```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #ifndef __SOCKET_H__
 5: #define __SOCKET_H__
 6:
 7: #include <cstring>
 8: #include <stdexcept>
 9: #include <string>
10: #include <vector>
11: using namespace std;
12:
13: #include <arpa/inet.h>
14: #include <netdb.h>
15: #include <netinet/in.h>
16: #include <string>
17: #include <sys/socket.h>
18: #include <sys/types.h>
19: #include <sys/wait.h>
20: #include <unistd.h>
21:
22: //
23: // class base_socket:
24: // mostly protected and not used by applications
25: //
26:
27: class base_socket {
28:    private:
29:       static constexpr size_t MAXRECV = 0xFFFF;
30:       static constexpr int CLOSED_FD = -1;
31:       int socket_fd {CLOSED_FD};
32:       sockaddr_in socket_addr;
33:       base_socket (const base_socket&) = delete; // prevent copying
34:       base_socket& operator= (const base_socket&) = delete;
35:    protected:
36:       base_socket(); // only derived classes may construct
37:       ~base_socket();
38:       // server_socket initialization
39:       void create();
40:       void bind (const in_port_t port);
41:       void listen() const;
42:       void accept (base_socket&) const;
43:       // client_socket initialization
44:       void connect (const string host, const in_port_t port);
45:       // accepted_socket initialization
46:       string to_string_socket_fd() { return to_string (socket_fd); }
47:       void set_socket_fd (int fd);
48:    public:
49:       void close();
50:       ssize_t send (const void* buffer, size_t bufsize);
51:       ssize_t recv (void* buffer, size_t bufsize);
52:       void set_non_blocking (const bool);
53:       friend string to_string (const base_socket& sock);
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:       accepted_socket(int fd) { set_socket_fd (fd); }
66:       string to_string_socket_fd() {
67:          return base_socket::to_string_socket_fd();
68:       }
69: };
70:
71: //
72: // class client_socket
73: // used by client application to connect to server
74: //
75:
76: class client_socket: public base_socket {
77:    public:
78:       client_socket (string host, in_port_t port);
79: };
80:
81: //
82: // class server_socket
83: // single use class by server application
84: //
85:
86: class server_socket: public base_socket {
87:    public:
88:       server_socket (in_port_t port);
89:       void accept (accepted_socket& sock) {
90:          base_socket::accept (sock);
91:       }
92: };
93:
```

```
 94:
 95: //
 96: // class socket_error
 97: // base class for throwing socket errors
 98: //
 99:
100: class socket_error: public runtime_error {
101:    public:
102:       explicit socket_error (const string& what): runtime_error(what){}
103: };
104:
105: //
106: // class socket_sys_error
107: // subclass to record status of extern int errno variable
108: //
109:
110: class socket_sys_error: public socket_error {
111:    public:
112:       int sys_errno;
113:       explicit socket_sys_error (const string& what):
114:                socket_error(what + ": " + strerror (errno)),
115:                sys_errno(errno) {}
116: };
117:
118: //
119: // class socket_h_error
120: // subclass to record status of extern int h_errno variable
121: //
122:
123: class socket_h_error: public socket_error {
124:    public:
125:       int host_errno;
126:       explicit socket_h_error (const string& what):
127:                socket_error(what + ": " + hstrerror (h_errno)),
128:                host_errno(h_errno) {}
129: };
130:
```

```
131:
132: //
133: // class hostinfo
134: // information about a host given hostname or IPv4 address
135: //
136:
137: class hostinfo {
138:    public:
139:       const string hostname;
140:       const vector<string> aliases;
141:       const vector<in_addr> addresses;
142:       hostinfo (); // localhost
143:       hostinfo (hostent*);
144:       hostinfo (const string& hostname);
145:       hostinfo (const in_addr& ipv4_addr);
146:       friend string to_string (const hostinfo&);
147: };
148:
149: string localhost();
150: string to_string (const in_addr& ipv4_addr);
151:
152: #endif
153:
```

```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #ifndef __SIGNAL_ACTION_H__
 5: #define __SIGNAL_ACTION_H__
 6:
 7: #include <stdexcept>
 8: using namespace std;
 9:
10: #include <signal.h>
11: #include <vector>
12: class cix_exit: public exception{};
13:
14: class signal_action {
15:    private:
16:       struct sigaction action;
17:    public:
18:       signal_action (int signal, void (*handler) (int));
19:       vector<string> split (const string& line, const
20:         string& delim);
21: };
22: class util{
23:     public:
24:      util() = default;
25:      vector<string> split (const string& line, const
26:         string& delim);
27: };
28:
29: class signal_error: runtime_error {
30:    public:
31:       int signal;
32:       explicit signal_error (int signal);
33: };
34:
35: #endif
36:
```

```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #ifndef __CIX_PROTOCOL__H__
 5: #define __CIX_PROTOCOL__H__
 6:
 7: #include <cstdint>
 8: #include <cstring>
 9: #include <iostream>
10: using namespace std;
11:
12: #include "sockets.h"
13:
14: enum cix_command {CIX_ERROR = 0, CIX_EXIT,
15:                   CIX_GET, CIX_HELP, CIX_LS, CIX_PUT, CIX_RM,
16:                   CIX_FILE, CIX_LSOUT, CIX_ACK, CIX_NAK};
17:
18: size_t constexpr CIX_FILENAME_SIZE = 59;
19: struct cix_header {
20:    uint32_t cix_nbytes {0};
21:    uint8_t cix_command {0};
22:    char cix_filename[CIX_FILENAME_SIZE] {};
23: };
24:
25: void send_packet (base_socket& socket,
26:                   const void* buffer, size_t bufsize);
27:
28: void recv_packet (base_socket& socket, void* buffer, size_t bufsize);
29:
30: ostream& operator<< (ostream& out, const cix_header& header);
31:
32: string get_cix_server_host (const vector<string>& args, size_t index);
33:
34: in_port_t get_cix_server_port (const vector<string>& args,
35:                                size_t index);
36:
37: #endif
38:
```

```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 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: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #include <cerrno>
 5: #include <cstring>
 6: #include <iostream>
 7: #include <sstream>
 8: #include <string>
 9: using namespace std;
10:
11: #include <fcntl.h>
12: #include <limits.h>
13:
14: #include "sockets.h"
15:
16: base_socket::base_socket() {
17:    memset (&socket_addr, 0, sizeof (socket_addr));
18: }
19:
20: base_socket::~base_socket() {
21:    if (socket_fd != CLOSED_FD) close();
22: }
23:
24: void base_socket::close() {
25:    int status = ::close (socket_fd);
26:    if (status < 0) throw socket_sys_error ("close("
27:                            + to_string(socket_fd) + ")");
28:    socket_fd = CLOSED_FD;
29: }
30:
31: void base_socket::create() {
32:    socket_fd = ::socket (AF_INET, SOCK_STREAM, 0);
33:    if (socket_fd < 0) throw socket_sys_error ("socket");
34:    int on = 1;
35:    int status = ::setsockopt (socket_fd, SOL_SOCKET, SO_REUSEADDR,
36:                               &on, sizeof on);
37:    if (status < 0) throw socket_sys_error ("setsockopt");
38: }
39:
40: void base_socket::bind (const in_port_t port) {
41:    socket_addr.sin_family = AF_INET;
42:    socket_addr.sin_addr.s_addr = INADDR_ANY;
43:    socket_addr.sin_port = htons (port);
44:    int status = ::bind (socket_fd,
45:                         reinterpret_cast<sockaddr*> (&socket_addr),
46:                         sizeof socket_addr);
47:    if (status < 0) throw socket_sys_error ("bind(" + to_string (port)
48:                                            + ")");
49: }
50:
51: void base_socket::listen() const {
52:    int status = ::listen (socket_fd, SOMAXCONN);
53:    if (status < 0) throw socket_sys_error ("listen");
54: }
55:
```

```
 56:
 57: void base_socket::accept (base_socket& socket) const {
 58:    int addr_length = sizeof socket.socket_addr;
 59:    socket.socket_fd = ::accept (socket_fd,
 60:                  reinterpret_cast<sockaddr*> (&socket.socket_addr),
 61:                  reinterpret_cast<socklen_t*> (&addr_length));
 62:    if (socket.socket_fd < 0) throw socket_sys_error ("accept");
 63: }
 64:
 65: ssize_t base_socket::send (const void* buffer, size_t bufsize) {
 66:    int nbytes = ::send (socket_fd, buffer, bufsize, MSG_NOSIGNAL);
 67:    if (nbytes < 0) throw socket_sys_error ("send");
 68:    return nbytes;
 69: }
 70:
 71: ssize_t base_socket::recv (void* buffer, size_t bufsize) {
 72:    memset (buffer, 0, bufsize);
 73:    ssize_t nbytes = ::recv (socket_fd, buffer, bufsize, 0);
 74:    if (nbytes < 0) throw socket_sys_error ("recv");
 75:    return nbytes;
 76: }
 77:
 78: void base_socket::connect (const string host, const in_port_t port) {
 79:    struct hostent *hostp = ::gethostbyname (host.c_str());
 80:    if (hostp == NULL) throw socket_h_error ("gethostbyname("
 81:                               + host + ")");
 82:    socket_addr.sin_family = AF_INET;
 83:    socket_addr.sin_port = htons (port);
 84:    socket_addr.sin_addr = *reinterpret_cast<in_addr*> (hostp->h_addr);
 85:    int status = ::connect (socket_fd,
 86:                            reinterpret_cast<sockaddr*> (&socket_addr),
 87:                            sizeof (socket_addr));
 88:    if (status < 0) throw socket_sys_error ("connect(" + host + ":"
 89:                           + to_string (port) + ")");
 90: }
 91:
 92: void base_socket::set_socket_fd (int fd) {
 93:    socklen_t addrlen = sizeof socket_addr;
 94:    int rc = getpeername (fd, reinterpret_cast<sockaddr*> (&socket_addr),
 95:                          &addrlen);
 96:    if (rc < 0) throw socket_sys_error ("set_socket_fd("
 97:                        + to_string (fd) + "): getpeername");
 98:    socket_fd = fd;
 99:    if (socket_addr.sin_family != AF_INET)
100:       throw socket_error ("address not AF_INET");
101: }
102:
103: void base_socket::set_non_blocking (const bool blocking) {
104:    int opts = ::fcntl (socket_fd, F_GETFL);
105:    if (opts < 0) throw socket_sys_error ("fcntl");
106:    if (blocking) opts |= O_NONBLOCK;
107:             else opts &= compl O_NONBLOCK;
108:    opts = ::fcntl (socket_fd, F_SETFL, opts);
109:    if (opts < 0) throw socket_sys_error ("fcntl");
110: }
111:
```

```
112:
113: client_socket::client_socket (string host, in_port_t port) {
114:    base_socket::create();
115:    base_socket::connect (host, port);
116: }
117:
118: server_socket::server_socket (in_port_t port) {
119:    base_socket::create();
120:    base_socket::bind (port);
121:    base_socket::listen();
122: }
123:
124: string to_string (const hostinfo& info) {
125:    return info.hostname + " (" + to_string (info.addresses[0]) + ")";
126: }
127:
128: string to_string (const in_addr& ipv4_addr) {
129:    char buffer[INET_ADDRSTRLEN];
130:    const char *result = ::inet_ntop (AF_INET, &ipv4_addr,
131:                                      buffer, sizeof buffer);
132:    if (result == NULL) throw socket_sys_error ("inet_ntop");
133:    return result;
134: }
135:
136: string to_string (const base_socket& sock) {
137:    hostinfo info (sock.socket_addr.sin_addr);
138:    return info.hostname + " (" + to_string (info.addresses[0])
139:           + ") port " + to_string (ntohs (sock.socket_addr.sin_port));
140: }
141:
```

```
142:
143: string init_hostname (hostent* host) {
144:    if (host == nullptr) throw socket_h_error ("gethostbyname");
145:    return host->h_name;
146: }
147:
148: vector<string> init_aliases (hostent* host) {
149:    if (host == nullptr) throw socket_h_error ("gethostbyname");
150:    vector<string> init_aliases;
151:    for (char** alias = host->h_aliases; *alias != nullptr; ++alias) {
152:       init_aliases.push_back (*alias);
153:    }
154:    return init_aliases;
155: }
156:
157: vector<in_addr> init_addresses (hostent* host) {
158:    vector<in_addr> init_addresses;
159:    if (host == nullptr) throw socket_h_error ("gethostbyname");
160:    for (in_addr** addr =
161:                  reinterpret_cast<in_addr**> (host->h_addr_list);
162:         *addr != nullptr; ++addr) {
163:       init_addresses.push_back (**addr);
164:    }
165:    return init_addresses;
166: }
167:
168: hostinfo::hostinfo (hostent* host):
169:    hostname (init_hostname (host)),
170:    aliases (init_aliases (host)),
171:    addresses (init_addresses (host)) {
172: }
173:
174: hostinfo::hostinfo(): hostinfo (localhost()) {
175: }
176:
177: hostinfo::hostinfo (const string& hostname):
178:           hostinfo (::gethostbyname (hostname.c_str())) {
179: }
180:
181: hostinfo::hostinfo (const in_addr& ipv4_addr):
182:           hostinfo (::gethostbyaddr (&ipv4_addr, sizeof ipv4_addr,
183:                                      AF_INET)) {
184: }
185:
186: string localhost() {
187:    char hostname[HOST_NAME_MAX] {};
188:    int rc = gethostname (hostname, sizeof hostname);
189:    if (rc < 0) throw socket_sys_error ("gethostname");
190:    return hostname;
191: }
192:
```

```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #include <cstring>
 5: #include <string>
 6: #include <unordered_map>
 7: using namespace std;
 8:
 9: #include "signal_action.h"
10:
11: signal_action::signal_action (int signal, void (*handler) (int)) {
12:    action.sa_handler = handler;
13:    sigfillset (&action.sa_mask);
14:    action.sa_flags = 0;
15:    int rc = sigaction (signal, &action, nullptr);
16:    if (rc < 0) throw signal_error (signal);
17: }
18:
19: vector<string> util::split(const string& line,
20:   const string& delimiter){
21:    vector<string> words;
22:    size_t ending = 0;
23:    for(;;){
24:     size_t starting = line.find_first_not_of (delimiter, ending);
25:    if (starting == string::npos) break;
26:    ending = line.find_first_of (delimiter, starting);
27:    words.push_back (line.substr (starting, ending - starting));
28:   } return words;
29:
30: }
31:
32: signal_error::signal_error (int signal):
33:               runtime_error (string ("signal_error(")
34:                              + strsignal (signal) + ")"),
35:               signal(signal) {}
36:
```

```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #include <unordered_map>
 5: #include <string>
 6: using namespace std;
 7:
 8: #include "cix_protocol.h"
 9:
10: const unordered_map<int,string> cix_command_map {
11:    {int (CIX_ERROR), "CIX_ERROR"},
12:    {int (CIX_EXIT ), "CIX_EXIT" },
13:    {int (CIX_GET  ), "CIX_GET"  },
14:    {int (CIX_HELP ), "CIX_HELP" },
15:    {int (CIX_LS   ), "CIX_LS"   },
16:    {int (CIX_PUT  ), "CIX_PUT"  },
17:    {int (CIX_RM   ), "CIX_RM"   },
18:    {int (CIX_FILE ), "CIX_FILE" },
19:    {int (CIX_LSOUT), "CIX_LSOUT"},
20:    {int (CIX_ACK  ), "CIS_ACK"  },
21:    {int (CIX_NAK  ), "CIS_NAK"  },
22: };
23:
24:
25: void send_packet (base_socket& socket,
26:                   const void* buffer, size_t bufsize) {
27:    const char* bufptr = static_cast<const char*> (buffer);
28:    ssize_t ntosend = bufsize;
29:    do {
30:       ssize_t nbytes = socket.send (bufptr, ntosend);
31:       if (nbytes < 0) throw socket_sys_error (to_string (socket));
32:       bufptr += nbytes;
33:       ntosend -= nbytes;
34:    }while (ntosend > 0);
35: }
36:
37: void recv_packet (base_socket& socket, void* buffer, size_t bufsize) {
38:    char* bufptr = static_cast<char*> (buffer);
39:    ssize_t ntorecv = bufsize;
40:    do {
41:       ssize_t nbytes = socket.recv (bufptr, ntorecv);
42:       if (nbytes < 0) throw socket_sys_error (to_string (socket));
43:       if (nbytes == 0) throw socket_error (to_string (socket)
44:                                            + " is closed");
45:       bufptr += nbytes;
46:       ntorecv -= nbytes;
47:    }while (ntorecv > 0);
48: }
49:
50: ostream& operator<< (ostream& out, const cix_header& header) {
51:    const auto& itor = cix_command_map.find (header.cix_command);
52:    string code = itor == cix_command_map.end() ? "?" : itor->second;
53:    cout << "{" << header.cix_nbytes << "," << code << "="
54:         << int (header.cix_command) << ",\"" << header.cix_filename
55:         << "\"}";
56:    return out;
57: }
58:
```

```
59:
60: string get_cix_server_host (const vector<string>& args, size_t index) {
61:    if (index < args.size()) return args[index];
62:    char* host = getenv ("CIX_SERVER_HOST");
63:    if (host != nullptr) return host;
64:    return "localhost";
65: }
66:
67: in_port_t get_cix_server_port (const vector<string>& args,
68:                                size_t index) {
69:    string port = "-1";
70:    if (index < args.size()) port = args[index];
71:    else {
72:       char* envport = getenv ("CIX_SERVER_PORT");
73:       if (envport != nullptr) port = envport;
74:    }
75:    cout<<"port: "<<port<<endl;
76:    return stoi (port);
77: }
78:
```

```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #include <iostream>
 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 "cix_protocol.h"
14: #include "logstream.h"
15: #include "signal_action.h"
16: #include "sockets.h"
17:
18: logstream log (cout);
19:
20: void fork_cixserver (server_socket& server, accepted_socket& accept) {
21:    pid_t pid = fork();
22:    if (pid == 0) {
23:       server.close();
24:       string sock_fd = accept.to_string_socket_fd();
25:       log << "execlp cixserver (fd" << sock_fd << ")" << endl;
26:       execlp ("cix-server", "cix-server", sock_fd.c_str(), nullptr);
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;
```

```
109: }
110:
```

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

```
 59:       cout << buffer;
 60:    }
 61: }
 62:
 63: void cix_get (client_socket& server, const string& info) {
 64:    cix_header header;
 65:    header.cix_command = CIX_GET;
 66:
 67:    for(size_t i =0;i<info.size();i++)
 68:      header.cix_filename[i]=info[i];
 69:    string file_name = header.cix_filename;
 70:    cout<<header<<": "<<endl;
 71:    log << "header sending " << header << endl;
 72:    send_packet (server, &header, sizeof header);
 73:    recv_packet (server, &header, sizeof header);
 74:    log << "header received " << header << endl;
 75:    if (header.cix_command != CIX_ACK) {
 76:       log << "sent RM, server did not return CIX_ACK" << endl;
 77:       log << "server returned " << header << endl;
 78:    }else {
 79:       ofstream outfile (header.cix_filename);
 80:       char buffer[header.cix_nbytes + 1];
 81:       recv_packet (server, buffer, header.cix_nbytes);
 82:       log << "received " << header.cix_nbytes << " bytes" << endl;
 83:       buffer[header.cix_nbytes] = '\0';
 84:       for(auto c: buffer)cout<<c<<endl;
 85:       outfile.write(buffer, header.cix_nbytes);
 86:       outfile.close();
 87:       cout << "here is your new file  @ 'new_file.txt'"<<endl;
 88:    }
 89: }
 90: void cix_put (client_socket& server, const string& info) {
 91:    cix_header header;
 92:    header.cix_command = CIX_PUT;
 93:    cout<<info<<endl;
 94:    for(size_t i =0;i<info.size();i++){
 95:      header.cix_filename[i]=info[i];
 96:    }
 97:       ifstream is (header.cix_filename, ifstream::binary);
 98:       int length = 0;
 99:    if (is == nullptr) {
100:       log << "get : popen failed: " << strerror (errno) << endl;
101:       header.cix_command = CIX_NAK;
102:       header.cix_nbytes = errno;
103:       send_packet (server, &header, sizeof header);
104:       cout<<"sent failure"<<endl;
105:    }else{
106:        is.seekg(0, is.end);
107:        length = is.tellg();
108:        is.seekg (0, is.beg);
109:        char* buffer = new char[length];
110:        is.read(buffer,length);
111:        header.cix_nbytes = length;
112:        log << "header sending " << header << endl;
113:        send_packet (server, &header, sizeof header);
114:        send_packet (server, buffer,
115:         header.cix_nbytes);
116:        log << "sent " << length << " bytes" << endl;
```

```
117:         is.close();
118:         delete[] buffer;
119:
120:    }
121:  }
122:
123: void cix_rm (client_socket& server, const string& info) {
124:    cix_header header;
125:    header.cix_command = CIX_RM;
126:
127:    strcpy(header.cix_filename, info.c_str());
128:    cout<<header<<": "<<endl;
129:    log << "header sending " << header << endl;
130:    send_packet (server, &header, sizeof header);
131:    recv_packet (server, &header, sizeof header);
132:    log << "header received " << header << endl;
133:    if (header.cix_command != CIX_ACK) {
134:       log << "sent RM, server did not return CIX_ACK" << endl;
135:       log << "server returned " << header << endl;
136:    }
137: }
138: void usage() {
139:    cerr << "Usage: " << log.execname() << " [host] [port]" << endl;
140:    throw cix_exit();
141: }
142:
143: bool SIGINT_throw_cix_exit {false};
144: void signal_handler (int signal) {
145:    log << "signal_handler: caught " << strsignal (signal) << endl;
146:    switch (signal) {
147:       case SIGINT: case SIGTERM: SIGINT_throw_cix_exit = true; break;
148:       default: break;
149:    }
150: }
151:
152: int main (int argc, char** argv) {
153:    util utilz;
154:    log.execname (basename (argv[0]));
155:    log << "starting" << endl;
156:    vector<string> args (&argv[1], &argv[argc]);
157:    signal_action (SIGINT, signal_handler);
158:    signal_action (SIGTERM, signal_handler);
159:    if (args.size() > 2) usage();
160:    string host = get_cix_server_host (args, 0);
161:    in_port_t port = get_cix_server_port (args, 1);
162:    cout<<"my port: "<<port<<endl;
163:    log << to_string (hostinfo()) << endl;
164:    try {
165:       log << "connecting to " << host << " port " << port << endl;
166:       client_socket server (host, port);
167:       log << "connected to " << to_string (server) << endl;
168:       for (;;) {
169:          string line;
170:
171:
172:          getline (cin, line);
173:          if (cin.eof()) throw cix_exit();
174:          if (SIGINT_throw_cix_exit) throw cix_exit();
```

```
175:          log << "command " << line << endl;
176:          vector<string> full_cmd = utilz.split(line, " ");
177:          string cmd_info;
178:          string cmd_core;
179:          if(full_cmd.size()>=2)
180:            cmd_info = full_cmd.at(1);
181:          cmd_core = full_cmd.at(0);
182:
183:          const auto& itor = command_map.find (cmd_core);
184:          cix_command cmd = itor == command_map.end()
185:                          ? CIX_ERROR : itor->second;
186:              switch (cmd) {
187:             case CIX_EXIT:
188:                throw cix_exit();
189:                break;
190:             case CIX_HELP:
191:                cix_help();
192:                break;
193:             case CIX_LS:
194:                cix_ls (server);
195:                break;
196:             case CIX_PUT:
197:                cout<<"To put:"<<endl;
198:                cix_put(server, cmd_info);
199:                break;
200:             case CIX_GET:
201:                cout<<"To get:"<<endl;
202:                cix_get(server, cmd_info);
203:                break;
204:             case CIX_RM:
205:                cout<<"To RM:"<<endl;
206:                cix_rm(server, cmd_info);
207:                break;
208:             default:
209:                log << line << ": invalid command" << endl;
210:                break;
211:          }
212:       }
213:    }catch (socket_error& error) {
214:       log << error.what() << endl;
215:    }catch (cix_exit& error) {
216:       log << "caught cix_exit" << endl;
217:    }
218:    log << "finishing" << endl;
219:    return 0;
220: }
221:
```

```
 1: //Dara Diba ddiba@ucsc.edu
 2: //Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: #include <iostream>
 5: using namespace std;
 6:
 7: #include <libgen.h>
 8: #include <iostream>
 9: #include <fstream>
10:
11: #include "cix_protocol.h"
12: #include "logstream.h"
13: #include "signal_action.h"
14: #include "sockets.h"
15:
16: logstream log (cout);
17:
18: void reply_ls (accepted_socket& client_sock, cix_header& header) {
19:    FILE* ls_pipe = popen ("ls -l", "r");
20:    if (ls_pipe == NULL) {
21:       log << "ls -l: popen failed: " << strerror (errno) << endl;
22:       header.cix_command = CIX_NAK;
23:       header.cix_nbytes = errno;
24:       send_packet (client_sock, &header, sizeof header);
25:    }
26:    string ls_output;
27:    char buffer[0x1000];
28:    for (;;) {
29:       char* rc = fgets (buffer, sizeof buffer, ls_pipe);
30:       if (rc == nullptr) break;
31:       ls_output.append (buffer);
32:    }
33:    header.cix_command = CIX_LSOUT;
34:    header.cix_nbytes = ls_output.size();
35:    memset (header.cix_filename, 0, CIX_FILENAME_SIZE);
36:    log << "sending header " << header << endl;
37:    send_packet (client_sock, &header, sizeof header);
38:    send_packet (client_sock, ls_output.c_str(), ls_output.size());
39:    log << "sent " << ls_output.size() << " bytes" << endl;
40: }
41: void reply_put(accepted_socket& client_sock, cix_header& header){
42:    char buffer[header.cix_nbytes +1];
43:    recv_packet (client_sock, buffer, sizeof header);
44:    log << "received header " << header << endl;
45:    ofstream outfile (header.cix_filename);
46:    if(outfile){
47:       buffer[header.cix_nbytes] = '\0';
48:       outfile.write(buffer, header.cix_nbytes);
49:       outfile.close();
50:       header.cix_command = CIX_ACK;
51:       send_packet(client_sock, &header, sizeof header);
52:       cout << "here is your new file foo @ 'new_file.txt'"<<endl;
53:    }else{
54:       header.cix_command = CIX_NAK;
55:       send_packet(client_sock, &header, sizeof header);
56:
57:    }
58:
```

```
 59: }
 60:
 61: void reply_get (accepted_socket& client_sock, cix_header& header) {
 62:    string get_output = "";
 63:    ifstream is (header.cix_filename, ifstream::binary);
 64:    int length = 0;
 65:    if (is == nullptr) {
 66:       log << "get : popen failed: " << strerror (errno) << endl;
 67:       header.cix_command = CIX_NAK;
 68:       header.cix_nbytes = errno;
 69:       send_packet (client_sock, &header, sizeof header);
 70:       cout<<"sent failure"<<endl;
 71:    }else{
 72:       is.seekg(0, is.end);
 73:       length = is.tellg();
 74:       is.seekg (0, is.beg);
 75:       char* buffer = new char[length];
 76:       is.read(buffer,length);
 77:       if(is){
 78:          cout<<"complete transfer"<<endl;
 79:
 80:      } else cout<<"error only: "<<is.gcount()<<"read"<<endl;
 81:       is.close();
 82:
 83:       header.cix_command = CIX_ACK;
 84:       header.cix_nbytes = length;
 85:       log << "sending header " << header << endl;
 86:       send_packet (client_sock, &header, sizeof header);
 87:       send_packet (client_sock, buffer,
 88:         header.cix_nbytes);
 89:       log << "sent " << length << " bytes" << endl;
 90:       delete[] buffer;
 91:    }
 92: }
 93:
 94: void reply_rm (accepted_socket& client_sock, cix_header& header) {
 95:    string rm_output = "";
 96:    int rc = unlink(header.cix_filename);
 97:    if (rc <0) {
 98:       log << "rm : popen failed: " << strerror (rc) << endl;
 99:       header.cix_command = CIX_NAK;
100:       header.cix_nbytes = rc;
101:       send_packet (client_sock, &header, sizeof header);
102:       cout<<"sent failure"<<endl;
103:    }
104:
105:    header.cix_command = CIX_ACK;
106:    header.cix_nbytes = rm_output.size();
107:    memset (header.cix_filename, 0, CIX_FILENAME_SIZE);
108:    log << "sending header " << header << endl;
109:    send_packet (client_sock, &header, sizeof header);
110:    send_packet (client_sock, rm_output.c_str(), rm_output.size());
111:    log << "sent " << rm_output.size() << " bytes" << endl;
112: }
```

```
113:
114: bool SIGINT_throw_cix_exit = false;
115: void signal_handler (int signal) {
116:    log << "signal_handler: caught " << strsignal (signal) << endl;
117:    switch (signal) {
118:       case SIGINT: case SIGTERM: SIGINT_throw_cix_exit = true; break;
119:       default: break;
120:    }
121: }
122:
123: int main (int argc, char** argv) {
124:    log.execname (basename (argv[0]));
125:    log << "starting" << endl;
126:    vector<string> args (&argv[1], &argv[argc]);
127:    signal_action (SIGINT, signal_handler);
128:    signal_action (SIGTERM, signal_handler);
129:    int client_fd = args.size() == 0 ? -1 : stoi (args[0]);
130:    log << "starting client_fd " << client_fd << endl;
131:    try {
132:       accepted_socket client_sock (client_fd);
133:       log << "connected to " << to_string (client_sock) << endl;
134:       for (;;) {
135:          if (SIGINT_throw_cix_exit) throw cix_exit();
136:          cix_header header;
137:          recv_packet (client_sock, &header, sizeof header);
138:          log << "header received" << header << endl;
139:          switch (header.cix_command) {
140:             case CIX_LS:
141:                reply_ls (client_sock, header);
142:                break;
143:             case CIX_GET:
144:                reply_get(client_sock, header);
145:                break;
146:             case CIX_RM:
147:                cout<<"SERVER IS AWAKE: RM.."<<endl;
148:                reply_rm(client_sock, header);
149:                break;
150:             case CIX_PUT:
151:                reply_put(client_sock, header);
152:                break;
153:             default:
154:                log << "invalid header from client" << endl;
155:                log << "cix_nbytes = " << header.cix_nbytes << endl;
156:                log << "cix_command = " << header.cix_command << endl;
157:                log << "cix_filename = " << header.cix_filename << endl;
158:                break;
159:          }
160:       }
161:    }catch (socket_error& error) {
162:       log << error.what() << endl;
163:    }catch (cix_exit& error) {
164:       log << "caught cix_exit" << endl;
165:    }
166:    log << "finishing" << endl;
167:    return 0;
168: }
169:
```

```
 1: # Dara Diba ddiba@ucsc.edu
 2: # Nirav Agrawal nkagrawa@ucsc.edu
 3:
 4: GPP        = g++ -g -O0 -Wall -Wextra -std=gnu++11
 5:
 6: DEPFILE    = Makefile.dep
 7: HEADERS    = sockets.h signal_action.h cix_protocol.h logstream.h
 8: CPPLIBS    = sockets.cpp signal_action.cpp cix_protocol.cpp
 9: CPPSRCS    = ${CPPLIBS} cix-daemon.cpp cix-client.cpp cix-server.cpp
10: LIBOBJS    = ${CPPLIBS:.cpp=.o}
11: CLIENTOBJS = cix-client.o ${LIBOBJS}
12: SERVEROBJS = cix-server.o ${LIBOBJS}
13: DAEMONOBJS = cix-daemon.o ${LIBOBJS}
14: OBJECTS    = ${CLIENTOBJS} ${SERVEROBJS} ${DAEMONOBJS}
15: EXECBINS   = cix-client cix-server cix-daemon
16: LISTING    = Listing.ps
17: SOURCES    = ${HEADERS} ${CPPSRCS} Makefile
18:
19: all: ${DEPFILE} ${EXECBINS}
20:         - checksource ${SOURCES}
21:
22: cix-client: ${CLIENTOBJS}
23:         ${GPP} -o $@ ${CLIENTOBJS}
24:
25: cix-server: ${SERVEROBJS}
26:         ${GPP} -o $@ ${SERVEROBJS}
27:
28: cix-daemon: ${DAEMONOBJS}
29:         ${GPP} -o $@ ${DAEMONOBJS}
30:
31: %.o: %.cpp
32:         ${GPP} -c $<
33:
34: ci:
35:         - checksource ${SOURCES}
36:         - cid + ${SOURCES}
37:
38: lis: all ${SOURCES} ${DEPFILE}
39:         mkpspdf ${LISTING} ${SOURCES} ${DEPFILE}
40:
41: clean:
42:         - rm ${LISTING} ${LISTING:.ps=.pdf} ${OBJECTS}
43:
44: spotless: clean
45:         - rm ${EXECBINS}
46:
47: dep:
48:         - rm ${DEPFILE}
49:         make --no-print-directory ${DEPFILE}
50:
51: ${DEPFILE}:
52:         ${GPP} -MM ${CPPSRCS} >${DEPFILE}
53:
54: again: ${SOURCES}
55:         make --no-print-directory spotless ci all lis
56:
57: include ${DEPFILE}
58:
```

```
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
```