:     public:
22:         int signal;
23:         explicit signal_error (int signal);
24: };
25:
26: #endif
27:
```

[illegible]

```
1: // $Id: logstream.h,v 1.3 2014-05-30 12:47:58-07 - - $
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 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:
```

```
1: // $Id: sockets.cpp,v 1.6 2014-07-25 11:54:44-07 - - $
2:
3: #include <cerrno>
4: #include <cstring>
5: #include <iostream>
6: #include <sstream>
7: #include <string>
8: using namespace std;
9:
10: #include <fcntl.h>
11: #include <limits.h>
12:
13: #include "sockets.h"
14:
15: base_socket::base_socket() {
16:     memset (&socket_addr, 0, sizeof (socket_addr));
17: }
18:
19: base_socket::~~base_socket() {
20:     if (socket_fd != CLOSED_FD) close();
21: }
22:
23: void base_socket::close() {
24:     int status = ::close (socket_fd);
25:     if (status < 0) throw socket_sys_error ("close("
26:                                     + to_string(socket_fd) + ")");
27:     socket_fd = CLOSED_FD;
28: }
29:
30: void base_socket::create() {
31:     socket_fd = ::socket (AF_INET, SOCK_STREAM, 0);
32:     if (socket_fd < 0) throw socket_sys_error ("socket");
33:     int on = 1;
34:     int status = ::setsockopt (socket_fd, SOL_SOCKET, SO_REUSEADDR,
35:                               &on, sizeof on);
36:     if (status < 0) throw socket_sys_error ("setsockopt");
37: }
38:
39: void base_socket::bind (const in_port_t port) {
40:     socket_addr.sin_family = AF_INET;
41:     socket_addr.sin_addr.s_addr = INADDR_ANY;
42:     socket_addr.sin_port = htons (port);
43:     int status = ::bind (socket_fd,
44:                         reinterpret_cast<sockaddr*> (&socket_addr),
45:                         sizeof socket_addr);
46:     if (status < 0) throw socket_sys_error ("bind(" + to_string (port)
47:                                     + ")");
48: }
49:
50: void base_socket::listen() const {
51:     int status = ::listen (socket_fd, SOMAXCONN);
52:     if (status < 0) throw socket_sys_error ("listen");
53: }
54:
```



```
55:
56: void base_socket::accept (base_socket& socket) const {
57:     int addr_length = sizeof socket.socket_addr;
58:     socket.socket_fd = ::accept (socket_fd,
59:                                 reinterpret_cast<sockaddr*> (&socket.socket_addr),
60:                                 reinterpret_cast<socklen_t*> (&addr_length));
61:     if (socket.socket_fd < 0) throw socket_sys_error ("accept");
62: }
63:
64: ssize_t base_socket::send (const void* buffer, size_t bufsize) {
65:     int nbytes = ::send (socket_fd, buffer, bufsize, MSG_NOSIGNAL);
66:     if (nbytes < 0) throw socket_sys_error ("send");
67:     return nbytes;
68: }
69:
70: ssize_t base_socket::recv (void* buffer, size_t bufsize) {
71:     memset (buffer, 0, bufsize);
72:     ssize_t nbytes = ::recv (socket_fd, buffer, bufsize, 0);
73:     if (nbytes < 0) throw socket_sys_error ("recv");
74:     return nbytes;
75: }
76:
77: void base_socket::connect (const string host, const in_port_t port) {
78:     struct hostent *hostp = ::gethostbyname (host.c_str());
79:     if (hostp == NULL) throw socket_h_error ("gethostbyname("
80:                                             + host + ")");
81:     socket_addr.sin_family = AF_INET;
82:     socket_addr.sin_port = htons (port);
83:     socket_addr.sin_addr = *reinterpret_cast<in_addr*> (hostp->h_addr);
84:     int status = ::connect (socket_fd,
85:                             reinterpret_cast<sockaddr*> (&socket_addr),
86:                             sizeof (socket_addr));
87:     if (status < 0) throw socket_sys_error ("connect(" + host + ":"
88:                                             + to_string (port) + ")");
89: }
90:
91: void base_socket::set_socket_fd (int fd) {
92:     socklen_t addrlen = sizeof socket_addr;
93:     int rc = getpeername (fd, reinterpret_cast<sockaddr*> (&socket_addr),
94:                           &addrlen);
95:     if (rc < 0) throw socket_sys_error ("set_socket_fd("
96:                                         + to_string (fd) + "): getpeername");
97:     socket_fd = fd;
98:     if (socket_addr.sin_family != AF_INET)
99:         throw socket_error ("address not AF_INET");
100: }
101:
102: void base_socket::set_non_blocking (const bool blocking) {
103:     int opts = ::fcntl (socket_fd, F_GETFL);
104:     if (opts < 0) throw socket_sys_error ("fcntl");
105:     if (blocking) opts |= O_NONBLOCK;
106:     else opts &= compl O_NONBLOCK;
107:     opts = ::fcntl (socket_fd, F_SETFL, opts);
108:     if (opts < 0) throw socket_sys_error ("fcntl");
109: }
110:
```

```
111:
112: client_socket::client_socket (string host, in_port_t port) {
113:     base_socket::create();
114:     base_socket::connect (host, port);
115: }
116:
117: server_socket::server_socket (in_port_t port) {
118:     base_socket::create();
119:     base_socket::bind (port);
120:     base_socket::listen();
121: }
122:
123: string to_string (const hostinfo& info) {
124:     return info.hostname + " (" + to_string (info.addresses[0]) + ")";
125: }
126:
127: string to_string (const in_addr& ipv4_addr) {
128:     char buffer[INET_ADDRSTRLEN];
129:     const char *result = ::inet_ntop (AF_INET, &ipv4_addr,
130:                                       buffer, sizeof buffer);
131:     if (result == NULL) throw socket_sys_error ("inet_ntop");
132:     return result;
133: }
134:
135: string to_string (const base_socket& sock) {
136:     hostinfo info (sock.socket_addr.sin_addr);
137:     return info.hostname + " (" + to_string (info.addresses[0])
138:           + ") port " + to_string (ntohs (sock.socket_addr.sin_port));
139: }
140:
```

```
141:
142: string init_hostname (hostent* host) {
143:     if (host == nullptr) throw socket_h_error ("gethostbyname");
144:     return host->h_name;
145: }
146:
147: vector<string> init_aliases (hostent* host) {
148:     if (host == nullptr) throw socket_h_error ("gethostbyname");
149:     vector<string> init_aliases;
150:     for (char** alias = host->h_aliases; *alias != nullptr; ++alias) {
151:         init_aliases.push_back (*alias);
152:     }
153:     return init_aliases;
154: }
155:
156: vector<in_addr> init_addresses (hostent* host) {
157:     vector<in_addr> init_addresses;
158:     if (host == nullptr) throw socket_h_error ("gethostbyname");
159:     for (in_addr** addr =
160:         reinterpret_cast<in_addr**> (host->h_addr_list);
161:         *addr != nullptr; ++addr) {
162:         init_addresses.push_back (**addr);
163:     }
164:     return init_addresses;
165: }
166:
167: hostinfo::hostinfo (hostent* host):
168:     hostname (init_hostname (host)),
169:     aliases (init_aliases (host)),
170:     addresses (init_addresses (host)) {
171: }
172:
173: hostinfo::hostinfo(): hostinfo (localhost()) {
174: }
175:
176: hostinfo::hostinfo (const string& hostname):
177:     hostinfo (::gethostbyname (hostname.c_str())) {
178: }
179:
180: hostinfo::hostinfo (const in_addr& ipv4_addr):
181:     hostinfo (::gethostbyaddr (&ipv4_addr, sizeof ipv4_addr,
182:                             AF_INET)) {
183: }
184:
185: string localhost() {
186:     char hostname[HOST_NAME_MAX] {};
187:     int rc = gethostname (hostname, sizeof hostname);
188:     if (rc < 0) throw socket_sys_error ("gethostname");
189:     return hostname;
190: }
191:
```

```
1: // $Id: signal_action.cpp,v 1.6 2014-05-27 17:16:42-07 - - $
2:
3: #include <cstring>
4: #include <string>
5: #include <unordered_map>
6: using namespace std;
7:
8: #include "signal_action.h"
9:
10: signal_action::signal_action (int signal, void (*handler) (int)) {
11:     action.sa_handler = handler;
12:     sigfillset (&action.sa_mask);
13:     action.sa_flags = 0;
14:     int rc = sigaction (signal, &action, nullptr);
15:     if (rc < 0) throw signal_error (signal);
16: }
17:
18: signal_error::signal_error (int signal):
19:     runtime_error (string ("signal_error(")
20:                     + strsignal (signal) + ")"),
21:     signal(signal) {}
22:
```

```
1: // $Id: cix_protocol.cpp,v 1.7 2014-07-24 20:24:51-07 - - $
2:
3: #include <unordered_map>
4: #include <string>
5: using namespace std;
6:
7: #include "cix_protocol.h"
8:
9: const unordered_map<int,string> cix_command_map {
10:     {int (CIX_ERROR), "CIX_ERROR"},
11:     {int (CIX_EXIT ), "CIX_EXIT" },
12:     {int (CIX_GET  ), "CIX_GET"  },
13:     {int (CIX_HELP ), "CIX_HELP" },
14:     {int (CIX_LS   ), "CIX_LS"   },
15:     {int (CIX_PUT  ), "CIX_PUT"  },
16:     {int (CIX_RM   ), "CIX_RM"   },
17:     {int (CIX_FILE ), "CIX_FILE" },
18:     {int (CIX_LSOUT), "CIX_LSOUT"},
19:     {int (CIX_ACK  ), "CIS_ACK"  },
20:     {int (CIX_NAK  ), "CIS_NAK"  },
21: };
22:
23:
24: void send_packet (base_socket& socket,
25:                  const void* buffer, size_t bufsize) {
26:     const char* bufptr = static_cast<const char*> (buffer);
27:     ssize_t ntosend = bufsize;
28:     do {
29:         ssize_t nbytes = socket.send (bufptr, ntosend);
30:         if (nbytes < 0) throw socket_sys_error (to_string (socket));
31:         bufptr += nbytes;
32:         ntosend -= nbytes;
33:     }while (ntosend > 0);
34: }
35:
36: void recv_packet (base_socket& socket, void* buffer, size_t bufsize) {
37:     char* bufptr = static_cast<char*> (buffer);
38:     ssize_t ntorecv = bufsize;
39:     do {
40:         ssize_t nbytes = socket.recv (bufptr, ntorecv);
41:         if (nbytes < 0) throw socket_sys_error (to_string (socket));
42:         if (nbytes == 0) throw socket_error (to_string (socket)
43:                                             + " is closed");
44:         bufptr += nbytes;
45:         ntorecv -= nbytes;
46:     }while (ntorecv > 0);
47: }
48:
49: ostream& operator<< (ostream& out, const cix_header& header) {
50:     const auto& itor = cix_command_map.find (header.cix_command);
51:     string code = itor == cix_command_map.end() ? "?" : itor->second;
52:     cout << "{" << header.cix_nbytes << ", " << code << " = "
53:          << int (header.cix_command) << ", \" " << header.cix_filename
54:          << "\"}";
55:     return out;
56: }
57:
```

```
58:
59: string get_cix_server_host (const vector<string>& args, size_t index) {
60:     if (index < args.size()) return args[index];
61:     char* host = getenv ("CIX_SERVER_HOST");
62:     if (host != nullptr) return host;
63:     return "localhost";
64: }
65:
66: in_port_t get_cix_server_port (const vector<string>& args,
67:                                size_t index) {
68:     string port = "-1";
69:     if (index < args.size()) port = args[index];
70:     else {
71:         char* envport = getenv ("CIX_SERVER_PORT");
72:         if (envport != nullptr) port = envport;
73:     }
74:     return stoi (port);
75: }
76:
```

```
1: // $Id: cix-daemon.cpp,v 1.7 2014-07-25 12:12:51-07 - - $
2:
3: #include <iostream>
4: #include <string>
5: #include <vector>
6: using namespace std;
7:
8: #include <libgen.h>
9: #include <sys/types.h>
10: #include <unistd.h>
11:
12: #include "cix_protocol.h"
13: #include "logstream.h"
14: #include "signal_action.h"
15: #include "sockets.h"
16:
17: logstream log (cout);
18:
19: void fork_cixserver (server_socket& server, accepted_socket& accept) {
20:     pid_t pid = fork();
21:     if (pid == 0) { // child
22:         server.close();
23:         string sock_fd = accept.to_string_socket_fd();
24:         log << "execlp cixserver (fd" << sock_fd << ")" << endl;
25:         execlp ("cix-server", "cix-server", sock_fd.c_str(), nullptr);
26:         // Can't get here?!
27:         log << "cix-server: execlp failed: " << strerror (errno) << endl;
28:         throw cix_exit();
29:     }else {
30:         accept.close();
31:         if (pid < 0) {
32:             log << "fork failed: " << strerror (errno) << endl;
33:         }else {
34:             log << "forked cixserver pid " << pid << endl;
35:         }
36:     }
37: }
38:
39: void reap_zombies() {
40:     for (;;) {
41:         int status;
42:         pid_t child = waitpid (-1, &status, WNOHANG);
43:         if (child <= 0) break;
44:         log << "child " << child
45:             << " exit " << (status >> 8)
46:             << " signal " << (status & 0x7F)
47:             << " core " << (status >> 7 & 1) << endl;
48:     }
49: }
50:
```

```
51:
52:
53: bool SIGINT_throw_cix_exit {false};
54: void signal_handler (int signal) {
55:     log << "signal_handler: caught " << strsignal (signal) << endl;
56:     reap_zombies();
57:     switch (signal) {
58:         case SIGINT: case SIGTERM: SIGINT_throw_cix_exit = true; break;
59:         default: break;
60:     }
61: }
62:
63: int main (int argc, char** argv) {
64:     log.execname (basename (argv[0]));
65:     log << "starting" << endl;
66:     vector<string> args (&argv[1], &argv[argc]);
67:     signal_action (SIGCHLD, signal_handler);
68:     signal_action (SIGINT, signal_handler);
69:     signal_action (SIGTERM, signal_handler);
70:     in_port_t port = get_cix_server_port (args, 0);
71:     try {
72:         server_socket listener (port);
73:         for (;;) {
74:             if (SIGINT_throw_cix_exit) throw cix_exit();
75:             log << to_string (hostinfo()) << " accepting port "
76:                 << to_string (port) << endl;
77:             accepted_socket client_sock;
78:             for (;;) {
79:                 if (SIGINT_throw_cix_exit) throw cix_exit();
80:                 try {
81:                     listener.accept (client_sock);
82:                     break;
83:                 }catch (socket_sys_error& error) {
84:                     switch (error.sys_errno) {
85:                         case EINTR:
86:                             log << "listener.accept caught "
87:                                 << strerror (EINTR) << endl;
88:                             break;
89:                         default:
90:                             throw;
91:                     }
92:                 }
93:             }
94:             log << "accepted " << to_string (client_sock) << endl;
95:             try {
96:                 fork_cixserver (listener, client_sock);
97:                 reap_zombies();
98:             }catch (socket_error& error) {
99:                 log << error.what() << endl;
100:            }
101:        }
102:    }catch (socket_error& error) {
103:        log << error.what() << endl;
104:    }catch (cix_exit& error) {
105:        log << "caught cix_exit" << endl;
106:    }
107:    log << "finishing" << endl;
108:    return 0;
```



**02/26/15**  
**17:55:21**

\$cmpps109-wm/Assignments/asg5-client-server/code/  
cix-daemon.cpp

**3/3**

```
109: }  
110:
```

```
1: // $Id: cix-client.cpp,v 1.7 2014-07-25 12:12:51-07 - - $
2:
3: #include <iostream>
4: #include <string>
5: #include <vector>
6: #include <unordered_map>
7: using namespace std;
8:
9: #include <libgen.h>
10: #include <sys/types.h>
11: #include <unistd.h>
12:
13: #include "cix_protocol.h"
14: #include "logstream.h"
15: #include "signal_action.h"
16: #include "sockets.h"
17:
18: logstream log (cout);
19:
20: unordered_map<string,cix_command> command_map {
21:     {"exit", CIX_EXIT},
22:     {"help", CIX_HELP},
23:     {"ls" , CIX_LS },
24: };
25:
26: void cix_help() {
27:     static vector<string> help = {
28:         "exit          - Exit the program.  Equivalent to EOF.",
29:         "get filename - Copy remote file to local host.",
30:         "help          - Print help summary.",
31:         "ls            - List names of files on remote server.",
32:         "put filename - Copy local file to remote host.",
33:         "rm filename  - Remove file from remote server.",
34:     };
35:     for (const auto& line: help) cout << line << endl;
36: }
37:
38: void cix_ls (client_socket& server) {
39:     cix_header header;
40:     header.cix_command = CIX_LS;
41:     log << "sending header " << header << endl;
42:     send_packet (server, &header, sizeof header);
43:     recv_packet (server, &header, sizeof header);
44:     log << "received header " << header << endl;
45:     if (header.cix_command != CIX_LSOUT) {
46:         log << "sent CIX_LS, server did not return CIX_LSOUT" << endl;
47:         log << "server returned " << header << endl;
48:     }else {
49:         char buffer[header.cix_nbytes + 1];
50:         recv_packet (server, buffer, header.cix_nbytes);
51:         log << "received " << header.cix_nbytes << " bytes" << endl;
52:         buffer[header.cix_nbytes] = '\0';
53:         cout << buffer;
54:     }
55: }
56:
```

```
57:
58: void usage() {
59:     cerr << "Usage: " << log.execname() << " [host] [port]" << endl;
60:     throw cix_exit();
61: }
62:
63: bool SIGINT_throw_cix_exit {false};
64: void signal_handler (int signal) {
65:     log << "signal_handler: caught " << strsignal (signal) << endl;
66:     switch (signal) {
67:         case SIGINT: case SIGTERM: SIGINT_throw_cix_exit = true; break;
68:         default: break;
69:     }
70: }
71:
72: int main (int argc, char** argv) {
73:     log.execname (basename (argv[0]));
74:     log << "starting" << endl;
75:     vector<string> args (&argv[1], &argv[argc]);
76:     signal_action (SIGINT, signal_handler);
77:     signal_action (SIGTERM, signal_handler);
78:     if (args.size() > 2) usage();
79:     string host = get_cix_server_host (args, 0);
80:     in_port_t port = get_cix_server_port (args, 1);
81:     log << to_string (hostinfo()) << endl;
82:     try {
83:         log << "connecting to " << host << " port " << port << endl;
84:         client_socket server (host, port);
85:         log << "connected to " << to_string (server) << endl;
86:         for (;;) {
87:             string line;
88:             getline (cin, line);
89:             if (cin.eof()) throw cix_exit();
90:             if (SIGINT_throw_cix_exit) throw cix_exit();
91:             log << "command " << line << endl;
92:             const auto& itor = command_map.find (line);
93:             cix_command cmd = itor == command_map.end()
94:                 ? CIX_ERROR : itor->second;
95:             switch (cmd) {
96:                 case CIX_EXIT:
97:                     throw cix_exit();
98:                     break;
99:                 case CIX_HELP:
100:                     cix_help();
101:                     break;
102:                 case CIX_LS:
103:                     cix_ls (server);
104:                     break;
105:                 default:
106:                     log << line << ": invalid command" << endl;
107:                     break;
108:             }
109:         }
110:     } catch (socket_error& error) {
111:         log << error.what() << endl;
112:     } catch (cix_exit& error) {
113:         log << "caught cix_exit" << endl;
114:     }
```

02/26/15  
17:55:21

\$cmpps109-wm/Assignments/asg5-client-server/code/  
cix-client.cpp

**3/3**

```
115:    log << "finishing" << endl;  
116:    return 0;  
117: }  
118:
```

```
1: // $Id: cix-server.cpp,v 1.7 2014-07-25 12:12:51-07 - - $
2:
3: #include <iostream>
4: using namespace std;
5:
6: #include <libgen.h>
7:
8: #include "cix_protocol.h"
9: #include "logstream.h"
10: #include "signal_action.h"
11: #include "sockets.h"
12:
13: logstream log (cout);
14:
15: void reply_ls (accepted_socket& client_sock, cix_header& header) {
16:     FILE* ls_pipe = popen ("ls -l", "r");
17:     if (ls_pipe == NULL) {
18:         log << "ls -l: popen failed: " << strerror (errno) << endl;
19:         header.cix_command = CIX_NAK;
20:         header.cix_nbytes = errno;
21:         send_packet (client_sock, &header, sizeof header);
22:     }
23:     string ls_output;
24:     char buffer[0x1000];
25:     for (;;) {
26:         char* rc = fgets (buffer, sizeof buffer, ls_pipe);
27:         if (rc == nullptr) break;
28:         ls_output.append (buffer);
29:     }
30:     header.cix_command = CIX_LSOUT;
31:     header.cix_nbytes = ls_output.size();
32:     memset (header.cix_filename, 0, CIX_FILENAME_SIZE);
33:     log << "sending header " << header << endl;
34:     send_packet (client_sock, &header, sizeof header);
35:     send_packet (client_sock, ls_output.c_str(), ls_output.size());
36:     log << "sent " << ls_output.size() << " bytes" << endl;
37: }
38:
```

```
39:
40: bool SIGINT_throw_cix_exit = false;
41: void signal_handler (int signal) {
42:     log << "signal_handler: caught " << strsignal (signal) << endl;
43:     switch (signal) {
44:         case SIGINT: case SIGTERM: SIGINT_throw_cix_exit = true; break;
45:         default: break;
46:     }
47: }
48:
49: int main (int argc, char** argv) {
50:     log.execname (basename (argv[0]));
51:     log << "starting" << endl;
52:     vector<string> args (&argv[1], &argv[argc]);
53:     signal_action (SIGINT, signal_handler);
54:     signal_action (SIGTERM, signal_handler);
55:     int client_fd = args.size() == 0 ? -1 : stoi (args[0]);
56:     log << "starting client_fd " << client_fd << endl;
57:     try {
58:         accepted_socket client_sock (client_fd);
59:         log << "connected to " << to_string (client_sock) << endl;
60:         for (;;) {
61:             if (SIGINT_throw_cix_exit) throw cix_exit();
62:             cix_header header;
63:             recv_packet (client_sock, &header, sizeof header);
64:             log << "received header " << header << endl;
65:             switch (header.cix_command) {
66:                 case CIX_LS:
67:                     reply_ls (client_sock, header);
68:                     break;
69:                 default:
70:                     log << "invalid header from client" << endl;
71:                     log << "cix_nbytes = " << header.cix_nbytes << endl;
72:                     log << "cix_command = " << header.cix_command << endl;
73:                     log << "cix_filename = " << header.cix_filename << endl;
74:                     break;
75:             }
76:         }
77:     } catch (socket_error& error) {
78:         log << error.what() << endl;
79:     } catch (cix_exit& error) {
80:         log << "caught cix_exit" << endl;
81:     }
82:     log << "finishing" << endl;
83:     return 0;
84: }
85:
```

```
1: # $Id: Makefile,v 1.6 2015-02-26 15:43:49-08 - - $
2:
3: GPP          = g++ -g -O0 -Wall -Wextra -std=gnu++11
4:
5: DEFPFILE     = Makefile.dep
6: HEADERS      = sockets.h signal_action.h cix_protocol.h logstream.h
7: CPPLIBS      = sockets.cpp signal_action.cpp cix_protocol.cpp
8: CPPSRCS      = ${CPPLIBS} cix-daemon.cpp cix-client.cpp cix-server.cpp
9: LIBOBSJS     = ${CPPLIBS:.cpp=.o}
10: CLIENTOBSJS  = cix-client.o ${LIBOBSJS}
11: SERVEROBSJS  = cix-server.o ${LIBOBSJS}
12: DAEMONOBSJS  = cix-daemon.o ${LIBOBSJS}
13: OBJECTS      = ${CLIENTOBSJS} ${SERVEROBSJS} ${DAEMONOBSJS}
14: EXECBINS     = cix-client cix-server cix-daemon
15: LISTING      = Listing.ps
16: SOURCES      = ${HEADERS} ${CPPSRCS} Makefile
17:
18: all: ${DEFPFILE} ${EXECBINS}
19:
20: cix-client: ${CLIENTOBSJS}
21:             ${GPP} -o $@ ${CLIENTOBSJS}
22:
23: cix-server: ${SERVEROBSJS}
24:             ${GPP} -o $@ ${SERVEROBSJS}
25:
26: cix-daemon: ${DAEMONOBSJS}
27:             ${GPP} -o $@ ${DAEMONOBSJS}
28:
29: %.o: %.cpp
30:             ${GPP} -c $<
31:
32: ci:
33:             - checksource ${SOURCES}
34:             - cid + ${SOURCES}
35:
36: lis: all ${SOURCES} ${DEFPFILE}
37:             mkpspdf ${LISTING} ${SOURCES} ${DEFPFILE}
38:
39: clean:
40:             - rm ${LISTING} ${LISTING:.ps=.pdf} ${OBJECTS} Makefile.dep
41:
42: spotless: clean
43:             - rm ${EXECBINS}
44:
45: dep:
46:             - rm ${DEFPFILE}
47:             make --no-print-directory ${DEFPFILE}
48:
49: ${DEFPFILE}:
50:             ${GPP} -MM ${CPPSRCS} >${DEFPFILE}
51:
52: again: ${SOURCES}
53:             make --no-print-directory spotless ci all lis
54:
55: include ${DEFPFILE}
56:
```

```
1: sockets.o: sockets.cpp sockets.h
2: signal_action.o: signal_action.cpp signal_action.h
3: cix_protocol.o: cix_protocol.cpp cix_protocol.h sockets.h
4: cix-daemon.o: cix-daemon.cpp cix_protocol.h sockets.h logstream.h \
5:   signal_action.h
6: cix-client.o: cix-client.cpp cix_protocol.h sockets.h logstream.h \
7:   signal_action.h
8: cix-server.o: cix-server.cpp cix_protocol.h sockets.h logstream.h \
9:   signal_action.h
```