

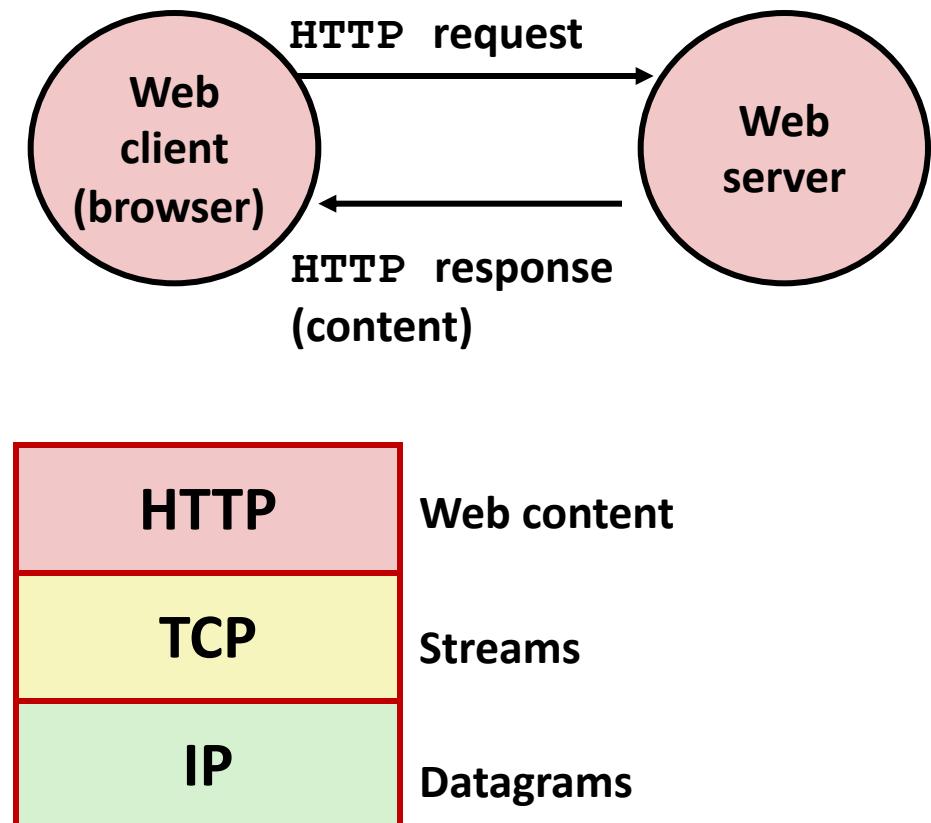
# Network Programming: Part II

**CS230 System Programming  
16<sup>th</sup> Lecture**

**Instructors:**  
**Jongse Park**

# Web Server Basics

- Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
  - Client and server establish TCP connection
  - Client requests content
  - Server responds with requested content
  - Client and server close connection (eventually)
- Current version is HTTP/1.1
  - RFC 2616, June, 1999.



<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

# Web Content

## ■ Web servers return *content* to clients

- *content*: a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type

## ■ Example MIME types

▪ text/html	HTML document
▪ text/plain	Unformatted text
▪ image/gif	Binary image encoded in GIF format
▪ image/png	Binary image encoded in PNG format
▪ image/jpeg	Binary image encoded in JPEG format

You can find the complete list of MIME types at:

<http://www.iana.org/assignments/media-types/media-types.xhtml>

# Static and Dynamic Content

- The content returned in HTTP responses can be either *static* or *dynamic*
  - *Static content*: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
    - Request identifies which content file
  - *Dynamic content*: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
    - Request identifies file containing executable code
- Bottom line: *Web content is associated with a file that is managed by the server*

# URLs and how clients and servers use them

- Unique name for a file: URL (Universal Resource Locator)
- Example URL: **http://www.cmu.edu:80/index.html**
- Clients use *prefix* (**http://www.cmu.edu:80**) to infer:
  - What kind (protocol) of server to contact (HTTP)
  - Where the server is (`www.cmu.edu`)
  - What port it is listening on (80)
- Servers use *suffix* (**/index.html**) to:
  - Determine if request is for static or dynamic content.
    - No hard and fast rules for this
    - One convention: executables reside in `cgi-bin` directory
  - Find file on file system
    - Initial “/” in suffix denotes home directory for requested content.
    - Minimal suffix is “/”, which server expands to configured default filename (usually, `index.html`)

# HTTP Requests

- HTTP request is a *request line*, followed by zero or more *request headers*
- Request line: <method> <uri> <version>
  - <method> is one of GET, POST, OPTIONS, HEAD, PUT, DELETE, or TRACE
  - <uri> is typically URL for proxies, URL suffix for servers
    - A URL is a type of URI (Uniform Resource Identifier)
    - See <http://www.ietf.org/rfc/rfc2396.txt>
  - <version> is HTTP version of request (HTTP/1.0 or HTTP/1.1)
- Request headers: <header name>: <header data>
  - Provide additional information to the server

# HTTP Responses

- HTTP response is a *response line* followed by zero or more *response headers*, possibly followed by *content*, with blank line ("`\r\n`") separating headers from content.

- Response line:

`<version> <status code> <status msg>`

- `<version>` is HTTP version of the response
- `<status code>` is numeric status
- `<status msg>` is corresponding English text
  - 200 OK Request was handled without error
  - 301 Moved Provide alternate URL
  - 404 Not found Server couldn't find the file

- Response headers: `<header name>: <header data>`

- Provide additional information about response
- Content-Type: MIME type of content in response body
- Content-Length: Length of content in response body

# Example HTTP Transaction

```
whaleshark> telnet www.cmu.edu 80
Trying 128.2.42.52...
Connected to WWW-CMU-PROD-VIP.ANDREW.cmu.edu.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.cmu.edu
```

```
HTTP/1.1 301 Moved Permanently
Date: Wed, 05 Nov 2014 17:05:11 GMT
Server: Apache/1.3.42 (Unix)
Location: http://www.cmu.edu/index.shtml
Transfer-Encoding: chunked
Content-Type: text/html; charset=...
```

```
15c
<HTML><HEAD>
...
</BODY></HTML>
0
Connection closed by foreign host.
```

Client: open connection to server  
Telnet prints 3 lines to terminal

Client: request line  
Client: required HTTP/1.1 header  
Client: empty line terminates headers

Server: response line  
Server: followed by 5 response headers  
Server: this is an Apache server  
Server: page has moved here  
Server: response body will be chunked  
Server: expect HTML in response body  
Server: empty line terminates headers  
Server: first line in response body  
Server: start of HTML content

Server: end of HTML content  
Server: last line in response body  
Server: closes connection

- HTTP standard requires that each text line end with “`\r\n`”
- Blank line (“`\r\n`”) terminates request and response headers

# Example HTTP Transaction, Take 2

```
whaleshark> telnet www.cmu.edu 80
Trying 128.2.42.52...
Connected to WWW-CMU-PROD-VIP.ANDREW.cmu.edu.
Escape character is '^]'.
GET /index.shtml HTTP/1.1
Host: www.cmu.edu

HTTP/1.1 200 OK
Date: Wed, 05 Nov 2014 17:37:26 GMT
Server: Apache/1.3.42 (Unix)
Transfer-Encoding: chunked
Content-Type: text/html; charset=...

1000
<html ..>
...
</html>
0
Connection closed by foreign host.
```

Client: open connection to server  
Telnet prints 3 lines to terminal

Client: request line  
Client: required HTTP/1.1 header  
Client: empty line terminates headers  
Server: response line  
Server: followed by 4 response headers

Server: empty line terminates headers  
Server: begin response body  
Server: first line of HTML content

Server: end response body  
Server: close connection

# Tiny Web Server

## ■ Tiny Web server described in text

- Tiny is a sequential Web server
- Serves static and dynamic content to real browsers
  - text files, HTML files, GIF, PNG, and JPEG images
- 239 lines of commented C code
- Not as complete or robust as a real Web server
  - You can break it with poorly-formed HTTP requests (e.g., terminate lines with “\n” instead of “\r\n”)

# Tiny Operation

- Accept connection from client
- Read request from client (via connected socket)
- Split into <method> <uri> <version>
  - If method not GET, then return error
- If URI contains “c<sub>gi</sub>-bin” then serve dynamic content
  - (Would do wrong thing if had file “abc<sub>gi</sub>-bingo.html”)
  - Fork process to execute program
- Otherwise serve static content
  - Copy file to output

# Tiny Serving Static Content

```
void serve_static(int fd, char *filename, int filesize)
{
    int srcfd;
    char *srcp, filetype[MAXLINE], buf[MAXBUF];

    /* Send response headers to client */
    get_filetype(filename, filetype);
    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    sprintf(buf, "%sServer: Tiny Web Server\r\n", buf);
    sprintf(buf, "%sConnection: close\r\n", buf);
    sprintf(buf, "%sContent-length: %d\r\n", buf, filesize);
    sprintf(buf, "%sContent-type: %s\r\n\r\n", buf, filetype);
    Rio_writen(fd, buf, strlen(buf));

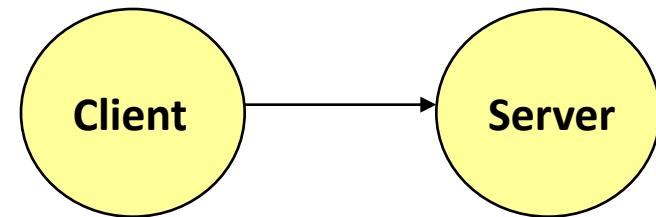
    /* Send response body to client */
    srcfd = Open(filename, O_RDONLY, 0);
    srcp = Mmap(0, filesize, PROT_READ, MAP_PRIVATE, srcfd, 0);
    Close(srcfd);
    Rio_writen(fd, srcp, filesize);
    Munmap(srcp, filesize);
}
```

tiny.c

# Serving Dynamic Content

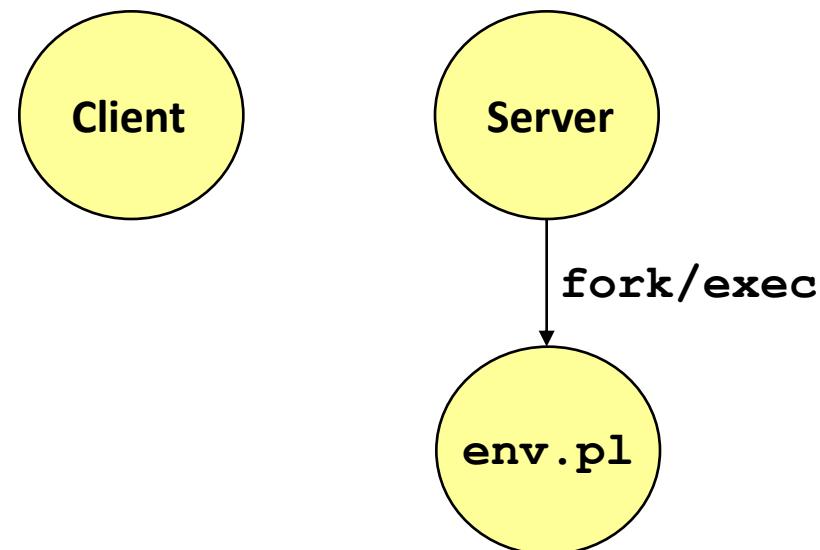
- Client sends request to server
- If request URI contains the string “/cgi-bin”, the Tiny server assumes that the request is for dynamic content

GET /cgi-bin/env.pl HTTP/1.1



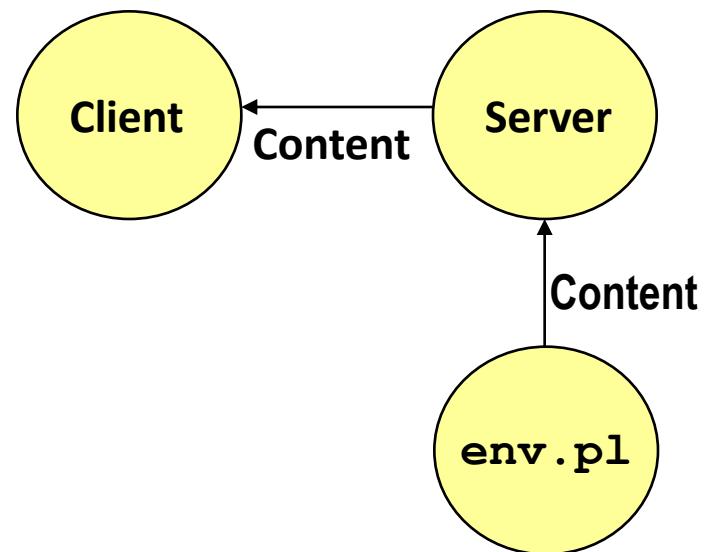
# Serving Dynamic Content (cont)

- The server creates a child process and runs the program identified by the URI in that process



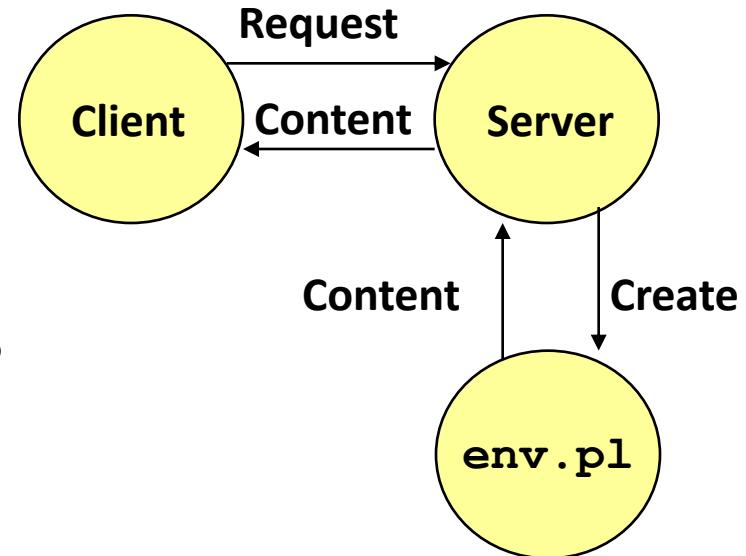
# Serving Dynamic Content (cont)

- The child runs and generates the dynamic content
- The server captures the content of the child and forwards it without modification to the client



# Issues in Serving Dynamic Content

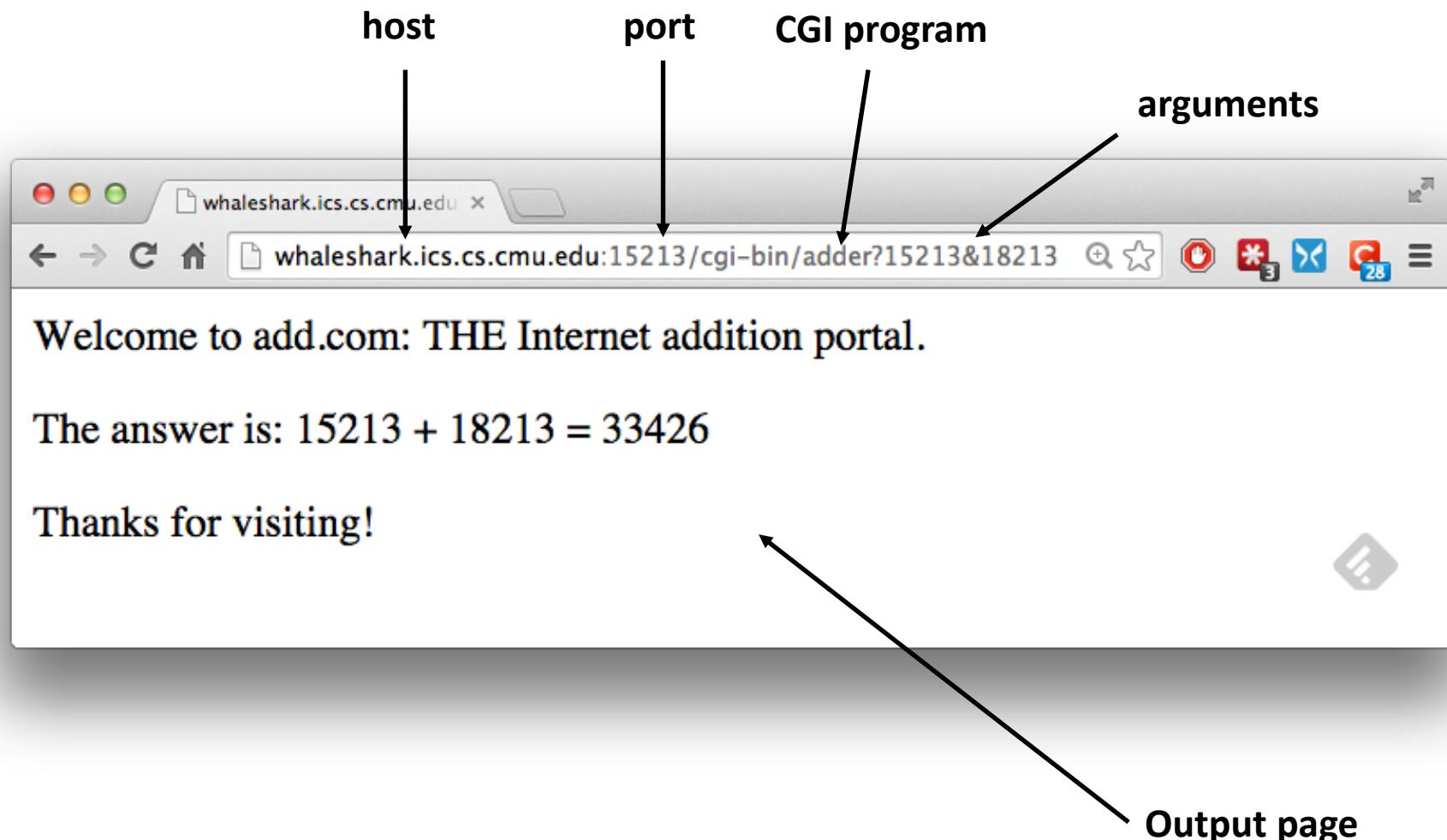
- How does the client pass program arguments to the server?
- How does the server pass these arguments to the child?
- How does the server pass other info relevant to the request to the child?
- How does the server capture the content produced by the child?
- These issues are addressed by the **Common Gateway Interface (CGI)** specification.



# CGI

- Because the children are written according to the CGI spec, they are often called *CGI programs*.
- However, CGI really defines a simple standard for transferring information between the client (browser), the server, and the child process.
- CGI is the original standard for generating dynamic content. Has been largely replaced by other, faster techniques:
  - E.g., fastCGI, Apache modules, Java servlets, Rails controllers
  - Avoid having to create process on the fly (expensive and slow).

# The add.com Experience



# Serving Dynamic Content With GET

- **Question:** How does the client pass arguments to the server?
- **Answer:** The arguments are appended to the URI
  
- **Can be encoded directly in a URL typed to a browser or a URL in an HTML link**
  - `http://add.com/cgi-bin/adder?15213&18213`
  - `adder` is the CGI program on the server that will do the addition.
  - argument list starts with “?”
  - arguments separated by “&”
  - spaces represented by “+” or “%20”

# Serving Dynamic Content With GET

- URL suffix:

- cgi-bin/adder?15213&18213

- Result displayed on browser:

Welcome to add.com: THE Internet  
addition portal.

The answer is:  $15213 + 18213 = 33426$

Thanks for visiting!

# Serving Dynamic Content With GET

- **Question:** How does the server pass these arguments to the child?
- **Answer:** In environment variable **QUERY\_STRING**
  - A single string containing everything after the “?”
  - For add: **QUERY\_STRING = “15213&18213”**

```
/* Extract the two arguments */
if ((buf = getenv("QUERY_STRING")) != NULL) {
    p = strchr(buf, '&');
    *p = '\0';
    strcpy(arg1, buf);
    strcpy(arg2, p+1);
    n1 = atoi(arg1);
    n2 = atoi(arg2);
}
```

adder.c

# Serving Dynamic Content with GET

- Question: How does the server capture the content produced by the child?
- Answer: The child generates its output on `stdout`. Server uses `dup2` to redirect `stdout` to its connected socket.

```
void serve_dynamic(int fd, char *filename, char *cgiargs)
{
    char buf[MAXLINE], *emptylist[] = { NULL };

    /* Return first part of HTTP response */
    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Server: Tiny Web Server\r\n");
    Rio_writen(fd, buf, strlen(buf));

    if (Fork() == 0) { /* Child */
        /* Real server would set all CGI vars here */
        setenv("QUERY_STRING", cgiargs, 1);
        Dup2(fd, STDOUT_FILENO);          /* Redirect stdout to client */
        Execve(filename, emptylist, environ); /* Run CGI program */
    }
    Wait(NULL); /* Parent waits for and reaps child */
}
```

# Serving Dynamic Content with GET

- Notice that only the CGI child process knows the content type and length, so it must generate those headers.

```
/* Make the response body */
sprintf(content, "Welcome to add.com: ");
sprintf(content, "%sTHE Internet addition portal.\r\n<p>", content);
sprintf(content, "%sThe answer is: %d + %d = %d\r\n<p>",
        content, n1, n2, n1 + n2);
sprintf(content, "%sThanks for visiting!\r\n", content);

/* Generate the HTTP response */
printf("Content-length: %d\r\n", (int)strlen(content));
printf("Content-type: text/html\r\n\r\n");
printf("%s", content);
fflush(stdout);

exit(0);
```

adder.c

# Serving Dynamic Content With GET

```
bash:makoshark> telnet whaleshark.ics.cs.cmu.edu 15213
Trying 128.2.210.175...
Connected to whaleshark.ics.cs.cmu.edu (128.2.210.175).
Escape character is '^]'.
GET /cgi-bin/adder?15213&18213 HTTP/1.0
```

*HTTP request sent by client*

```
HTTP/1.0 200 OK
Server: Tiny Web Server
Connection: close
Content-length: 117
Content-type: text/html
```

*HTTP response generated  
by the server*

```
Welcome to add.com: THE Internet addition portal.
<p>The answer is: 15213 + 18213 = 33426
<p>Thanks for visiting!
```

*HTTP response generated  
by the CGI program*

```
Connection closed by foreign host.
bash:makoshark>
```

# For More Information

- **W. Richard Stevens et. al. “Unix Network Programming: The Sockets Networking API”, Volume 1, Third Edition, Prentice Hall, 2003**
  - THE network programming bible.
- **Michael Kerrisk, “The Linux Programming Interface”, No Starch Press, 2010**
  - THE Linux programming bible.
- **Complete versions of all code in this lecture is available from the 213 schedule page.**
  - <http://www.cs.cmu.edu/~213/schedule.html>
  - csapp.{c,h}, hostinfo.c, echoclient.c, echoserveri.c, tiny.c, adder.c
  - You can use any of this code in your assignments.