Web Search

Interfaces

Web Search Interface

- Web search engines of course need a web-based interface.
- Search page must accept a query string and submit it within an HTML <form>.
- Program on the server must process requests and generate HTML text for the top ranked documents with pointers to the original and/or cached web pages.
- Server program must also allow for requests for more relevant documents for a previous query.

Submit Forms

- HTML supports various types of program input in forms, including:
  - Text boxes
  - Menus
  - Check boxes
  - Radio buttons
- When user submits a form, string values for various *parameters* are sent to the server program for processing.
- Server program uses these values to compute an appropriate HTML response page.

Simple Search Submit Form

```
<form method= "POST" action="/form">
<input type="text" name="FirstInput" size = "20">
<font color="red">
Type input into the box</font><br>
<br>
<input type="text" name="SecondInput" size = "20">
<font color="green">
Type input into the box</font><br>
<br>
<font color = "yellow">
<input type="submit" name="Submit" value = "Submit">
</font><br>
<br>
</form>
```

How To Handle Form Submissions?

- There are many ways of handling form submissions.
- Servlet (written in Java and other languages) that provides action on the server side, the opposite of Applet
- Apache Tomcat is an example of Java implementation jakarta.apache.org/tomcat/
- CGI: Common Gateway Interface
- We will write our own server that supports search

Basic Web Server Structure

- Server program creates a socket for connection.
- Server program waits for *clients* request for connection. Clients here typically are Web browser such as Netscape.
- Once the server receives a request, it examines the type of request and perform the service as requested.
- The server then sends the results back to the client, typically in an HTML format.

Code Example of a Simple Web Server

- See transparency for the code example
- Also at http://www.eg.bucknell.edu/~csci335/2006-fall/code/javaServer/EasyWebServer.java

Socket API in Java

- A socket is a communication point. Java has two types of socket, a ServerSocket that waits for clients to connect at a given port
  ServerSocket server = new ServerSocket(PORT);
- When a client (a browser) connects to a server, the server creates a socket to work with that client (Socket sock = server.accept();)
- When the work is finished, the server closes the socket
- A server may work with many clients any any moment

Server-Client Communication

- When a browser connects to a server it sends a collection of information to the server. Here is an example

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [en] (X11; U; SunOS 5.8 sun4u)
Host: polaris:9999
Accept: image/gif, image/x-xbitmap, image/jpeg,
    image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

Server-Client Communication -- cont

- The first line is most important. It indicates the client requests a "GET" operation at the given path "/"
- When the server receives this request, it first checks to see if the request is a valid one. If it is, the server performs the service and returns the results to the client.
- If the request is a regular Web page, as the above example, the requested page is sent.

Server-Client Communication -- cont

- Code example (the method processHTTPCmd) is on the transparency and at http://www.eg.bucknell.edu/~csci335/2006-fall/code/javaServer/EasyWebServer.java
- If the client is sending a form (typically a search request), the server has to process the form and extract the information from the the form.
- When the client sends a form, it is requesting to POST the form to the server

Server-Client Communication -- cont

- The header sent to the server looks as follows.

```
POST /form HTTP/1.0
Referer: http://polaris:9999/search
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [en] (X11; U; SunOS 5.8 sun4u)
Host: polaris:9999
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
    image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 44
```

Server-Client Communication -- cont

- Key differences from previous "GET" example:
  - The command is now "POST"
  - It has a "Content-type" and a "Content-length" component
- The server responds according to the header
- The request has a "POST" so the server knows an action is needed
- The request has a "Content-type" of form

Server-Client Communication -- cont

- The request has a "Content-length" so the server knows how long is the form. In our example, the length is 44
- The server will read the form following the header from the client.
- The forms are sent in from the client in pairs of name=value separated by &. In our example, it looks as follows, 44 chars long.
  FirstInput=123&SecondInput=abc&Submit=Sub
  mit

Server-Client Communication -- cont

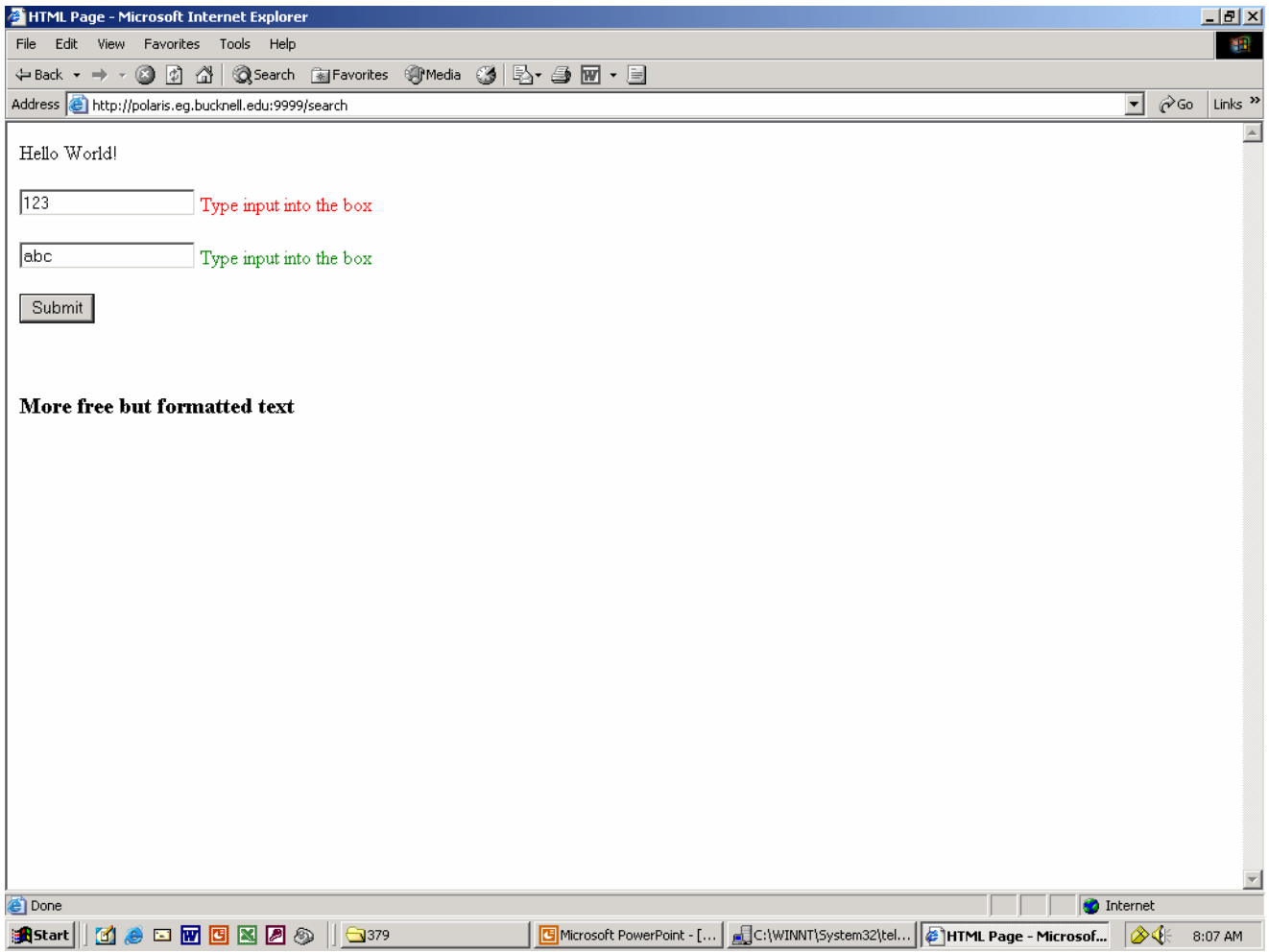
- How was this string formed? Check the HTML code for the form.

```
<input type="text" name="FirstInput">
Type input into the box</font><br>
<input type="text" name="SecondInput">
Type input into the box</font><br>
<input type="submit" name="Submit" value
  = "Submit">
```

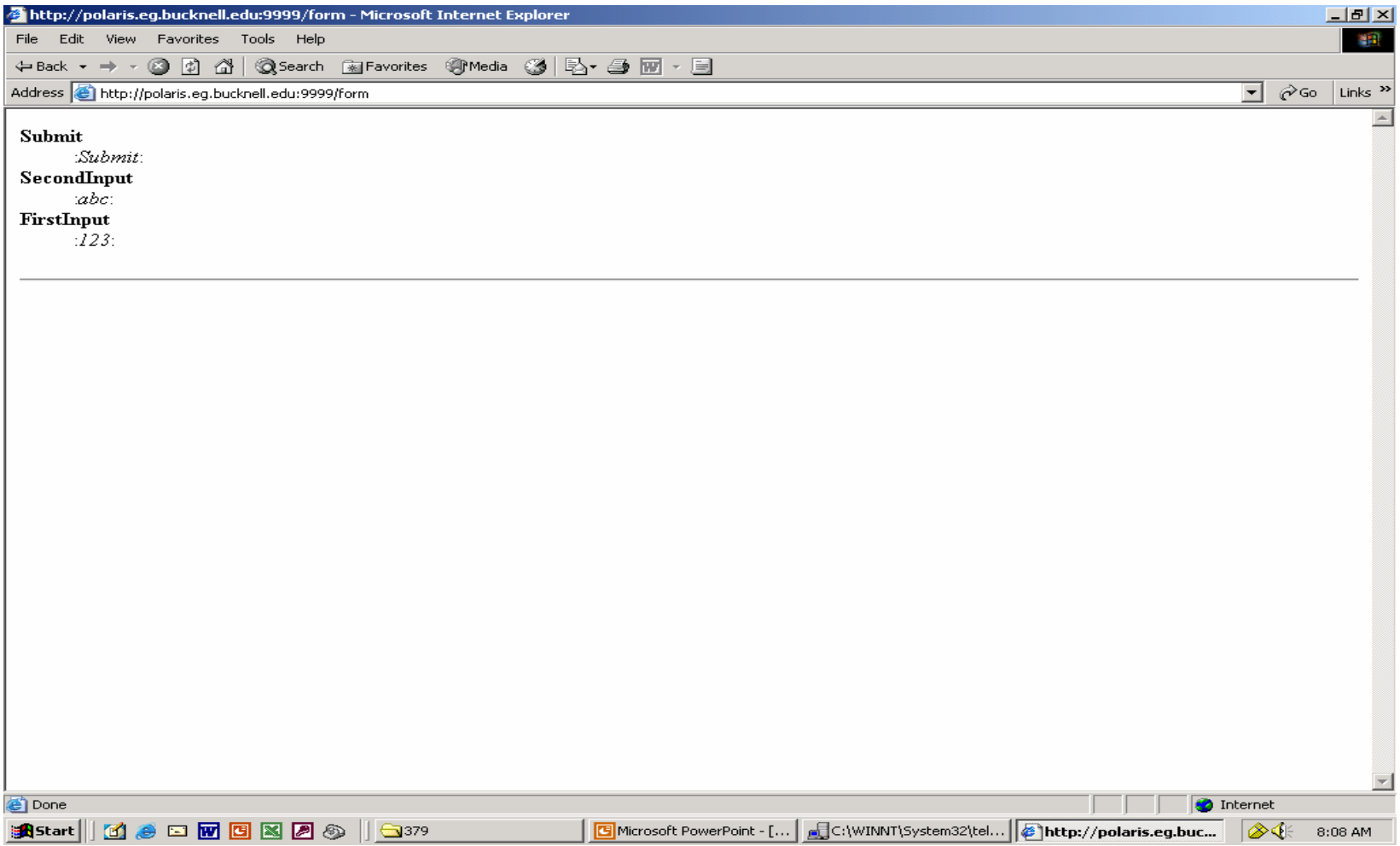
Server-Client Communication -- cont

- The server then parses out the form and act accordingly.
- In our sample program, we simply echo back the values filled in the form. In actual search engine, the parsed words will be used to retrieve the relevant documents.
- To parse the form input, we used the Java method StringTokenizer

Snapshots of the Sample Web Server



Snapshots of the Sample Web Server



Simple Search Interface Refinements

- Currently reprocesses query for "More results" requests.
  - Could store current ranked list with the user session.
- Could integrate relevance feedback interaction.
- Could provide "Get similar pages" request for each retrieved document (as in Google).
  - Just use given document text as a query.

Other Search Interface Refinements

- Highlight search terms in the displayed document.
  - Provided in cached file on Google.
- Allow for “advanced” search:
  - Phrasal search (“..”)
  - Mandatory terms (+)
  - Negated term (-)
  - Language preference
  - Reverse link
  - Date preference
- Machine translation of pages.

Clustering Results

- Group search results into coherent “clusters”:
  - “microwave dish”
    - One group of on food recipes or cookware.
    - Another group on satellite TV reception.
  - “Austin bats”
    - One group on the local flying mammals.
    - One group on the local hockey team.
- Vivisimo groups results into “folders” based on a pre-established categorization of pages (like Yahoo or DMOZ categories).
- Alternative is to dynamically cluster search results into groups of similar documents.

User Behavior

- Users tend to enter short queries.
  - Study in 1998 gave average length of 2.35 words.
  - A 2003 study result is similar
- Users tend not to use advance search options.
- Users need to be instructed on using more sophisticated queries.