Lecture 7

Networking, HTTP, CGI

Network Application

- Client application and server application communicate via a network protocol
- A protocol is a set of rules on how the client and server communicate

Internet Protocol Suite

- TCP and IP were developed as a standard networking protocol to connect a diverse set of networks
- Two layers:
  - IP – determines routing of packets of data from sender to receiver. Uses 32-bit addresses (e.g. 128.122.20.15)
  - TCP – connection-oriented protocol for reliable delivery of data. Acknowledgements, sequencing, retransmission, timeouts

Network Packet

- With TCP/IP, each machine has a number of ports that can be contacted from a client.
- A machine has to serve a port by listening for connections to it.
- Ports for popular services are fixed:
  - 1-1023 are reserved (well-known)
  - 1024-49151 are user level
  - 49152-65535 are private to the machine
- Clients use ephemeral ports
Naming

• In addition to addresses, nodes on the network can have associated names
• Names are translated into addresses by a server called a nameserver
• Local name address mappings stored in /etc/hosts

Sockets

• Sockets provide access to TCP/IP on UNIX systems
• Invented in Berkeley UNIX
• Allows a network connection to be opened as a file (returns a file descriptor)

Major Network Services

• Telnet
  – Provides a virtual terminal for a remote user
  – Port 23
  – telnet program can be used to connect to other ports
• FTP: File Transfer Protocol
  – A service that allows files to be transferred from one machine to another.
  – Uses port 20 for data, 21 for control
• SSH
  – Like telnet but encrypts data. Port 22

Major Network Services (cont.)

• SMTP
  – Host-to-host mail transport
  – Port 25
• IMAP
  – Email access
  – Port 143 (993 for SSL)
• HTTP
  – “… protocol for distributed, collaborative, hypermedia information systems”
  – Port 80

Ksh93: /dev/tcp

• Files in the form /dev/tcp/hostname/port result in a socket connection to the given service:

```bash
exec 3</dev/tcp/smtp.cs.nyu.edu/25 #SMTP
print -u3 "HELO cs.nyu.edu"
print -u3 "QUIT"
while IFS= read -u3 do
  print -r "$REPLY"
done
```

HTTP

• The Hyper Text Transfer Protocol: Port 80
• Language used to communicate between browsers (IE, Mozilla) and web servers (Apache, IIS)
• Browsers make requests:
  – Request a URL
  – Also includes info such as the browser type, formats accepted, etc.
• Web servers reply with two parts
  – Header information describing the data
  – The actual data (e.g. HTML document)
Sample HTTP session

GET /index.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Linux i686)
Host: www.cs.nyu.edu
Accept: image/gif, image/x-bitmap, image/jpeg, */*

HTTP/1.0 200 Document follows
Date: Tue, 05 Nov 2002 12:03:23 EST
Server: Apache 1.1
Last-modified: Mon, 04 Nov 2002 03:34:43 EST
Content-type: text/html
Content-length: 2493

<H1> This is a test </H1>

Sample HTTP session

GET /index.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Linux i686)
Host: www.cs.nyu.edu
Accept: image/gif, image/x-bitmap, image/jpeg, */*

HTTP/1.0 200 Document follows
Date: Tue, 05 Nov 2002 12:03:23 EST
Server: Apache 1.1
Last-modified: Mon, 04 Nov 2002 03:34:43 EST
Content-type: text/html
Content-length: 2493

<H1> This is a test </H1>

URLs

- Uniform Resource Locator


protocol host port resource

Connect to port 80 on machine www.cs.nyu.edu

GET /courses/fall02/G22.2245-001/index.htm

HTML

- Hyper-Text Markup Language
- A text document with formatting information
  - Tags are embedded in the text
  - Common tags: <P>, <B>...<B>, <UL>, <PRE>, <H1>
- Browsers turn HTML into visual presentation

HTML

- HTML is a file format that describes a web page.
- These files can be made by hand, or generated by a program
- A good way to generate an HTML file is by writing a shell script

CGI Overview

- Web servers allow HTML documents to be generated on the fly through the CGI standard.
- A request is made for a web page, your program is called by the web server to generate the HTML, the HTML is rendered in the browser
  - Your program outputs HTML to standard output
- There are ways to get input to your script
  - Through standard input and/or environment variables
### HTML Forms

- An HTML form provides a way to collect user input
  - Text Areas
  - Buttons
  - Menus
  - Checkboxes
- Browser sends data via HTTP request
- Invokes a URL of a CGI script to process data when submitted

### Forms and CGI

- HTTP defines how form variables are sent to the web server
- Two methods:
  - GET
    - Form variables encoded into an environment variable
  - POST
    - Form variables encoded into standard input as the content of the HTTP request

### Sending form variables

- Browser sends form variables as name-value pairs:
  ```
  name1=value1&name2=value2&name3=value3
  ```
- Names are defined in form elements
- Values are specified by user
  - Encoded into special format: special characters replaced with %## (2-digit hex number), spaces replaced with +
    - Avoids parsing problems
    - E.g. "10/20 Wed" encoded as "10%2F20+Wed"

### Submitting forms

- POST
  ```
  POST /cgi-bin/sample.cgi HTTP/1.1
  Host: www.cs.nyu.edu
  Content-Length: 50
  Content-Type: application/x-www-form-urlencoded
  name=value1&name2=value2&name3=value3
  ```
- GET
  ```
  GET /cgi-bin/sample.cgi HTTP/1.1
  Host: www.cs.nyu.edu
  ```

### Reading form inputs

- Forms specify whether to use GET or POST style HTTP request
  ```
  <FORM method="POST" action="/cgi-bin/sample.cgi">
  ...<br />
  </FORM>
  ```
- GET: input encoded into QUERY_STRING
- POST: standard input (body of the request)
- Most scripts parse the input into an associative array
  - You can parse these yourself
  - But most people use libraries for this
**CGI Environment Variables**

- DOCUMENT_ROOT
- HTTP_HOST
- HTTP_REFERER
- HTTP_USER_AGENT
- HTTP_COOKIE
- REMOTE_ADDR
- REMOTE_HOST
- REMOTE_USER
- REQUEST_METHOD
- SERVER_NAME
- SERVER_PORT

**CGI Script: Example**

```bash
#!/home/unixtool/bin/ksh

# Read special functions to help parse
ReadParse
PrintHeader
print "r -- "${Cgi.comment}"
print "<H2>You submitted the comment</H2>"
print "<pre>"
print "r -- "${Cgi.comment}"
print "</pre>"
```

**Part 1: HTML Form**

```html
<html>
<center>
<H1>Anonymous Comment Submission</H1>
</center>
Please enter your comment below which will be sent anonymously to <tt>kornj@cs.nyu.edu</tt>. If you want to be extra cautious, access this page through <a href="http://www.anonymizer.com">Anonymizer</a>.
<form action="cgi-bin/comment.cgi" method="post">
<textarea name="comment" rows="20" cols="80">
</textarea>
<input type="submit" value="Submit Comment">
</form>
</html>
```

**Part 2: CGI Script (ksh)**

```bash
#!/home/unixtool/bin/ksh

cgi-lib.ksh # Read special functions to help parse
ReadParse
PrintHeader
print -r -- "${Cgi.comment}" | /bin/mailx -s "COMMENT" kornj
print "<H2>You submitted the comment</H2>"
print "<pre>"
print -r -- "${Cgi.comment}"
print "</pre>"
```

**Debugging**

- Debugging can be tricky, since error messages don't always print well as HTML
- One method: run interactively

```bash
$ QUERY_STRING='birthday=10/15/03'
$ ./birthday.cgi
Content-type: text/html

<html>
Your birthday is <tt>10/15/02</tt>.
</html>
```

**How to get your script run**

- This can vary by web server type
  [http://www.cims.nyu.edu/systems/resources/webhosting/index.html](http://www.cims.nyu.edu/systems/resources/webhosting/index.html)
- Typically, you give your script a name that ends with .cgi
- Give the script execute permission
- Specify the location of that script in the URL
CGI Security Risks

- Often CGI scripts are run as the author – setuid
- Be careful of security holes
- Never trust the input
- Clean up (don't leave sensitive data around)

CGI Benefits

- Simple
- Language independent
- UNIX tools are good for this because
  - Work well with text
  - Integrate programs well
  - Easy to prototype
  - No compilation (CGI scripts)

Example: Dump Some Info

```bash
#!/home/unixtool/bin/ksh

# /cgi-lib.ksh
PrintHeader
ReadParse
print "<h1>Date</h1>
print "<pre>
date
</pre>
print "<h1>Form Variables</h1>
print "<pre>
set -s -- ${!Cgi.*}
for var
do
    nameref ${var} = $var
    unset ${var}
done
print "</pre>
print "<h1>Environment</h1>
print "<pre>
env | sort
</pre>
```

Example: Find words in Dictionary

```bash
#!/home/unixtool/bin/ksh

PATH="/usr/bin:

# /cgi-lib.ksh
ReadParse
PrintHeader
print "<h1>Words matching <tt>${Cgi.re}</tt> in the dictionary</h1>
print "<ol>
grep "$Cgi.re" /usr/dict/words | while read word
do
    print "<li>$word"
done
print "</ol>"
```

Example: Find words in Dictionary

```bash
<form action=dict.cgi
 Regular expression: <input type=entry
 name=re value=".*">
<input type=submit>
</form>
```