C SOCKETS

LECTURE 12-2

JIM FIX, REED COLLEGE CS2-S20

TODAY'S PLAN

- FINAL SEMESTER LOGISTICS
- SHOW YOU TWO DEMOS HOSTED BY AWS
  - ECHOING SERVICE
  - NETGRID TURTLE GAME SERVER
- OVERVIEW OF SOCKETS LIBRARY
  - ECHO CLIENT CODE
  - ECHO SERVER CODE, MULTITHREADED VERSION
- `netgrid` SERVER, `turtle` CLIENT

LOGISTICS

- Final Exam
  - Officially *Tuesday, May 12, 6-9pm PST*
    - ➡ Will instead post instructions and a Git repo at 3pm PST that day
    - ➡ You will submit your work by Wednesday 3pm (the next day)
  - Comprehensive. Two hours of work, but will give you 3+ hours.
  - Have shared two practice final exams.
- Review Session
  - Monday, May 11, 3-5pm PST over Zoom

SEMESTER TOPICS

- Basic C++ coding
  - procedures, functions, loops, conditionals.
  - int, double, char, bool; data sizes
  - modular arithmetic
  - std::string, std::cin, std::cout

SEMESTER TOPICS

- Data structures
  - C-style arrays and structs
  - stacks, queues, and other containers
  - link-based data structures: linked lists and trees
  - hash tables

SEMESTER TOPICS

- C++ memory model
  - pointer types
  - stack versus heap allocation
  - new, delete, delete []
  - the * and the & operator
  - passing by reference

SEMESTER TOPICS

- Logic and digital representation
  - AND, OR, NOT; boolean algebra
  - truth tables; sum-of-products
  - combinatorial circuit design
  - flip-flops and registers
  - sequential circuits
  - binary coding and binary arithmetic
  - two's complement

SEMESTER TOPICS

- MIPS32 Assembly programming
  - register operations; ALU operations
  - loops and branches
  - load and store
  - SPIM system calls
  - function call and return
  - register conventions and the stack frame

SEMESTER TOPICS

- C++ object-orientation
  - class definition
  - instance methods, instance variables; class methods, class variables
  - constructors and destructor; initializers
  - operator overloading
  - inheritance, abstract classes, virtual methods
  - templates

SEMESTER TOPICS

- C++ standard template library
  - vector, ordered_map, unordered_map
  - lambdas
  - smart pointers

LOGISTICS

- Feedback and help
  - Will be grading the 2nd midterm this weekend until Tuesday.
  - TAs will be grading Homeworks 9 and 10 next week.
  - Solutions to all assignments (except 11) posted by Sunday before finals.
  - Will have office hours next week.
- Deadlines
  - Homework 11 lambdas due Monday of finals week.
  - All remaining work due Thursday of finals week.

ECHO SERVICE DEMO

- Have two C programs in the **`sockets`** folder
  - **`echo_client.c`** - sends message to server, recieves message back
  - **`echo_server.c`** - receives client messages, repeats it back
    - ➡ compile with **`make`**; or with **`gcc -o pgm pgm.c`**

- Run the server first with a line like

```
    ./echo_server 8009
```

- Run clients in other consoles with a line like

```
    ./echo_client localhost 8009
```

ECHO SERVICE DEMO

- Have two C programs in the **`sockets`** folder
  - **`echo_client.c`** - sends message to server, recieves message back
  - **`echo_server.c`** - receives client messages, repeats it back
    - ➡ compile with **`make`**; or with **`gcc -o pgm pgm.c`**

- Run the server first with a line like

```
./echo_server SOME_PORT_#
```

- Run clients in other consoles with a line like

```
./echo_client HOST_NAME SAME_PORT_#
```

ECHO SERVICE DEMO

- Have two C programs in the **`sockets`** folder
  - **`echo_client.c`** - sends message to server, recieves message back
  - **`echo_server.c`** - receives client messages, repeats it back
    - ➡ compile with **`make`**; or with **`gcc -o pgm pgm.c`**

- Run the server first with a line like

```
./echo_server SOME_PORT_#
```

- Run clients in other consoles with a line like

```
./echo_client IP_ADDRESS SAME_PORT_#
```

ECHO SERVICE DEMO

- Have two C programs in the **`sockets`** folder
  - **`echo_client.c`** - sends message to server, recieves message back
  - **`echo_server.c`** - receives client messages, repeats it back
    - ➡ compile with **`make`**; or with **`gcc -o pgm pgm.c`**

- Run the server first with a line like

```
./echo_server 8009
```

- Run clients in other consoles with a line like

```
./echo_client 3.21.148.148 8009
```

LIVE SERVER ON AWS

GRID TURTLE GAME SERVER DEMO

- Have two C programs in the **`netgrid`** folder
  - **`netgrid.c`** - provides an 8x18 grid for turtles to connect
  - **`turtle.c`** - client program that plays the game
    - ➡ compile the client with **`make turtle`**
- I'm running a netgrid server on **port # 8001** of that same machine

```
./netgrid 8001
```

- You can join the grid with the command

```
./turtle 3.21.148.148 8001
```

GRID TURTLE GAME SERVER DEMO

- You can join the grid with the command

```
./turtle 3.21.148.148 8001
```

- It will ask you for your turtle's name and a starting coordinate.
- Then you can type in commands like:

```
forward
left
right
on
off
build
clear
who
text NAME MESSAGE
```

GRID TURTLE GAME SERVER DEMO

- You can join the grid with the command

```
./turtle 3.21.148.148 8001
```

- It will ask you for your turtle's name and a starting coordinate.
- Then you can type in commands like:

```
forward
left
right
on
off
build
clear
who
text NAME MESSAGE
```

TRY IT OUT!!!

GRID TURTLE GAME SERVER DEMO

- You can join the grid with the command

```
./turtle 3.21.148.148 8001
```

- It will ask you for your turtle's name and a starting coordinate.
- Then you can type in commands like:

```
forward
left
right
on
off
build
clear
who
text NAME MESSAGE
```

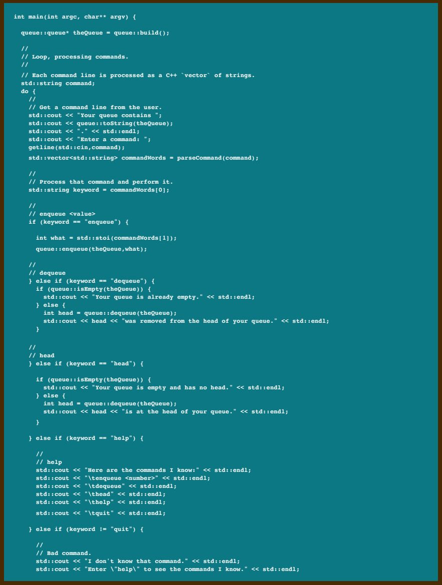
TRY IT OUT!!!

(sorry about the bugs)

SOCKETS LIBRARY

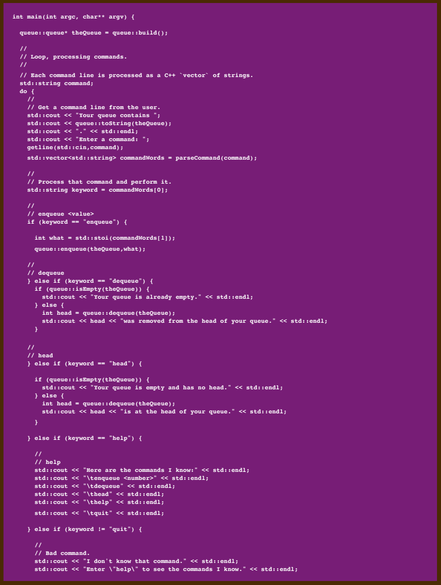
- The code is built using the C sockets library
  - This is an old code base built for Berkeley's Unix distribution
- Can communicate using TCP/IP to a process on another machine
  - That server process listens on a "port."
  - The machine is identified by its internet protocol (IP) address.

client.c



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

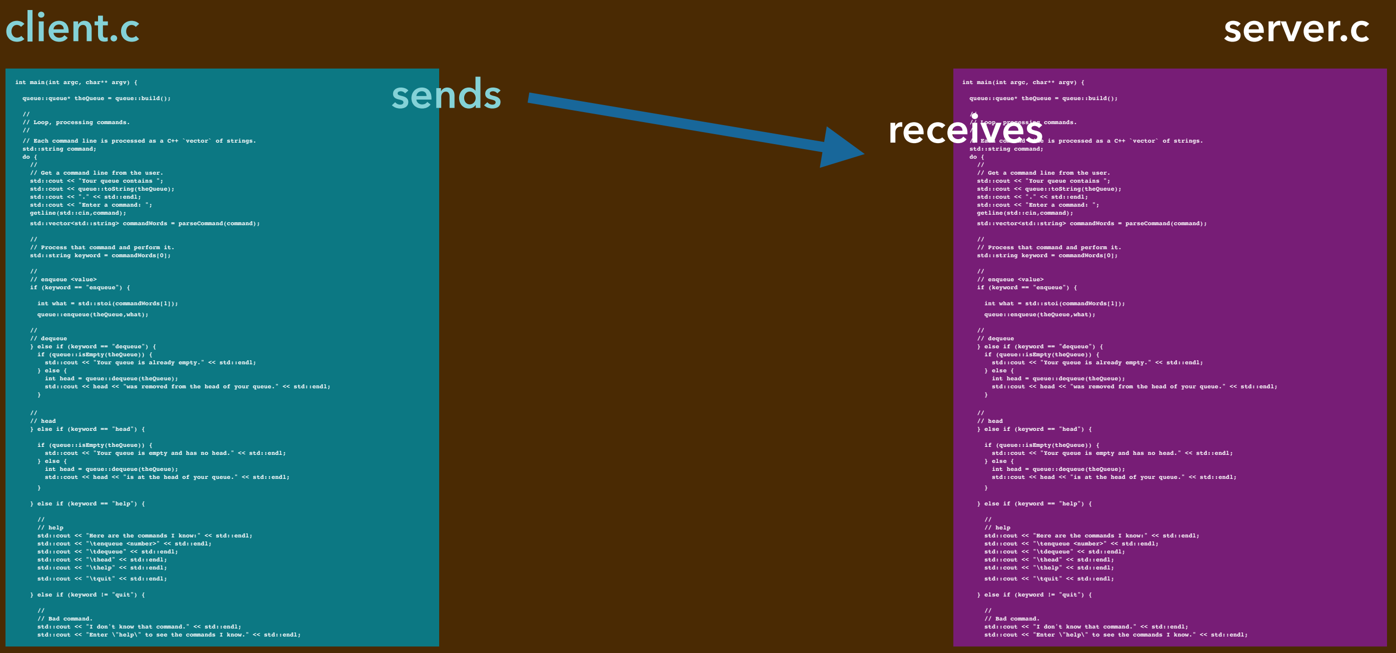
server.c



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- The code is built using the C sockets library
  - This is an old code base built for Berkeley's Unix distribution
- Can communicate using TCP/IP to a process on another machine
  - That server process listens on a "port."
  - The machine is identified by its internet protocol (IP) address.

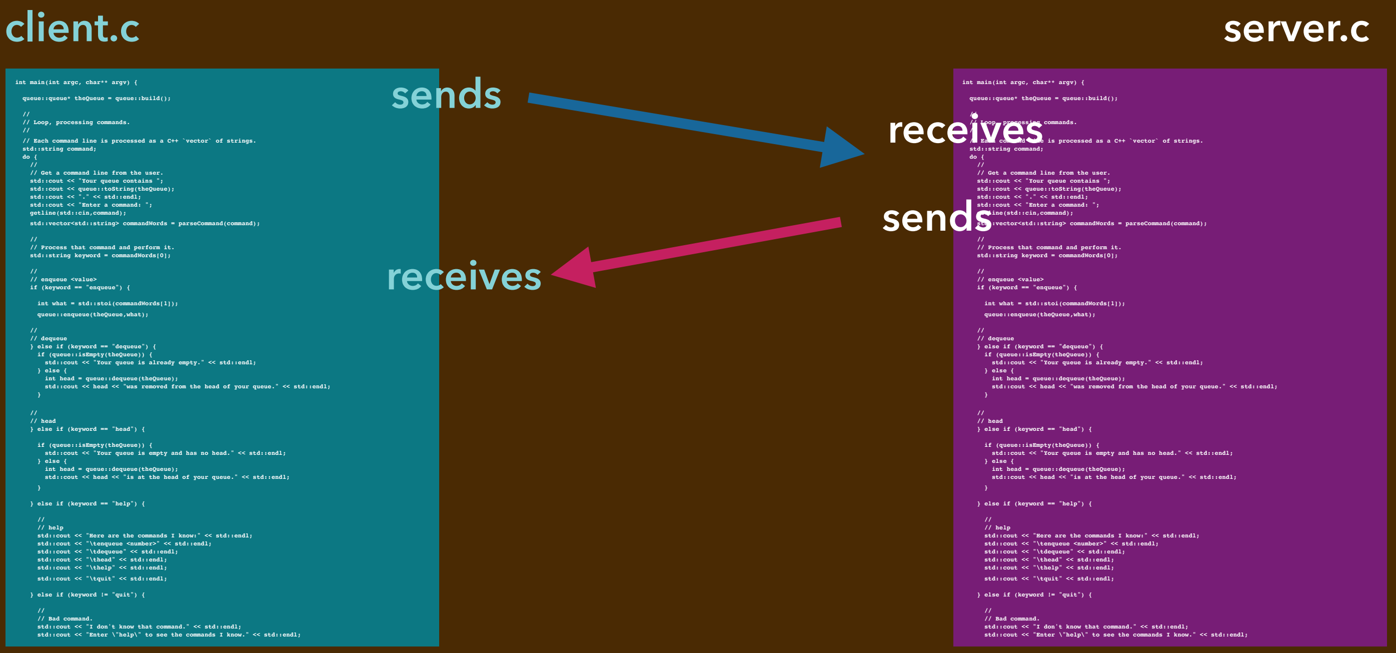


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- The code is built using the C sockets library
  - This is an old code base built for Berkeley's Unix distribution
- Can communicate using TCP/IP to a process on another machine
  - That server process listens on a "port."
  - The machine is identified by its internet protocol (IP) address.

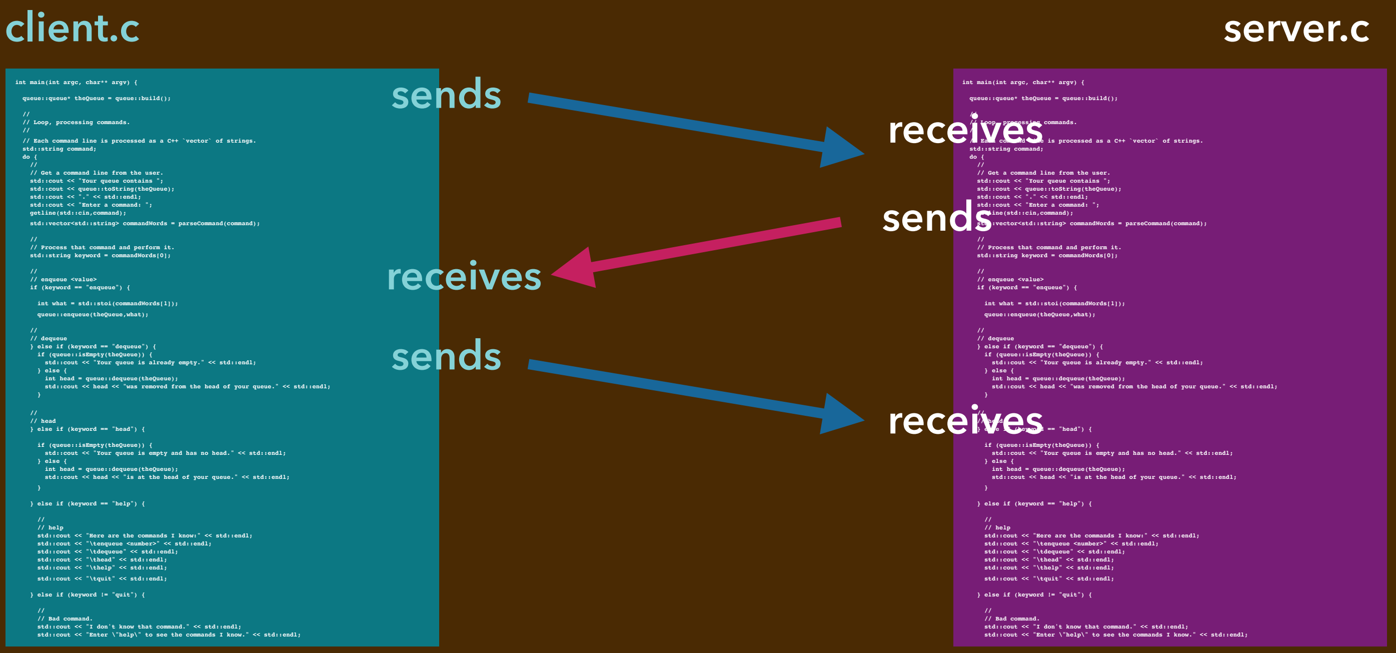


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

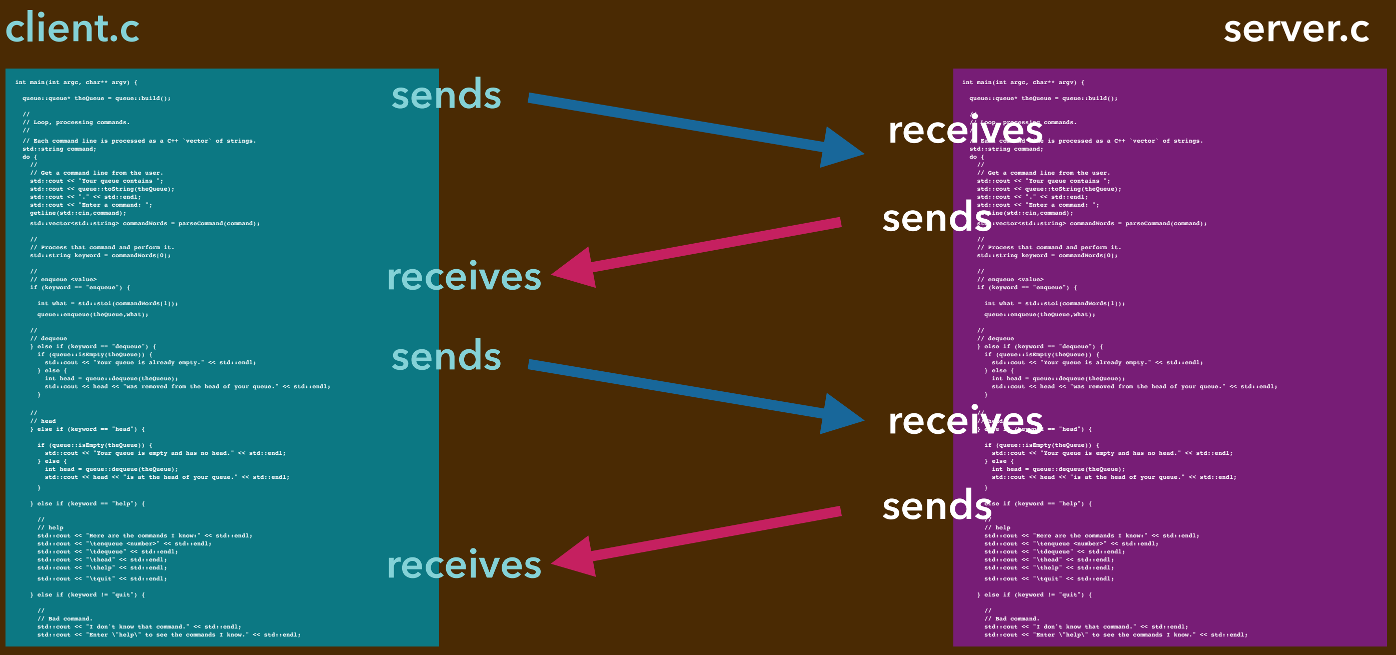
- The code is built using the C sockets library
    - This is an old code base built for Berkeley's Unix distribution
- Can communicate using TCP/IP to a process on another machine
    - That server process listens on a "port."
    - The machine is identified by its internet protocol (IP) address.



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- The code is built using the C sockets library
  - This is an old code base built for Berkeley's Unix distribution
- Can communicate using TCP/IP to a process on another machine
  - That server process listens on a "port."
  - The machine is identified by its internet protocol (IP) address.



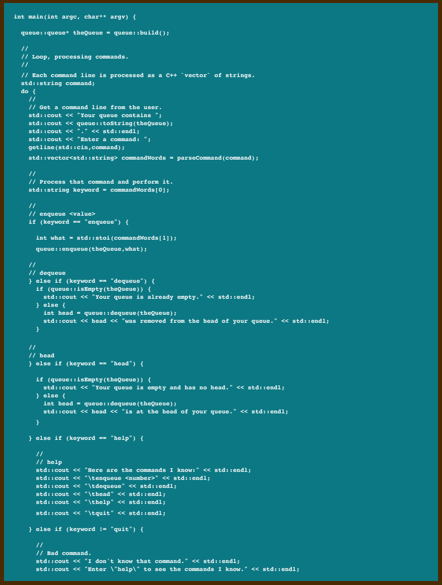
```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

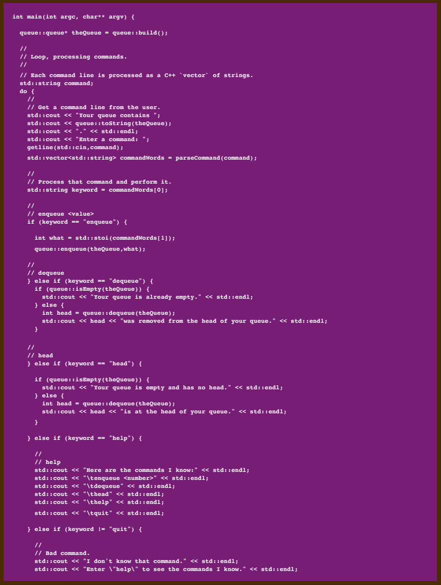
- The code you write looks just like file I/O.
- To "send" you actually **`write`** to an open Unix *file descriptor*

client.c



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

server.c

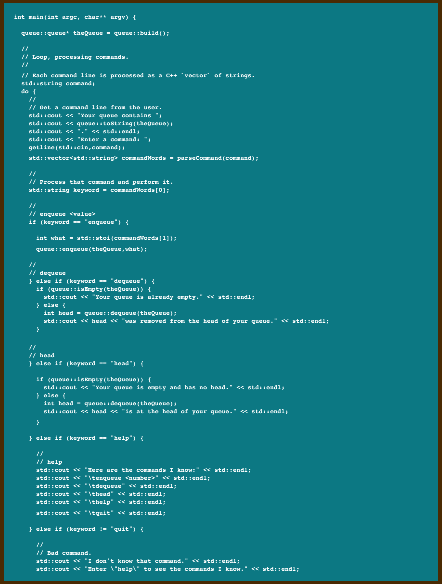


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

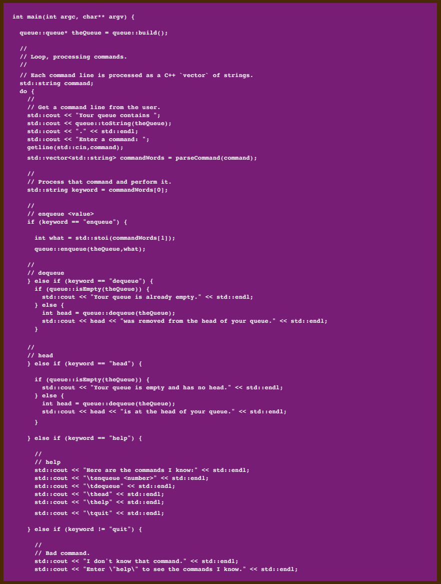
- The code you write looks just like file I/O.
- To "send" you actually `write` data to an open Unix *file descriptor.*
- To "receive" you actually `read` data from that same open f.d.

client.c



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

server.c

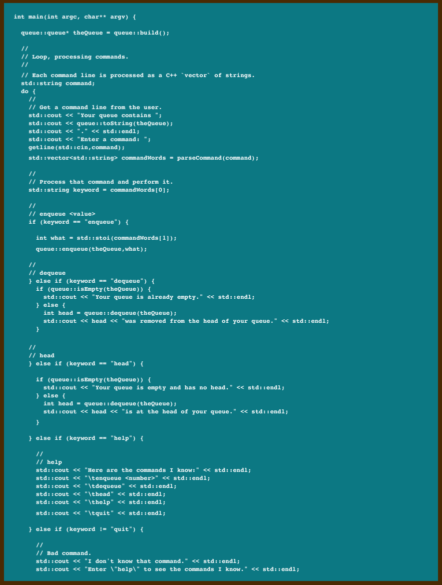


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

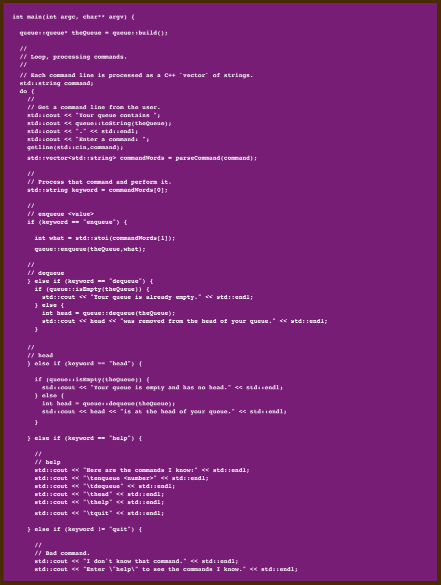
- The code you write looks just like file I/O.
- To "send" you actually `write` data to an open Unix *file descriptor.*
- To "receive" you actually `read` data from that same open f.d.
- That file descripter is a "socket connection".
- The data are just bytes of `char` strings.

client.c



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

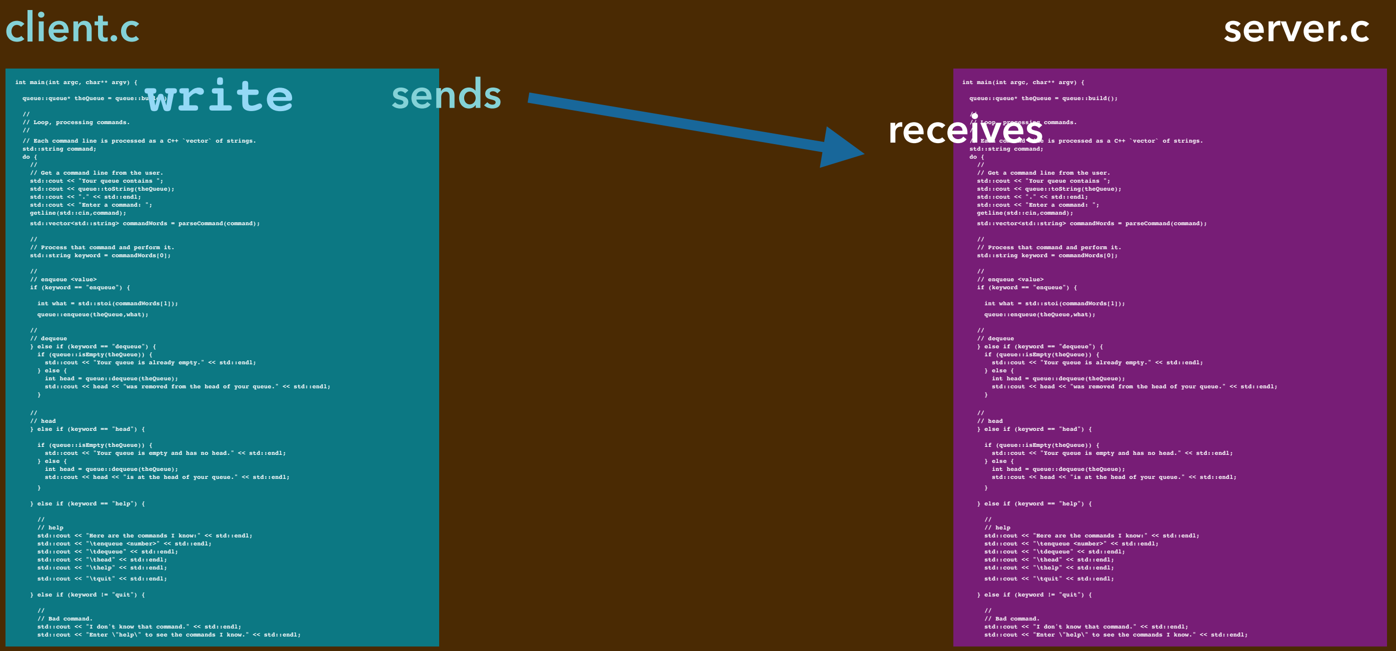
server.c



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- The code you write looks just like file I/O.
- To "send" you actually `write` data to an open Unix *file descriptor.*
- To "receive" you actually `read` data from that same open f.d.
- That file descripter is a "socket connection".
- The data are just bytes of `char` strings.

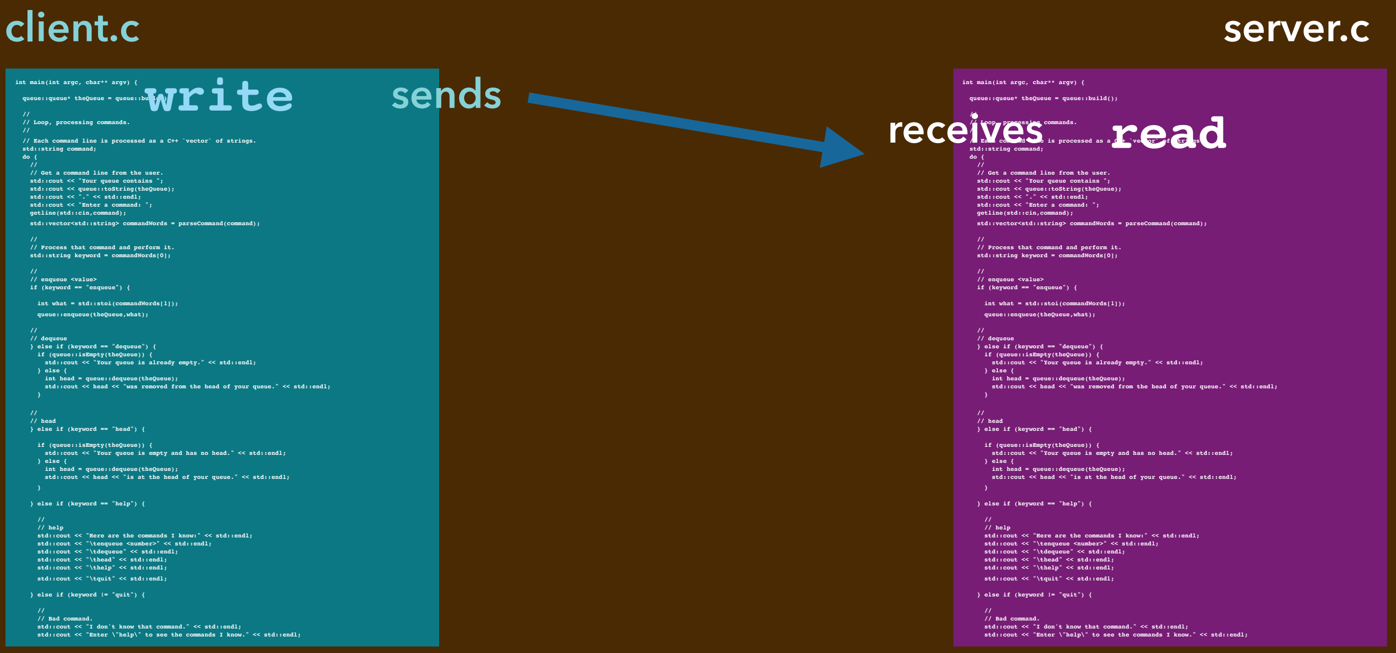


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- The code you write looks just like file I/O.
- To "send" you actually **`write`** data to an open Unix *file descriptor.*
- To "receive" you actually **`read`** data from that same open f.d.
- That file descripter is a "socket connection".
- The data are just bytes of `char` strings.

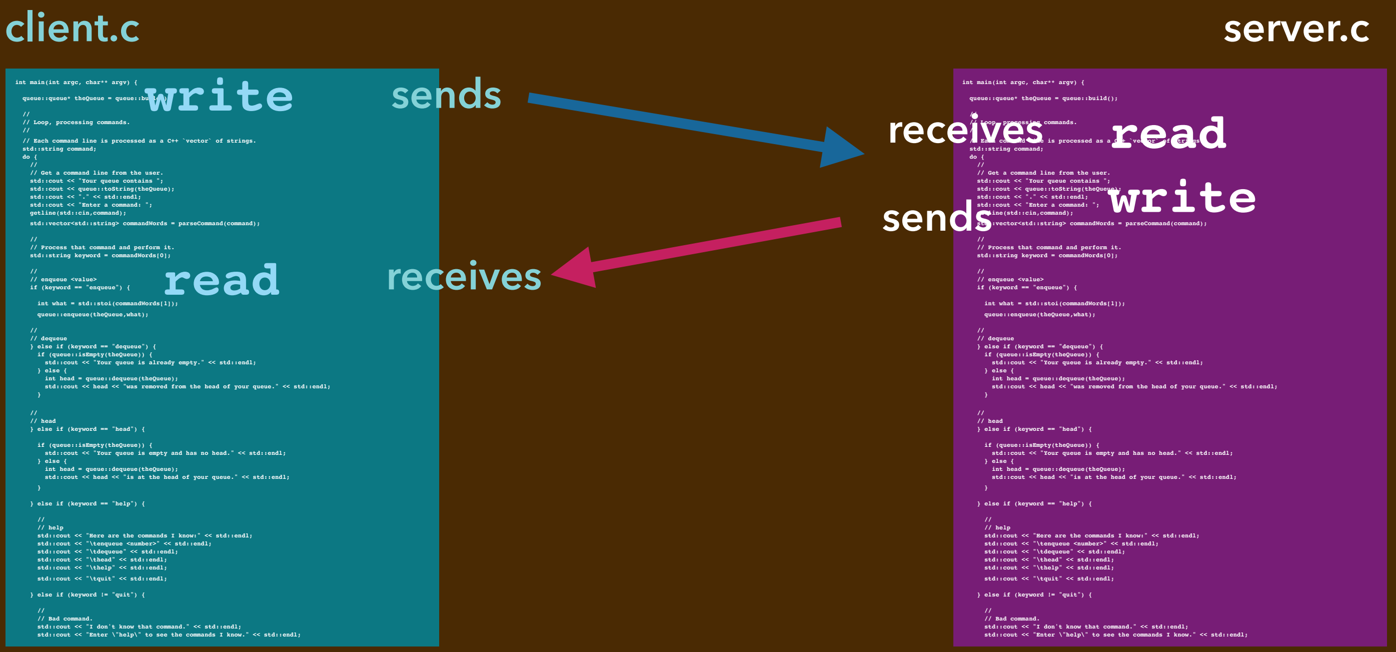


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- The code you write looks just like file I/O.
- To "send" you actually `write` data to an open Unix *file descriptor.*
- To "receive" you actually `read` data from that same open f.d.
- That file descripter is a "socket connection".
- The data are just bytes of `char` strings.

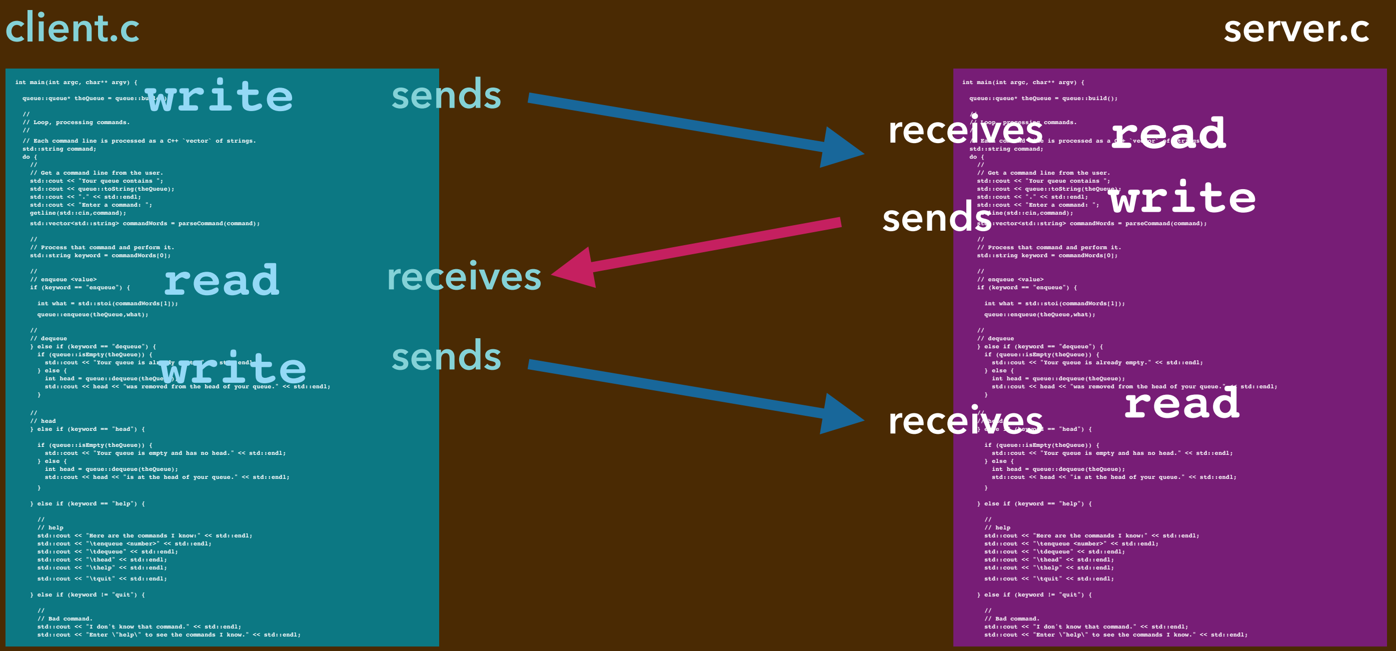


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

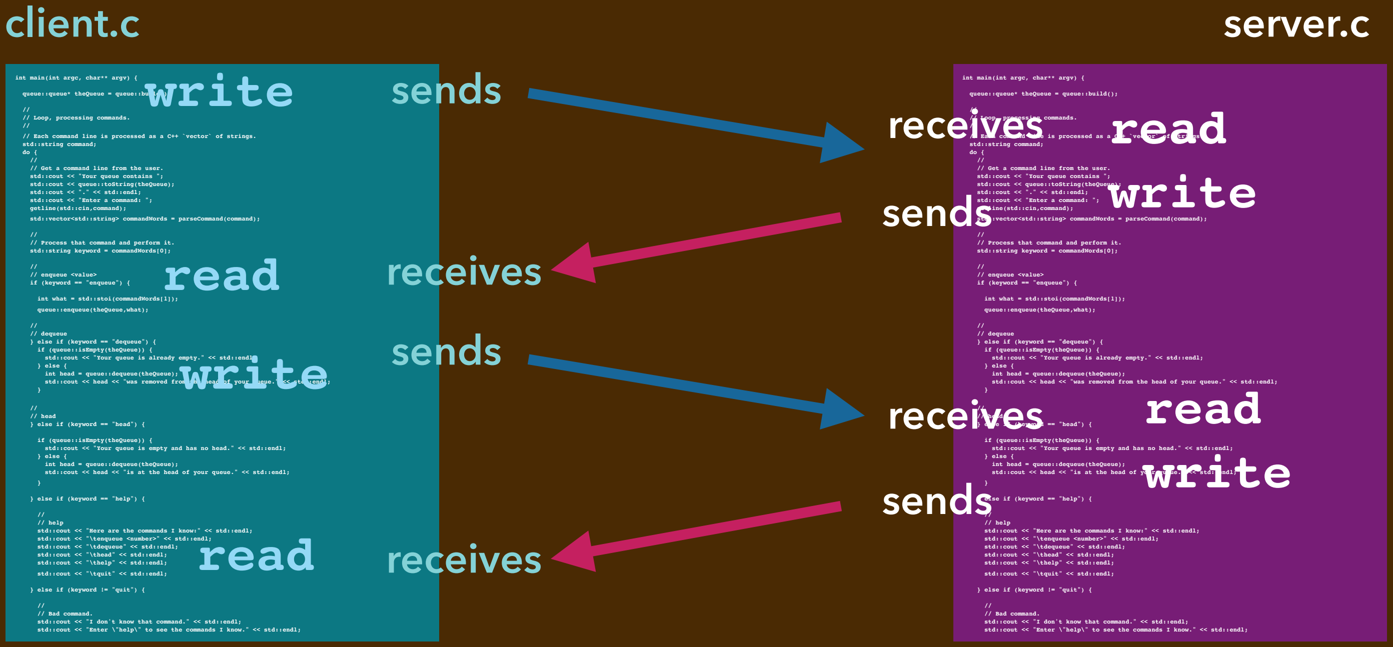
- The code you write looks just like file I/O.
- To "send" you actually `write` data to an open Unix *file descriptor.*
- To "receive" you actually `read` data from that same open f.d.
- That file descripter is a "socket connection".
- The data are just bytes of `char` strings.



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- The code you write looks just like file I/O.
- To "send" you actually `write` data to an open Unix *file descriptor.*
- To "receive" you actually `read` data from that same open f.d.
- That file descripter is a "socket connection".
- The data are just bytes of `char` strings.

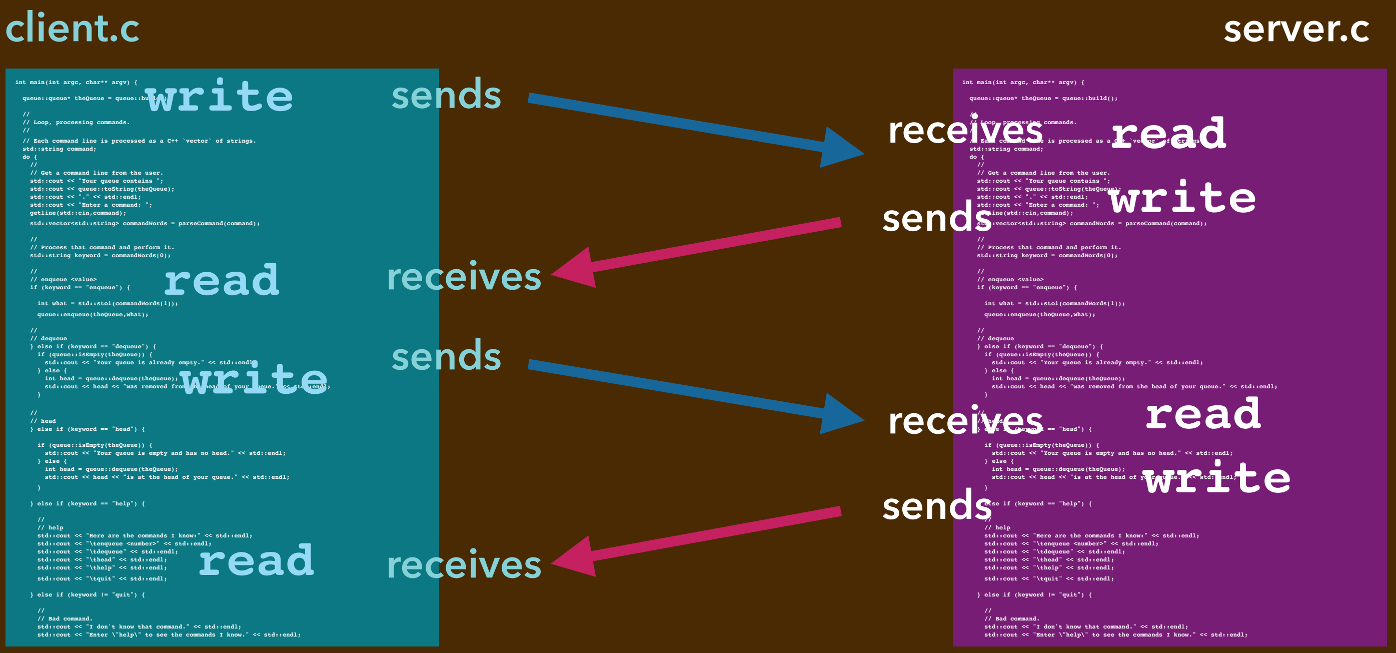


```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

SOCKETS LIBRARY

- By mimicking file I/O, socket uses is fairly easy.
- Trickiest part of the coding is
  - how the server binds to a port, then listens for and accepts connections
  - how the client connects to the server



```
int main(int argc, char** argv) {

  queue::queue* theQueue = queue::build();

  //
  // Loop, processing commands.
  //
  // Each command line is processed as a C++ `vector` of strings.
  std::string command;
  do {
    //
    // Get a command line from the user.
    std::cout << "Your queue contains ";
    std::cout << queue::toString(theQueue);
    std::cout << "." << std::endl;
    std::cout << "Enter a command: ";
    getline(std::cin,command);
    std::vector<std::string> commandWords = parseCommand(command);

    //
    // Process that command and perform it.
    std::string keyword = commandWords[0];

    //
    // enqueue <value>
    if (keyword == "enqueue") {

      int what = std::stoi(commandWords[1]);
      queue::enqueue(theQueue,what);

    //
    // dequeue
    } else if (keyword == "dequeue") {
      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is already empty." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "was removed from the head of your queue." << std::endl;
      }

    //
    // head
    } else if (keyword == "head") {

      if (queue::isEmpty(theQueue)) {
        std::cout << "Your queue is empty and has no head." << std::endl;
      } else {
        int head = queue::dequeue(theQueue);
        std::cout << head << "is at the head of your queue." << std::endl;
      }

    } else if (keyword == "help") {

      //
      // help
      std::cout << "Here are the commands I know:" << std::endl;
      std::cout << "\tenqueue <number>" << std::endl;
      std::cout << "\tdequeue" << std::endl;
      std::cout << "\thead" << std::endl;
      std::cout << "\thelp" << std::endl;
      std::cout << "\tquit" << std::endl;

    } else if (keyword != "quit") {

      //
      // Bad command.
      std::cout << "I don't know that command." << std::endl;
      std::cout << "Enter \"help\" to see the commands I know." << std::endl;
```

ECHO SERVER CODE

```
// create the listener socket
listenfd = socket(AF_INET, SOCK_STREAM, 0);
...
// BIND to the port on that machine
struct sockaddr_in serveraddr;
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons((unsigned short)port);
bind(listenfd,&serveraddr, sizeof(struct sockaddr));
...
// set up to LISTEN
listen(listenfd, CONNECTIONS);
...
// ACCEPT a client connection
struct sockaddr_in clientaddr;
int connfd = accept(listenfd, &clientaddr, &acceptlen);

read(connfd,...);
write(connfd,...);
read(connfd,...);
write(connfd,...);
...
```

ECHO CLIENT CODE

```
// create the connection socket
struct hostent *hp;
hp = gethostbyname(host);
clientfd = socket(AF_INET, SOCK_STREAM, 0)

...
// use the server info to CONNECT
struct hostent *hp;
hp = gethostbyname(host);
struct sockaddr_in serveraddr;
serveraddr.sin_family = AF_INET;
bcopy(hp->h_addr_list[0],&serveraddr.sin_addr.s_addr,...);
serveraddr.sin_port = htons(port);
connect(clientfd, &serveraddr, sizeof(struct sockaddr_in));
...

write(clientfd,...);
read(clientfd,...);
write(clientfd,...);
read(clientfd,...);
...
```

INTERNET SERVICES

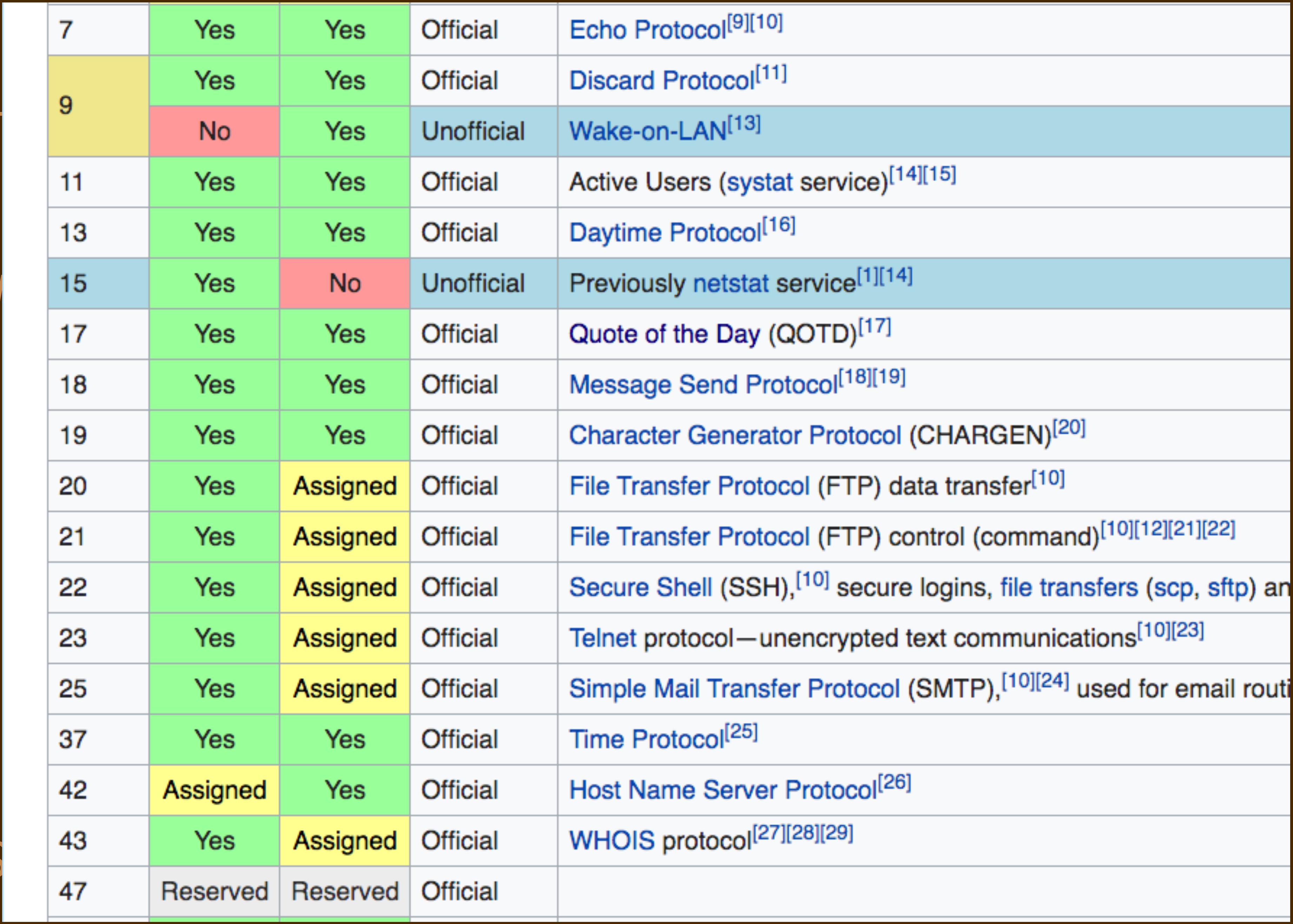
- The Domain Name Service (DNS) provides name => IP address lookup
  - www.reed.edu => 134.10.50.30
- Well-known services have well-known port numbers
  - SSH uses port 22
  - HTTP uses port 80 and 8080
  - DNS uses port 53
  - SMTP (email server) uses port 25
  - Telnet uses port 23
  - QOTD uses port 17
- See: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

IN

▸ T

▸ W

▸ S



7	Yes	Yes	Official	Echo Protocol[9][10]
9	Yes	Yes	Official	Discard Protocol[11]
	No	Yes	Unofficial	Wake-on-LAN[13]
11	Yes	Yes	Official	Active Users (systat service)[14][15]
13	Yes	Yes	Official	Daytime Protocol[16]
15	Yes	No	Unofficial	Previously netstat service[1][14]
17	Yes	Yes	Official	Quote of the Day (QOTD)[17]
18	Yes	Yes	Official	Message Send Protocol[18][19]
19	Yes	Yes	Official	Character Generator Protocol (CHARGEN)[20]
20	Yes	Assigned	Official	File Transfer Protocol (FTP) data transfer[10]
21	Yes	Assigned	Official	File Transfer Protocol (FTP) control (command)[10][12][21][22]
22	Yes	Assigned	Official	Secure Shell (SSH),[10] secure logins, file transfers (scp, sftp) an
23	Yes	Assigned	Official	Telnet protocol—unencrypted text communications[10][23]
25	Yes	Assigned	Official	Simple Mail Transfer Protocol (SMTP),[10][24] used for email routi
37	Yes	Yes	Official	Time Protocol[25]
42	Assigned	Yes	Official	Host Name Server Protocol[26]
43	Yes	Assigned	Official	WHOIS protocol[27][28][29]
47	Reserved	Reserved	Official	