10.1 The Common Gateway Interface

- Markup languages cannot be used to specify computations, interactions with users, or to provide access to databases

- There are three approaches to providing these computational needs for documents

1. CGI is one common way to provide for these needs, by allowing browsers to request the execution of server-resident software

CGI is part of the appraoch often called LAMP (Linux, Apache, MySQL, Px), where Px is Perl, PHP, or Python

- Open-source software

2. Active Server Pages - ASP or ASP.NET
   - Microsoft

3. Java Server Pages – Sun Microsystems

- Similar to ASP and ASP.NET

10.1 The Common Gateway Interface (continued)

- CGI is just an interface between browsers and servers

- An HTTP request to run a CGI program specifies a program, rather than a document

  - Servers can recognize such requests in two ways:

    1. By the location of the requested file (special subdirectories for such files)

    2. A server can be configured to recognize executable files by their file name extensions

- A CGI program can produce a complete HTTP response, or just the URL of an existing document

- CGI programs often are stored in a directory named `cgi-bin`

10.2 CGI Linkage

- Some CGI programs are in machine code, but Perl programs are usually kept in source form, so `perl` must be run on them

- A source file can be made to be “executable” by adding a line at their beginning that specifies that a language processing program be run on them first

  For Perl programs, if the `perl` system is stored in `/usr/local/bin/perl`, as is often is in UNIX systems, this is

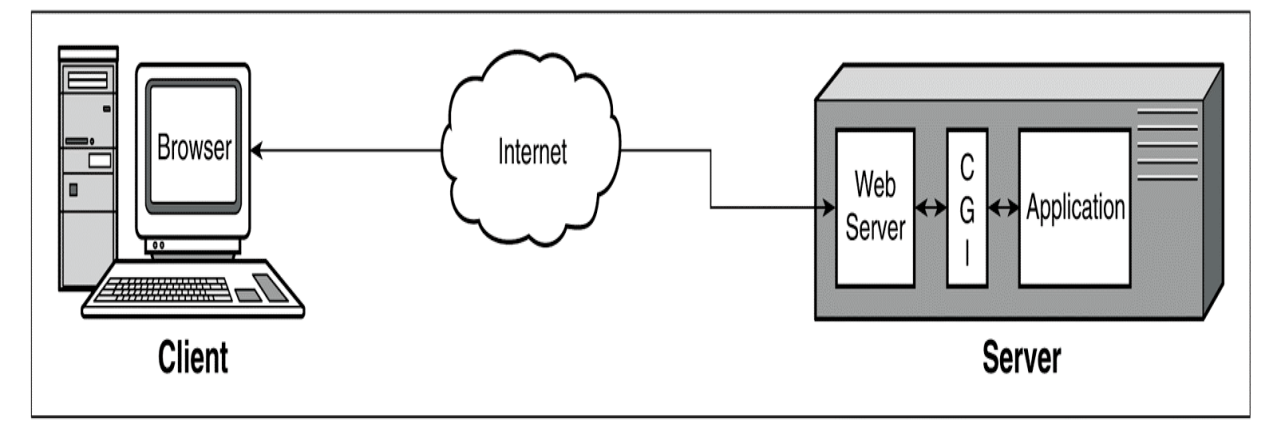
```
#!/usr/local/bin/perl -w
```

- The file extension `.cgi` is sometimes used for Perl CGI programs (e.g., for the CGI Wrap system)

- An HTML document specifies a CGI program with the hypertext reference attribute, `href`, of an anchor tag, `<a>`, as in

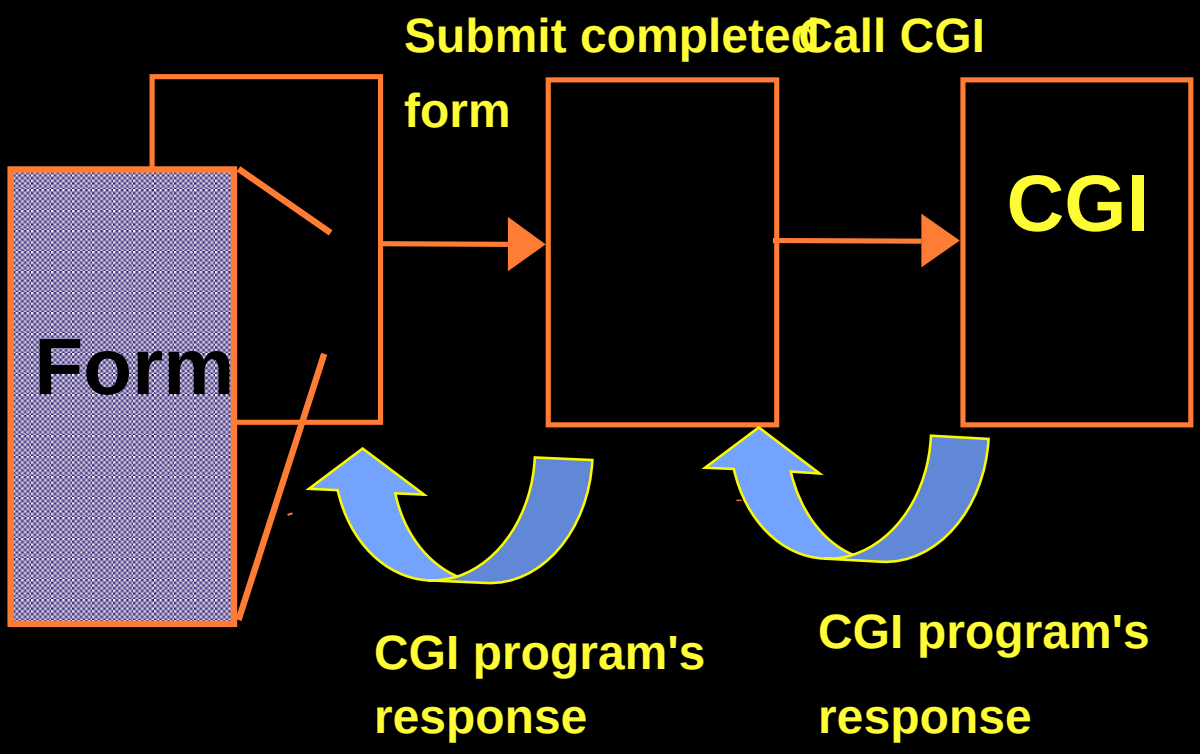
```
<a href =
   “./cgi-bin/reply.cgi>"
Click here to run the CGI program, reply.pl
</a>
```

FIGURE 10.1
Communications and computation using CGI

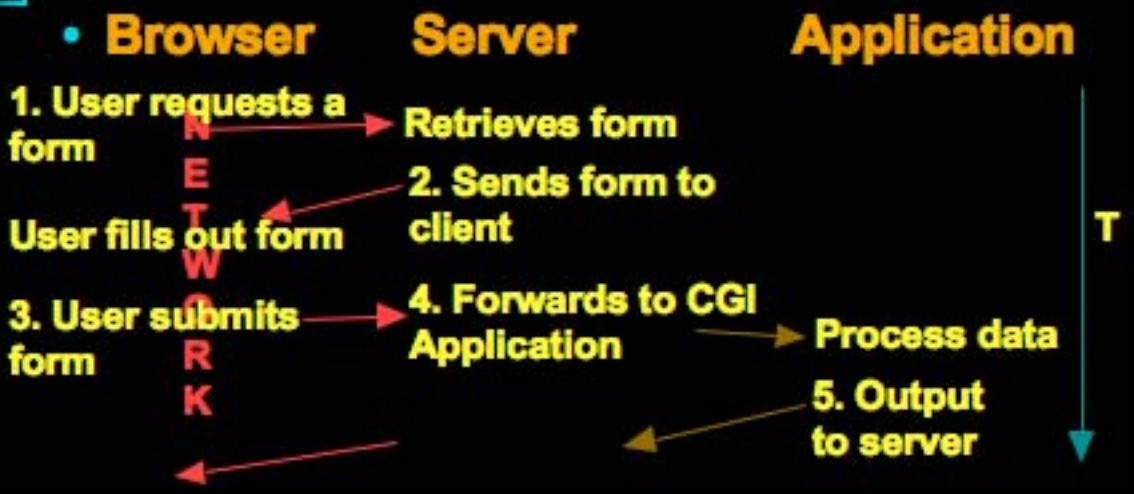


Model ...

- Browser application server



Form interaction with CGI



10.2 CGI Linkage (continued)

```
<!-- reply.html - calls a trivial cgi program
     -->
 <html>
<head>
<title>
   HTML to call the CGI-Perl program reply.cgi
</title>
</head>
<body>
This is our first CGI-Perl example
<a href = "./cgi-bin/reply.cgi">
Click here to run the CGI program, reply.cgi
</a>
</body>
</html>
```

- The connection from a CGI program back to the requesting browser is through standard output, usually through the server

- The HTTP header needs only the content type, followed by a blank line, as is created with:

```
print "Content-type: text/html \n\n";
```

→ SHOW `reply.cgi`

10.3 Query String Format

- A query string includes names and values of widgets

- Widget values are always coded as strings

- The form of a name/value pair in a query string is: name=value

- If the form has more than one widget, their values are separated with ampersands

```
milk=2&payment=visa
```

- Each special character is coded as a percent sign and a two-character hexadecimal number (the ASCII code for the character)

- Some browsers code spaces a plus signs, rather than as `%20`

- `get` versus `post` HTTP methods

GET vs. POST ...

- GET
  - + Access CGI prog. w/ query without a form
    - Pass parameters to program ...
    - Can send extra path info ...
  - – Query might get truncated
- Post
  - + Unlimited query length
  - – No “canned” queries (hard-coded url)

10.4 The `CGI.pm` Module

- A Perl module serves as a library

  - Can be used as a function library or a class library

  - Get info about by typing `perldoc CGI`

- The Perl `use` declaration is used to make a module available to a program

  - To make only part of a module available, specify the part name after a colon

    (For our purposes, only the `standard` part of the CGI module is needed)

```
use CGI ":standard";
```

- Common `CGI.pm` Functions

  - "Shortcut" functions produce tags, using their parameters as attribute values

```
br;
```

returns

```
"<br/>"
```

10.4 The CGI.pm Module (continued)

- e.g., `h2("Very easy!");` returns
  `<h2> Very easy! </h2>`

  - In this example, the parameter to the function `h2` is used as the content of the `<h2>` tag

- To get the output of a function to the return document, the call must be a parameter to `print`

```
print h2("Very easy!");
```

- Tags can have both content and attributes

  - Each attribute is passed as a name/value pair, just as in a hash literal

    - Attribute names are passed with a preceding dash

```
textarea(-name => "Description",
         -rows => "2",
        -cols => "35"
 );
```

Produces:

```
<textarea name ="Description" rows=2
         cols=35> </textarea>
```

10.4 The CGI.pm Module (continued)

- If both content and attributes are passed to a function, the attributes are specified in a hash literal as the first parameter

```
a({-href => "fruit.html"},
    "Press here for fruit descriptions");
```

Output:
```
<a href="fruit.html">
Press here for fruit descriptions</a>
```

- Tags and their attributes are distributed over the parameters of the function

```
ol(li({-type => "square"},
        ["milk", "bread", "cheese"]));
```

Output:
```
<ol>
  <li type="square"milk</li>
  <li type="square"bread</li>
  <li type="square"cheese</li>
</ol>
```

- CGI.pm also includes non-shortcut functions, which produce output for return to the user

- A call to header() produces:

```
Content-type: text/html;charset=ISO-8859-1
  -- blank line --
```

10.4 The CGI.pm Module (continued)

- The `start_html` function is used to create the head of the return document, as well as the `<body>` tag

  - The parameter to `start_html` is used as the title of the document

```
start_html("Bill's Bags");

DOCTYPE html PUBLIC
 "-//W3C//DTD XHTML Basic 1.0 //EN"
 "http://www.w3.org/TR/xhtml-basic/
     xhtml-basic10.dtd">
<html xmlns=
  "http://www.w3.org/1999/xhtml lang="en-US">
<head><title>Bill's Bags</title>
</head><body>
```

- The `param` function is given a widget's name; it returns the widget's value

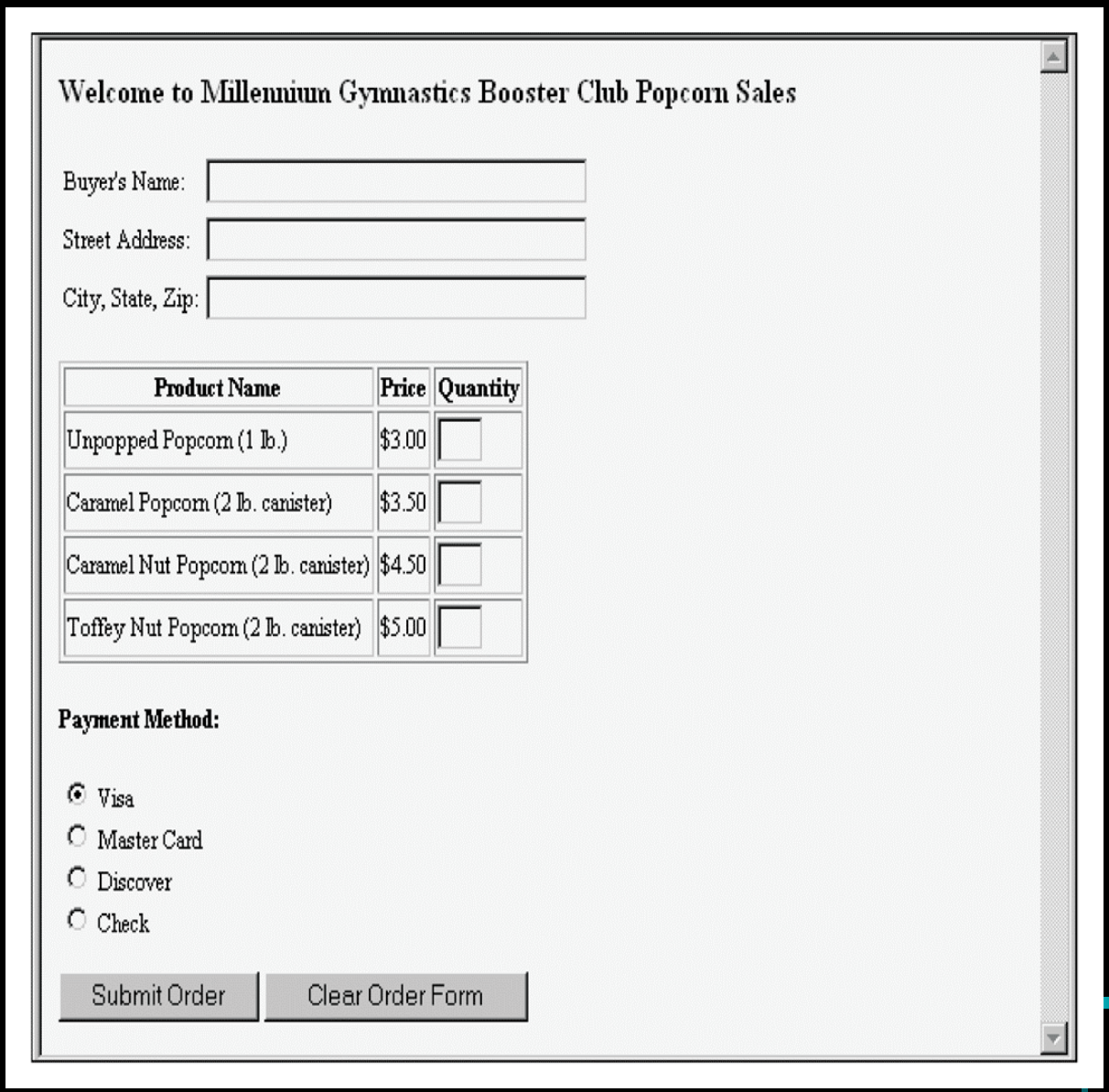
  - If the query string has `name=Abraham` in it,

    `param("name")` will return `"Abraham"`

- The `end_html` function generates `</body></html>`

→SHOW `popcorn.html` , its display, and `popcorn.cgi`

FIGURE 10.2 Display of popcorn.html



Welcome to Millennium Gymnastics Booster Club Popcorn Sales

Buyer's Name:

Street Address:

City, State, Zip:

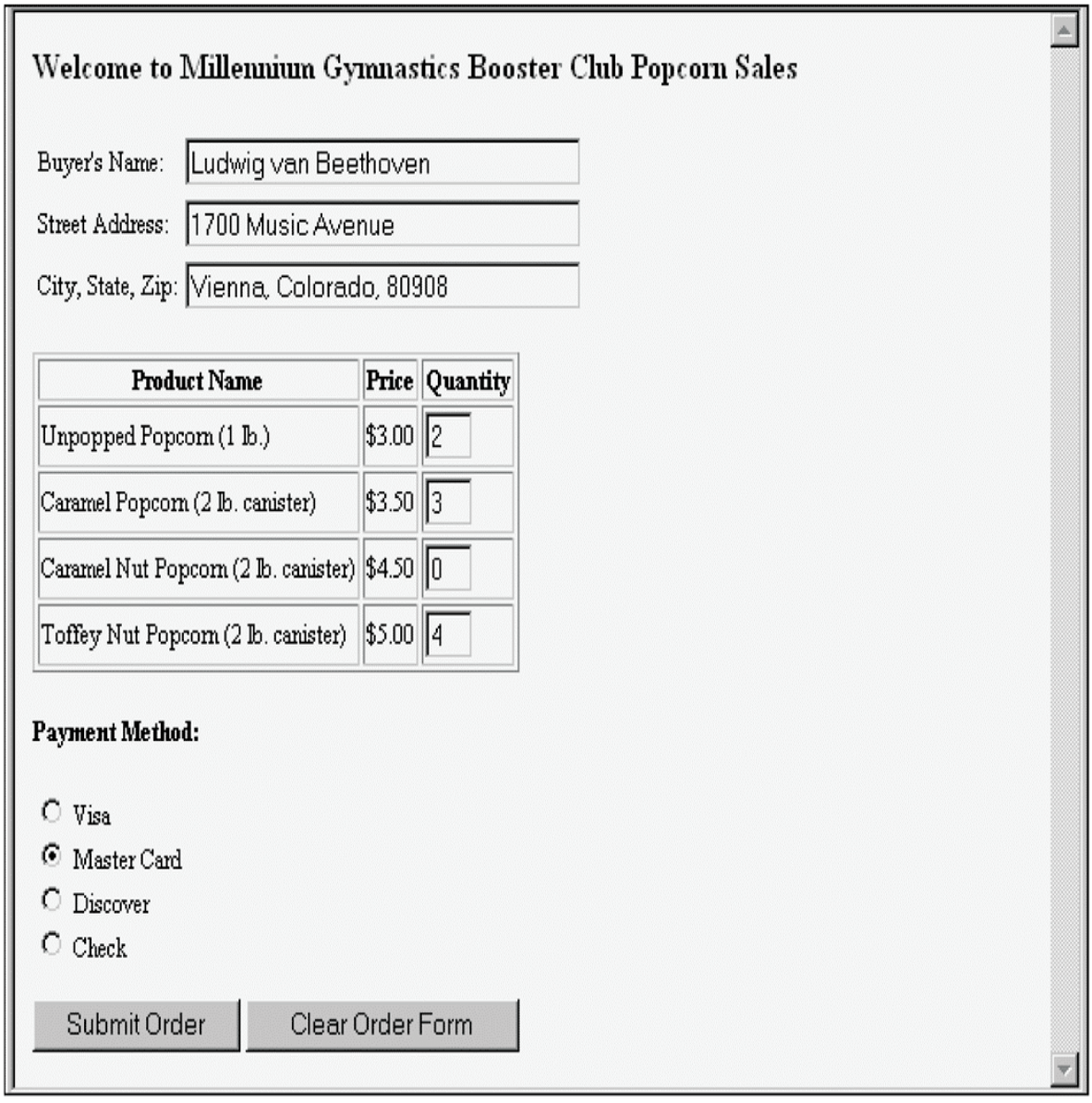
Product Name	Price	Quantity
Unpopped Popcorn (1 lb.)	$3.00	
Caramel Popcorn (2 lb. canister)	$3.50	
Caramel Nut Popcorn (2 lb. canister)	$4.50	
Toffey Nut Popcorn (2 lb. canister)	$5.00	

Payment Method:

- (•) Visa
- ( ) Master Card
- ( ) Discover
- ( ) Check

Submit Order | Clear Order Form

FIGURE 10.3 Display of popcorn.html with the form filled out



Welcome to Millennium Gymnastics Booster Club Popcorn Sales

Buyer's Name: Ludwig van Beethoven

Street Address: 1700 Music Avenue

City, State, Zip: Vienna, Colorado, 80908

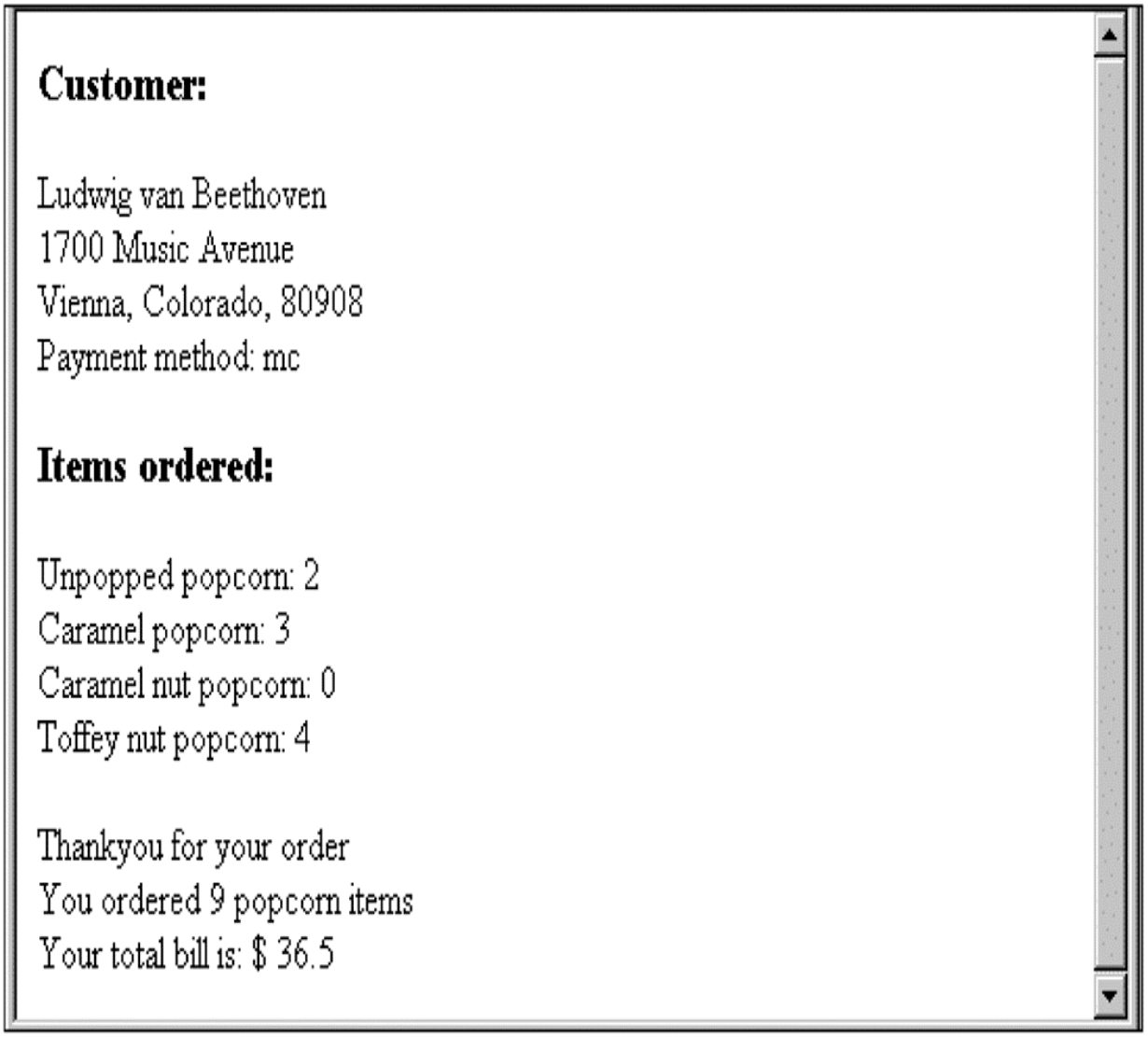
Product Name	Price	Quantity
Unpopped Popcorn (1 lb.)	$3.00	2
Caramel Popcorn (2 lb. canister)	$3.50	3
Caramel Nut Popcorn (2 lb. canister)	$4.50	0
Toffey Nut Popcorn (2 lb. canister)	$5.00	4

Payment Method:

- ○ Visa
- ◉ Master Card
- ○ Discover
- ○ Check

Submit Order | Clear Order Form

FIGURE 10.4
Output of popcorn.cgi



Customer:

Ludwig van Beethoven
1700 Music Avenue
Vienna, Colorado, 80908
Payment method: mc

Items ordered:

Unpopped popcorn: 2
Caramel popcorn: 3
Caramel nut popcorn: 0
Toffey nut popcorn: 4

Thankyou for your order
You ordered 9 popcorn items
Your total bill is: $ 36.5

10.5 A Survey Example

- Use a form to collect survey data from users

- The program must accumulate survey results, which must be stored between form submissions in a file on the server

  - Because of concurrent use of the file, it must be protected from corruption by blocking other accesses while it is being updated

    - Under UNIX, this can be done with the Perl function, `flock`

    - The parameter values for flock are defined in the Perl `Fcntl` module (write lock is `LOCK_EX`)

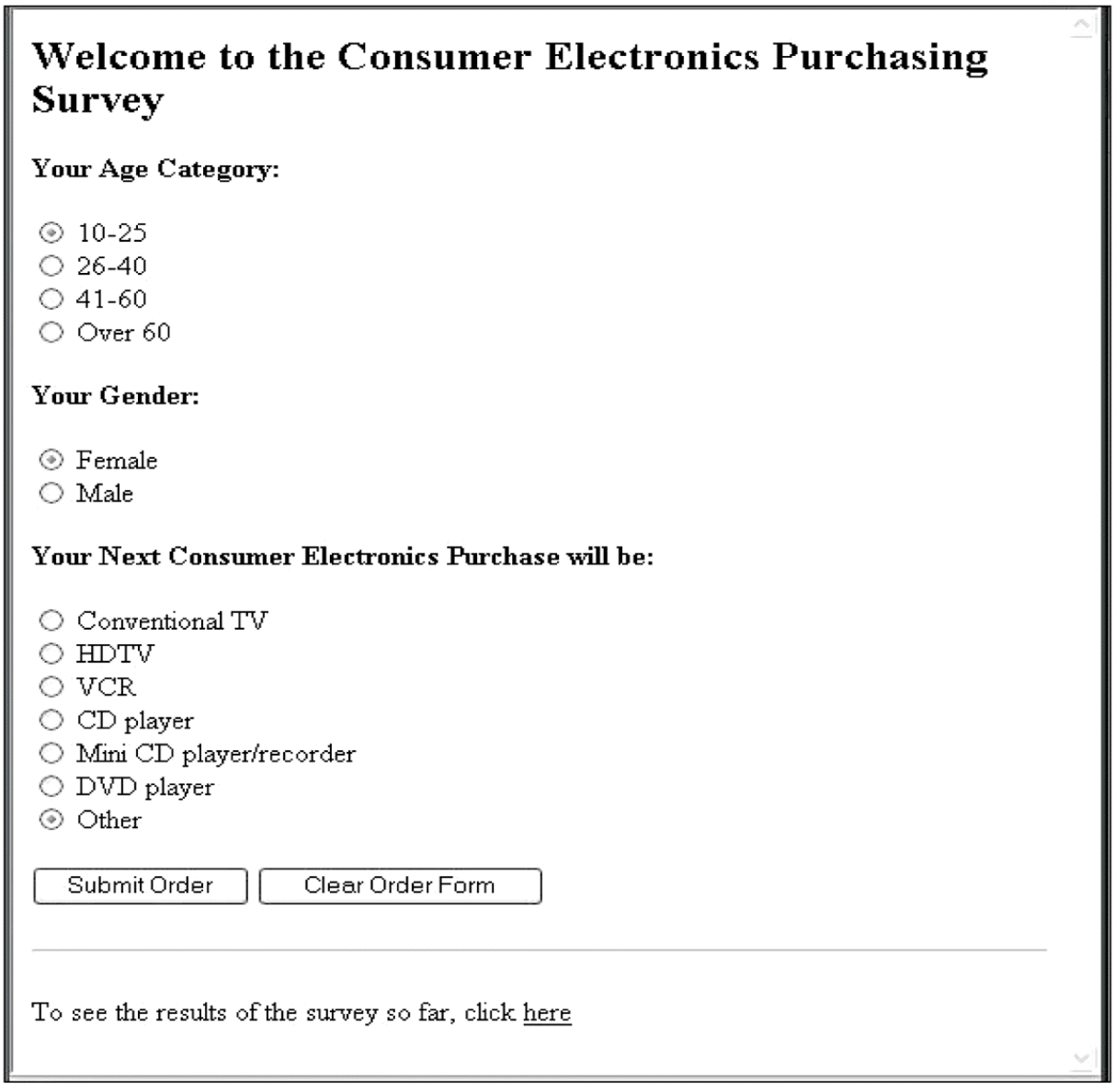
- To update a file, open the file, lock it with flock, modify it, rewind it with `seek`, and close the file

--> SHOW `conelec.html` and its display

- Two CGI programs are used for this application, one to collect survey submissions and record the new data, and one to produce the current totals

- The file format is eight lines, each having seven values, the first four for female responses and the last four for male responses

FIGURE 10.5 Display of conelec.html



Welcome to the Consumer Electronics Purchasing Survey

Your Age Category:

- (•) 10-25
- ( ) 26-40
- ( ) 41-60
- ( ) Over 60

Your Gender:

- (•) Female
- ( ) Male

Your Next Consumer Electronics Purchase will be:

- ( ) Conventional TV
- ( ) HDTV
- ( ) VCR
- ( ) CD player
- ( ) Mini CD player/recorder
- ( ) DVD player
- (•) Other

Submit Order | Clear Order Form

To see the results of the survey so far, click here

10.5 A Survey Example (continued)

- The program to collect and record form data must:

1. Get the form values (with `param`)
2. Determine which row of the file must be modified
3. Open, lock, and read the survey data file
4. Split the affected data string into numbers and store them in an array
5. Modify the affected array element and join the array back into a string (use gender to determine which half of the data is affected)
6. Rewind the data file (with `seek`)
7. Rewrite and close the survey data file

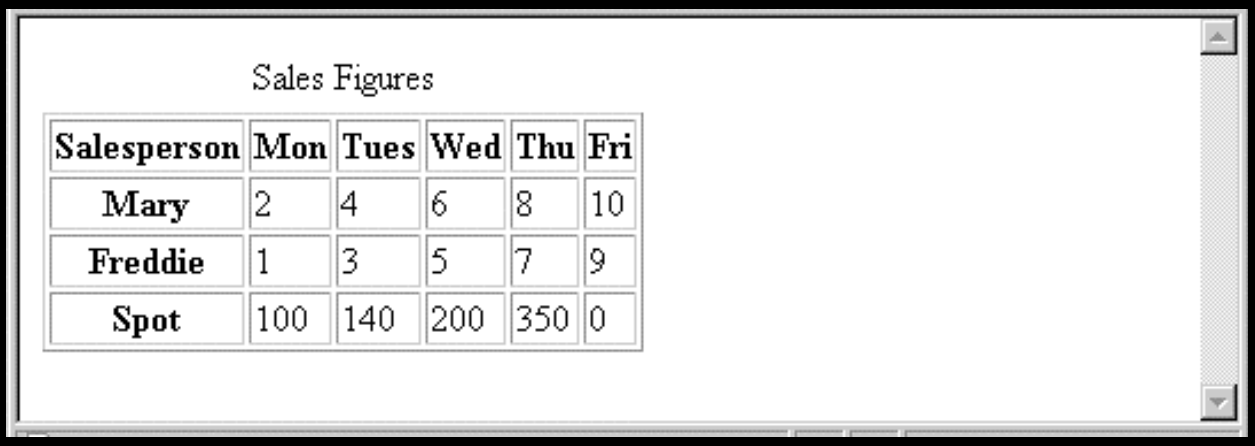
--> SHOW `conelec1.pl`

10.5 A Survey Example (continued)

- Tables are easier to specify with `CGI.pm`
  - The table is created with the `table` function
  - The `border` attribute is specified as a parameter
  - The table's caption is created with a call to `caption`, as the second parameter to table
  - Each row of the table is created with a call to `Tr`
  - A heading row is created with a call to `th`
  - Data cells are created with calls to `td`
  - The calls to `Tr`, `th`, and `td` require references as parameters
  - Suppose we have three arrays of sales numbers, one for each of three salespersons; each array has one value for each day of the work week
    - We want to build a table of this information, using `CGI.pm`

10.5 A Survey Example (continued)

```
table({-border => "border"},
       caption("Sales Figures"),
       Tr(
           [th(["Salesperson", "Mon", "Tues",
                "Wed", "Thu", "Fri"]),
            th("Mary").td(\@marysales),
            th("Freddie").td(\@freddiesales),
            th("Spot").td(\@spotsales),
                  ]
              )
       );
```



Sales Figures

Salesperson	Mon	Tues	Wed	Thu	Fri
Mary	2	4	6	8	10
Freddie	1	3	5	7	9
Spot	100	140	200	350	0

10.5 A Survey Example (continued)

- *The program that produces current results must*:

1. Open the file and read the lines into an array of strings

2. Split the first four rows (responses from females) into arrays of votes for the four age groups

3. Unshift row titles into the vote rows (making them the first elements)

4. Create the column titles row with `th` and put its address in an array

5. Use `td` on each rows of votes

6. Push the addresses of the rows of votes onto the row address array

7. Create the table using `Tr` on the array of row addresses

8. Repeat Steps 2-7 for the last four rows of data (responses from males)

10.5 A Survey Example (continued)

--> SHOW `conelec2.cgi`
--> SHOW Figure 10.7

FIGURE 10.7
Survey results page

Results of the Consumer Electronics Purchasing Survey

Survey Data for Females

Age Group	Conventional TV	HDTV	VCR	CD player	MiniCD player/recorder	DVD player	Other
10-25	3	0	0	2	0	0	1
26-40	0	2	0	0	0	0	0
41-60	0	0	2	0	0	0	0
Over 60	0	0	0	0	0	0	0

Survey Data for Males

Age Group	Conventional TV	HDTV	VCR	CD player	MiniCD player/recorder	DVD player	Other
10-25	1	3	0	0	0	0	0
26-40	0	0	0	0	1	0	0
41-60	0	0	0	0	2	3	0
Over 60	0	0	0	2	0	0	0

10.6 Cookies

- A *session* is the time span during which a browser interacts with a particular server

- The HTTP protocol is stateless

- But, there are several reasons why it is useful for the server to relate a request to a session

  - Shopping carts for many different simultaneous customers

  - Customer profiling for advertising

  - Customized interfaces for specific clients

- *Approaches to storing client information:*

  - Store it on the server – often too much to store!

    - Store it on the client machine - this works

10.6 Cookies (continued)

- A cookie is a small object of information consisting of a name and a textual value

- Cookies are created by some software system on the server (maybe a CGI program)

- Every HTTP communication between the browser and the server includes information in its header about the message

- At the time a cookie is created, it is given a lifetime

- Every time the browser sends a request to the server that created the cookie, while the cookie is still alive, the cookie is included

- A browser can be set to reject all cookies

- `CGI.pm` includes support for cookies

```
cookie(-name => a_cookie_name,
       -value => a_value,
       -expires => a_time_value);
```

  - The name can be any string
  - The value can be any scalar value
  - The time is a number followed by a unit code (`d`, `s`, `m`, `h`, `M`, `y`)

10.6 Cookies (continued)

- Cookies must be placed in the HTTP header at the time the header is created

```
header(-cookie => $my_cookie);
```

- To fetch the cookies from an HTTP request, call `cookie` with no parameters

  - A hash of all current cookies is returned

- To fetch the value of one particular cookie, send the cookie's name to the `cookie` function

```
$age = cookie('age');
```

- *Example:*

  A cookie that tells the client the time of his or her last visit to this site

  - Use the Perl function, `localtime`, to get the parts of time

```
($sec, $min, $hour, $mday, $mon, $year,
      $wday, $yday, $isdst) = localtime;
```

→SHOW `time_date.pl`

10.6 Cookies (continued)

- The CGI program to solve our problem:

1. Get the cookie named `last_time`
2. Get the current day of the week, month, and day of the month and put them in a cookie named `last_time`
3. Put the cookie in the header of the return document
4. If there was no existing cookie, produce a welcome message for the first-time visitor
5. If there was a cookie, produce a welcome message that includes the previous day of the week, month, and day of the month

→ SHOW `day_cookie.pl`