Unix
Bash
C
GNU
Systems

Software Systems

Lectures Week 10

Introduction to Systems Programming 2

(Files, Networks, Inter process communication via CGI)

Prof. Joseph Vybihal

Computer Science

McGill University

Part A

Sequential, Block, and Random Access Files

Readings: https://www.thoughtco.com/random-access-file-handling-958450 and http://www.dummies.com/programming/c/basics-of-sequential-file-access-in-c-programming/

Basic File Organization

- Stream
  - Defined as a contiguous series of bytes, such that, traversal of the file occurs only in one direction, from the beginning of the file to the end of the file, one byte at a time.
- Block
  - The file is understood to be composed of units of equal sized data stored randomly. All the information is structured similarly, in units of N-bytes. The file can be traversed in any direction.
- Line
  - Data is organized into unequal byte sized units. Each unit needs a terminating symbol. File traversal occurs only in one direction, from beginning to end, looking for these markers.

Basic File Terminology

- Sequential access files
  - Stream and Line files are examples.
  - The fundamental property is that these files are accessed in only one direction, from the beginning to the end one byte at a time.
- Random access files
  - Block file is an example.
  - The fundamental property is that a data unit can be accessed randomly. These files operate analogously to arrays of struct.

Basic File Types

- Text (like .txt) or Binary (like a.out)
  - Text files and compiled programs are examples of sequential access files.
  - We have already seen how to read, write and append to Text files. Our lecture will focus on CSV and RAF. A compiler course will cover reading and writing a.out files.
- CSV (comma separated vector)
  - Many file types fall under the Line organization technique: .csv .ini .json, to name a few.
- RAF (random access file)
  - Examples would include: databases, caches, and quick access files.

The CSV File

Common uses for CSV are:

- As a configuration file
- As a simple database file

Other files that are Line based and used for similar purposes include:

- INI as strictly a configuration file
- JSON as either configuration, simple databases, and temporary information transfer format

The CSV File

Format:

- Record
  - Defined to be a Line of data terminating with a carriage return character.
- Field
  - Defined to be a sequence of characters terminating with the comma character or the carriage return character.

Example:

User,Password,FirstName,LastName

Jvybihal,abc123,Joseph,Vybihal

Mary.our,xyzAb!,Mary,Lou

Notice that the carriage return character and the comma character become reserved words that cannot be used as data. Escape characters can be used to overcome this limitation.

Example

```
#include<stdio.h>   #include<stdlib.h>   // in reality these needs to be different lines

char buffer[2000];  // some large number to handle long lines
char user[100], passw[100], firstName[100], lastName[100]; // fields with large sizes
int bufferIndex=0;

FILE *csv = fopen("file.csv","rt");
if (csv == NULL) exit(1);

fgets(buffer,1999,csv);
while(!feof(csv)) {
   bufferIndex = nextField(buffer, bufferIndex, user);
   bufferIndex = nextField(buffer, bufferIndex, passw);
   bufferIndex = nextField(buffer, bufferIndex, firstName);
   bufferIndex = nextField(buffer, bufferIndex, lastName);

   fgets(buffer,1999,csv);
}
```

Example (cont.)

```
int nextField(char theBuffer[], int index, char theField[]) {
        // theBuffer and theField use call-by-reference
        // index uses call-by-value
    int x, y=0;

    for(x=index; theBuffer[x]!='\0' && theBuffer[x]!=','; x++) {
        theField[y] = theBuffer[x];
        y++;
    }

    theField[y] = '\0'; // terminate it like a string

    return x+1; // as the new buffer index
}
```

RAF and Binary

To be able to move randomly within a file each unit of information needs to have a standard size.

RAF files can be text or binary, however binary files have the advantage of being faster to process because data type conversions from internal storage and disk storage can be skipped.

We will look only at binary RAF files.

The RAF Binary File

Takes advantage of:

- The C struct statement
- The stdio.h commands fread, fwrite & fseek to read/write entire struct data structures in one action
- Since struct structures are of the same byte-size we can compute the distance:
  - struct STUD array[10];
  - struct STUD *p = array + (2 * sizeof(struct STUD);
    - Which is the same as saying: p = array[2];

Example

```
#include<stdio.h>    #include<stdlib.h>    // in reality these need to be different lines

struct STUD {
   char name[100];
   int age;
   float GPA;
};

struct STUD x; // assume we put data in this
FILE *p = fopen("file.raf", "wb"); // write binary,   "rb" to read, "ab" to append
if (p == NULL) exit(1);

fwrite(x, sizeof(struct STUD), 1, p); // destructive, overwrites previous file, since "wb"
```

fwrite(struct *, sizeof, int repeat, FILE *);
fread (struct *, sizeof, int repeat, FILE *);

These two commands, by themselves, are sequential.

Example

```
// Writing to a RAF database

#include<stdio.h>    #include<stdlib.h>    // in reality these need to be different lines

struct STUD {
   char name[100];
   int age;
   float GPA;
} students[100];   // assume populated with values

FILE *p = fopen("database.raf", "wb");
if (p == NULL) exit(1);

For(x=0; x<100; x++)   fwrite(students[x], sizeof(struct STUD), 1, p);

fclose(p);
```

Example

```
// Reading from a RAF database

#include<stdio.h>    #include<stdlib.h>    // in reality these need to be different lines

struct STUD {
   char name[100];
   int age;
   float GPA;
} students[100];

int x = 0;
FILE *p = fopen("database.raf", "rb");
if (p == NULL) exit(1);

do {fread(students[x], sizeof(struct STUD), 1, p); x++;}   while(!feof(p));

fclose(p);
```

Random Access

The fseek command permits random motion within a file.

Syntax:

- int fseek(FILE *stream, long offset, int whence);
- void rewind(FILE *stream);

Where:

- Whence = SEEK_SET jump from beginning of file
- Whence = SEEK_CUR jump from current position in file
- Whence = SEEK_END jump from the end of the file
- Jumps are measured in OFFSET bytes. Positive numbers move forward through the file, negative numbers backwards.

Example

```
fseek(fp, 100, SEEK_SET);      // seek to the 100th byte of the file
fseek(fp, -30, SEEK_CUR);      // seek backward 30 bytes from the current pos
fseek(fp, -10, SEEK_END);      // seek to the 10th byte before the end of file

fseek(fp, 0, SEEK_SET);        // seek to the beginning of the file
rewind(fp);                    // seek to the beginning of the file
```

Question

If I have a database of students, struct STUD, and I would like to load into memory the 10^{th} student, how would I do this?

Question

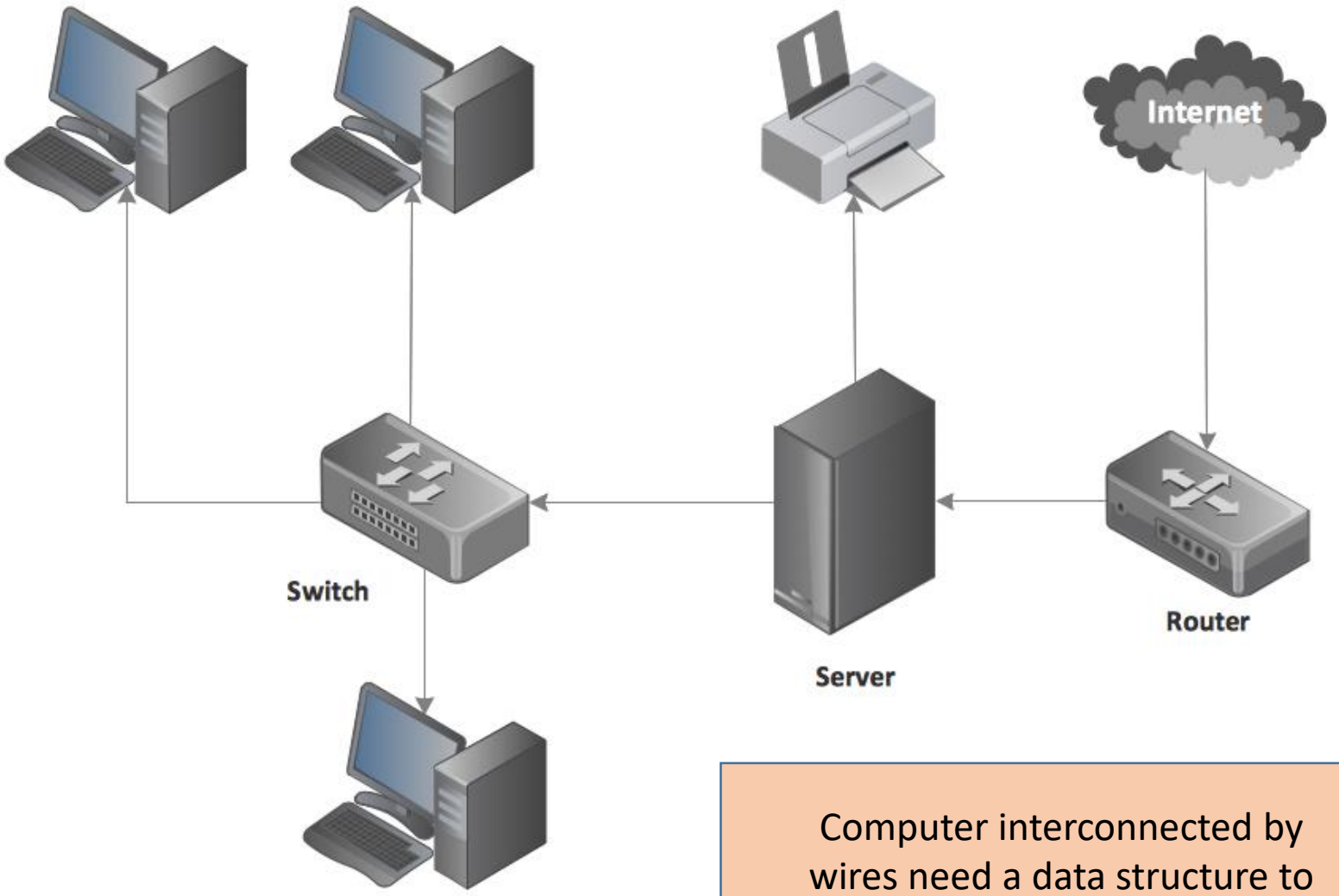
Suggest how we might do this:

I have a sorted RAF of students, STRUCT STUD, sorted by student's last name. I want to find someone by their last name quickly. Assume last names are unique.

Part B

Networks and Websites

Sample Network Diagram



Computer interconnected by wires need a data structure to send messages to one another.

Every computer in a network has a unique ID number assigned to it, called an address

Surprisingly, the payload is String type.

Data Packet

Destination MAC Address	Source MAC Address	Destination IP Address	Source IP Address	Payload	CRC

The “payload” is the message being send from the source computer to the destination computer

Example

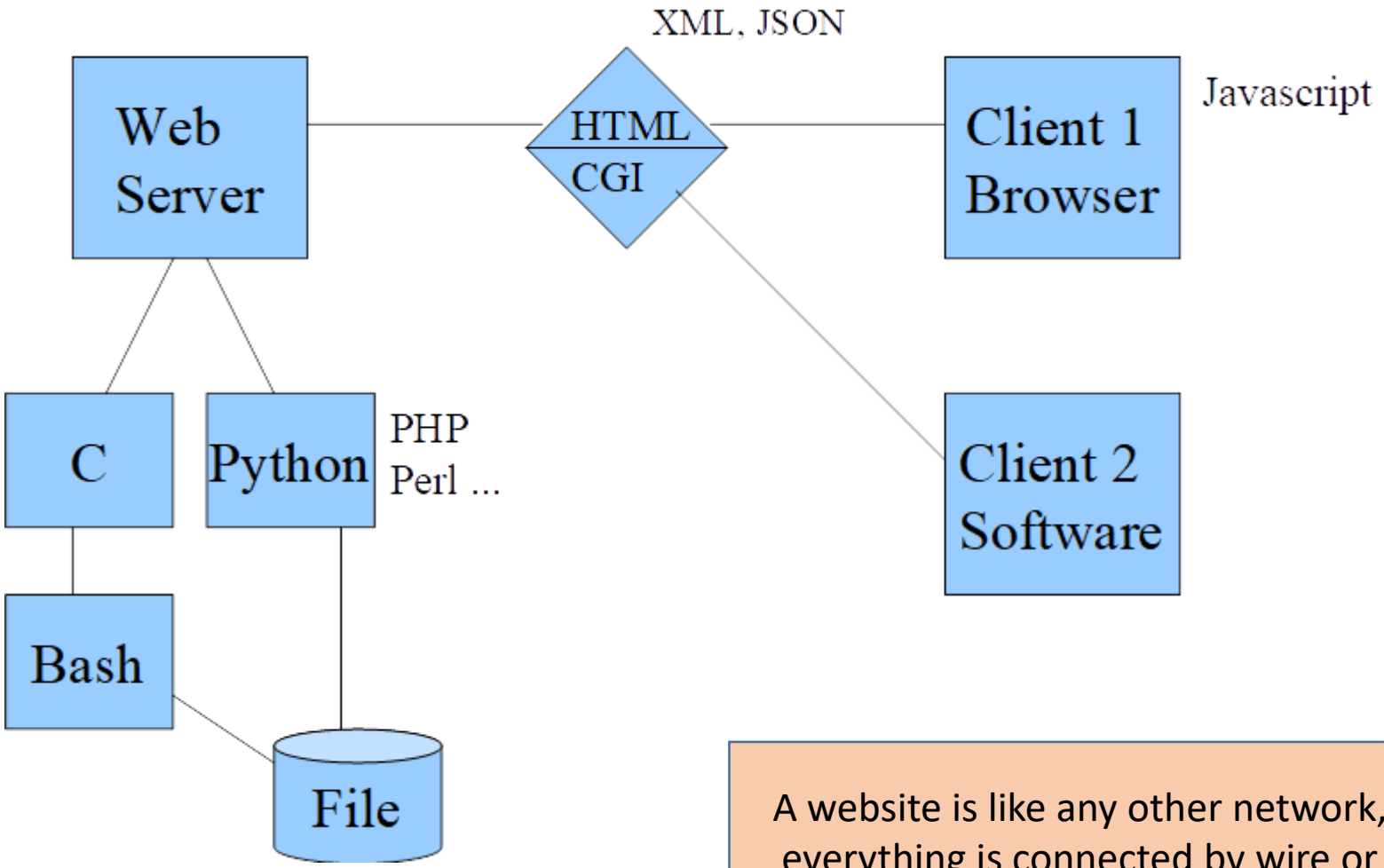
How would three computers connected to each other in a ring shape communicate?

How would three computers connected to a server in a star shape communicate?

The "payload" is formatted as HTML, XML, JSON or other

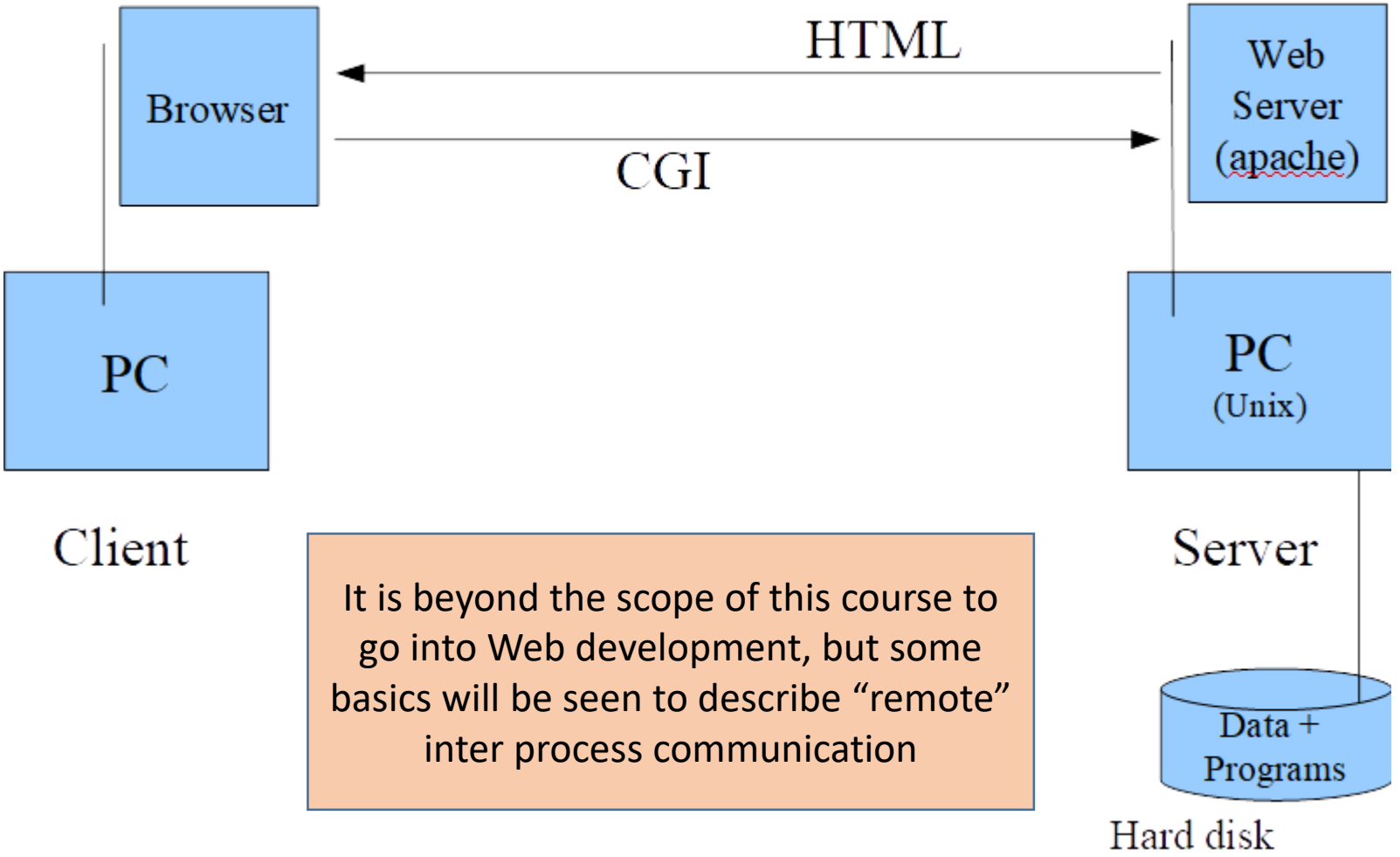
Websites

The communication rules, "protocol" is called CGI



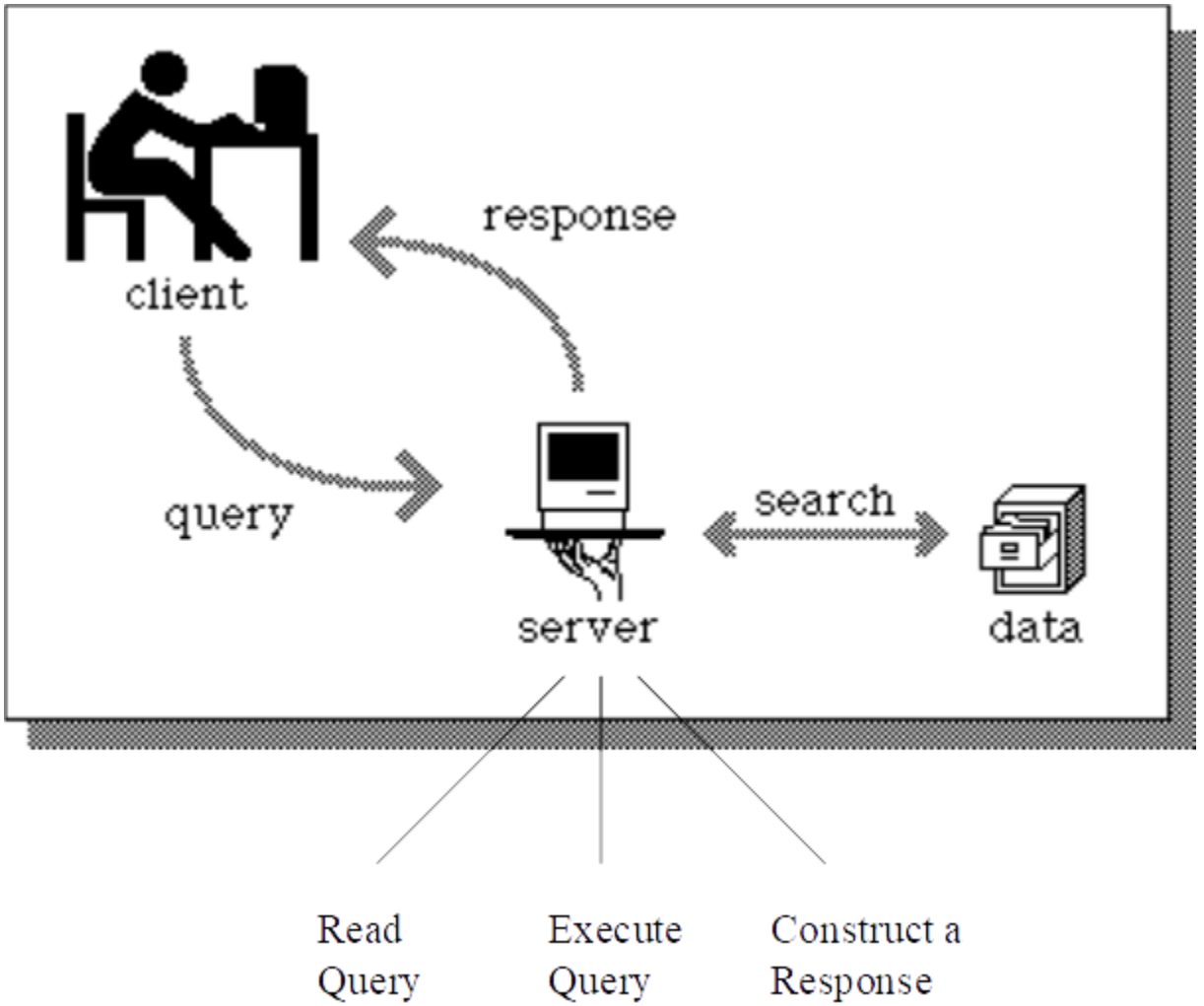
A website is like any other network, everything is connected by wire or some other medium

CGI
(Common Gateway Interface)



It is beyond the scope of this course to go into Web development, but some basics will be seen to describe "remote" inter process communication

Client Server
(at the heart of the CGI protocol)



Trottier Web Server

- In your home directory create:
  - MKDIR public_html
- Make sure the world can read and execute from the directory
  - CHMOD 755 public_html
- Create your home page
  - VI index.html
  - CHMOD 755 index.html
- Look at your home page
  - URL: http://www.cs.mcgill.ca/~your_user_name

Demo

Show teacher's public_html

- First with winscp
- Then with browser
- Last, let us create it together

Part C

Inter process communication with CGI

Bare Bones HTML

- Chapter 5 from the textbook
- HTML: http://www.w3schools.com/
- HTML: http://www.htmlgoodies.com/

By Example 1

```
<html>
        <head>
                <title>Title of page</title>
        </head>

        <body>
                This is my first homepage.
                <b>This text is bold</b>
        </body>
</html>
```

Is a text file (a script)

By Example 1

Created in sections

```
<html>
        <head>
                <title>Title of page</title>
        </head>

        <body>
                This is my first homepage.
                <b>This text is bold</b>
        </body>
</html>
```

By Example 1

```
<html>
      <head>
            <title>Title of page</title>
      </head>

      <body>
            This is my first homepage.
            <b>This text is bold</b>
      </body>
</html>
```

Uses tag commands

At the command line

```
vi bla.html
mkdir public_html       (in home directory)
chmod a+rx public_html
cp bla.html public_html
```

```
Browser html-command-bar:
Http://www.cs.mcgill.ca/~jvybihal/bla.html
```

```
Browser: file/open
```

CGI

Uses the <form> tag

```
<form name="input" action="script.py‟ method="get">
  <b>Username:<\b> <input type="text" name="user"> <br />


  Type of student: <br />


  <input type="radio" name="student" value="ugrad">Undergrad<br />


  <input type="radio" name="student" value="grad">Graduate<br />

  Graduating : <input type="checkbox" name="graduating"> <br />
  <input type="submit" value="Press Here">
</form>
```

Normal HTML
plus <input>
and <form>

CGI

Uses the <form> tag

```
<form name="input" action="script.py‟ method="get">
   <b>Username:<\b> <input type="text" name="user"> <br />


   Type of student: <br />


   <input type="radio" name="student" value="ugrad">Undergrad<br />


   <input type="radio" name="student" value="grad">Graduate<br />

   Graduating : <input type="checkbox" name="graduating"> <br />
   <input type="submit" value="Press Here">
</form>
```

A form can be “named”.
The **action** is the program in your public_html directory.
The **method** is the protocol

CGI

Uses the <form> tag

```
<form name="input" action="script.py‟ method="get">
   <b>Username:<\b> <input type="text" name="user"> <br />


   Type of student: <br />


   <input type="radio" name="student" value="ugrad">Undergrad<br />


   <input type="radio" name="student" value="grad">Graduate<br />

   Graduating : <input type="checkbox" name="graduating"> <br />
   <input type="submit" value="Press Here">
</form>
```

If no URL is provided in the **action** attribute, then the webpage's URL is assumed.

CGI Payload = "URL/script.py?student=ugrad&graduating=true"

CGI

Uses the <form> tag

```
<form name="input" action="script.py" method="get">
  <b>Username:<\b> <input type="text" name="user"> <br />


  Type of student: <br />


  <input type="radio" name="student" value="ugrad">Undergrad<br />


  <input type="radio" name="student" value="grad">Graduate<br />

  Graduating : <input type="checkbox" name="graduating"> <br />
  <input type="submit" value="Press Here">
</form>
```

There are two possible protocols:
GET and POST


```
<html>
  <head>
    <title>Form Example</title>
  </head>

  <body>

    <form action="a.out" method="get">
      <input type="text" name="name">
      <input type="submit" value="Submit">
    </form>

  </body>
</html>
```

GET vs POST

- Post :
  - Data placed into stdin
    - Readable by scanf, gets, etc.
- Get:
  - Data placed into shell memory
    - Readable by shell memory commands (`getenv()`)

Interfacing with C and GET

```
#include <stdlib.h>

 void main() {
    int a, b, c;
    char array[100];
    char *data = getenv("QUERY_STRING");                    // as string

    printf("%s\n", data);

    sscanf(data,"x=%d&y=%d",&a,&b);                         // as types

    for(c=0;c<length(data); c++) array[c] = *(data+c); // as char
 }
```

The payload

The string can be parsed in other ways as well: using strtok().

Interfacing with C and POST #1

Payload length

Payload data sent to STDIN

```
#include <stdlib.h>
char string[200];
char c;
int a = 0;
int n = atoi(getenv("CONTENT_LENGTH"));

fgets(string,n,stdin);    // reading it like a file
```

Interfacing with C and POST #2

```
#include <stdlib.h>
char string[200];
char c;
int a = 0;
int n = atoi(getenv("CONTENT_LENGTH"));

while ((c = getchar()) != EOF && a<n)   // reading char by char
{
      if (a < 200)
      {
            if (c!='+') string[a]=c;    // converting chars
            else string[a]=' ';
            a++;
      }
}
String[a] = '\0';
```

Output to Browser

```
#include <stdio.h>   #include <stdlib.h>
int main(void)
{
        FILE *f = fopen("data.txt","r");
        int ch;

        printf("Content-Type:text/html\n\n");         // CGI output tag
        printf("<html>");

        if (f==NULL)
        {
                printf("<head><title>ERROR</title></head>");
                printf("<body><p>Unable to open file!</p></body>");
        }
        else
        {
                while((ch=fgetc(f)) != EOF) putchar(ch);

                fclose(f);
        }

        printf("</html>");

        return 0;
}
```

Notice that once the CGI output tag is printed then we can simpley output HTML, CGI, JS, etc. and the browser will understand

Server Errors and Help

Server run-time errors can be viewed from:
http://cgi.cs.mcgill.ca/cgi-bin/geterrors.cgi

To find help:
http://www.cs.mcgill.ca/docs/labs/webservers

McGill Servers

mimi.cs.mcgill.ca – a SOCS server
freebsd.cs.mcgill.ca – default web server
cgi.cs.mcgill.ca – updated script server

Note: every server is installed with it's own OS, libraries, and software. They are not usually identical.

Note: make sure to compile on freebsd or cgi.