Lecture 11
Server Side programming Using Perl

Print page

In the previous lecture, we have learnt the basic concept of CGI and server side programming. In this lecture we'll reinforce CGI programming skills via a set of examples, where file operations are to be illustrated. Note that all these examples are very fragile. I did not include too much error catching in the code in order to make the code more readable. Any change on the code could cause problem to run.

All the code is tested in my working environment. To make the code working for you, the URL of your CGI program should be in the following format:

http://student.scem.westernsydney.edu.au/cgi-bin/cgiwrap/~XXXXXXX/YYYYYYY.cgi

where XXXXXXX is your school user id and YYYYYY is your CGI program name. To run your CGI, you must be connect to the school using VPN or work at school lab.

Key words

CGI (Common Gateway Interface), Client-side/Server-side Web Programming, Perl, PHP

Reference to textbook chapters

For basic knowledge on Perl, here is an online book: Simon Cozens, Beginning Perl. There are also plenty of online Perl tutorials over the Web.

File operations: read files

The following script readTextFile.cgi can read a text file (`lyrics.txt`) from the server and publish the content of the file to the client's browser.

```
#!/usr/bin/perl -w
use CGI qw ( :standard);
print "Content-type: text/html\n\n";
$file="lyrics.txt";

open(FILE, "<$file") or die("The file could not be opened!");
while($line=<FILE>){
   print("$line <br>");
}
close(FILE);
```

The program simply reads the file `lyrics.txt` line-by-line and outputs to the client's browser. `$file` is a variable, which is used to store the name of the input file. Similar to VBScript, we do not need to declare a variable before we use it. We also do not need to worry about the type of an variable. The statement `open(FILE, "<$file")` is used to open the file. The label `FILE` is called **filehandle**, which is used for file processing. We can view it as a reference of `$file`. Note that the symbol "<" in the statement indicates that the file is opened for *reading*. You do not need to set the text file to be 'everybody can read'. It just needs to be accessible by your script. If the file is placed in a different directory other than cgi-bin, you need to specify its relative path, say "../../documents/lyrics.txt".

To call this script, you can simply type in this URL:
http://staff.scem.westernsydney.edu.au/cgi-bin/cgiwrap/~jianhua/readTextFile.cgi
in your Web browser. You can also write a HTML file to call the script (readTextFile.html):

```
<HTML>
<HEAD>
<TITLE>Read File</TITLE>
</HEAD>
<BODY>
<FORM ACTION="http://staff.scem.westernsydney.edu.au/cgi-bin/cgiwrap/~jianhua/readTextFile.cgi"
METHOD="POST">
<INPUT TYPE="submit" VALUE="Get file">
</FORM>
</BODY>
</HTML>
```

Remember to change the URL to fix your working environment when testing your own CGI code.

File operations: write files

The code for write a message into a text file is similar. Note that the resulting file is stored in the server side.

First, let's look at the client-side program writeTextFile.html:

```
<HTML>
<HEAD>
<TITLE>Write a message</TITLE>
</HEAD>
<BODY>
<FORM ACTION="http://staff.scem.westernsydney.edu.au/cgi-bin/cgiwrap/~jianhua/writeTextFile.cgi"
METHOD="POST">
Write your message here<br>
<TEXTAREA NAME="message" ROWS=4 COLS=45></TEXTAREA>
<P><INPUT TYPE="submit" VALUE="Send">
</FORM>
</BODY>
</HTML>
```

Run the client-side program before you install the server-side CGI program to get a feeling of the HTML form. In fact, it takes client's message from a textarea and send it to the server via a CGI query. The server catches this query and write the message into a file (logfile.txt). The server side program writeTextFile.cgi is the following:

```
#!/usr/bin/perl
use CGI qw(:standard);
print "Content-type: text/html\n\n";

$content = param("message");
$file="logfile.txt";

open(FILE, ">$file") || die("Fail to open the file");
print FILE "$content";
close (FILE);
print "Thanks for your message.";
```

The program is almost the same as "`readTextFile.cgi`" except:

1. the statement that catches the message from the client side program and save it to the variable `$content`:
   ```
   $content = param("message");
   ```
2. and the statement that open the file `logfile.txt` for write:
   ```
   open(FILE, ">$file")
   ```

Note that ">" means overwriting the file. To append data to the file, use ">>".

Upload files to the server programmatically

First, create a special folder at your server (not necessarily readable and writable to every body but should be true to your script) where you are going to upload files. For instance, you can make a folder named "upload" under your home directory in Unix drive by the following commands:

```
cd ~
mkdir upload
```

Next you create a client side program to collect client's files (uploading.html):

```
<html><head><title>Uploading Files</title>
</head><body>
<form action="http://staff.scem.westernsydney.edu.au/cgi-bin/cgiwrap/~jianhua/uploading.cgi"
method="post" enctype="multipart/form-data">

What file would you like to upload?
<input type="file" name="uploadfile" size="30"><br>
<input type="submit">
</form>
</body>
</html>
```

Download the file and save it locally. Change the URL to point to the CGI code on your server.

Now the job left is to write a server side program to accept the files to upload. See uploading.cgi:

```
#!/usr/bin/perl -w
use CGI qw/:standard/;
print "Content-type: text/html\n\n";

my $file = param('uploadfile');
my $upload_dir = ">../../upload";
my $fileName = $file;
$fileName =~ s/.*[\/\\](.*)/$1/;

if ($file) {
open (UPLOAD, ">$upload_dir/$fileName") || Error ();
my ($data, $length, $chunk);
while ($chunk = read ($file, $data, 1024)) {
print UPLOAD $data;
}
close (UPLOAD);
print "<p>Thank you for uploading <b>$fileName</b>.";
} else {
print "No file was chosen.";
}

sub Error {
print "Couldn't open temporary file: $!";
exit;
}
```

In the above code, the first three line are standard. The highlighted block in blue defines the file names of the file to upload and the file to save. The variable `$file` holds the file name to upload (now on your local machine). `$upload_dir` specifies the folder "upload" your just created. `$fileName` gives the name of the file to be saved on the server. Note that the saved file name is the same as its original name except the possible path of the original file has been removed (use regular expression). The highlighted part in red reads the file to upload, block-by-block, and save it to the file on the server. The *read* function reads 1024 bytes data from $file to $data variable each time. The *print* statement saves the data from `$data` to UPLOAD, a reference of the saved file.

Password checking

The final example I am going to show you is the code for checking user id and password. The client side code password.html is the following:

```
<html><head><title>Verifying a username and a password</title>
```

```
</head><body>
<h2>Type in your username and password below. </h2>
<form action="http://staff.scem.westernsydney.edu.au/cgi-bin/cgiwrap/~jianhua/password.cgi"
method="post">
  <table><tbody><tr>
        <td>Username:</td>
        <td> <input name="username" type="text"> </td>
      </tr><tr>
        <td>Password:</td>
       <td> <input name="password" type="password"> </td>
      </tr><tr>
        <td> <input type="submit" value="Enter"> </td>
      </tr>
    </tbody></table>
</form>
</body></html>
```

As usual the URL points to the CGI program `password.cgi`. Two inputs widgets accept user name (as text) and password (as password), respectively. Note that the Web browser can automatically blind user's input of password.

The server side program (password.cgi) is a bit more complicated.

```
#!/usr/bin/perl -w
use CGI qw( :standard );
print "Content-type: text/html\n\n";

$testUsername = param( "username" );
$testPassword = param( "password" );

open(FILE, "password.txt" ) || die( "The database could not be opened." );
while ( $line = <FILE> ) {
    chomp( $line );
    ( $username, $password ) = split(",", $line );
    if( $testUsername eq $username ) {
        $userVerified = 1;
   if( $testPassword eq $password ) {
$passwordVerified = 1;
last;
        }
    }
}
close( FILE );

if ( $userVerified && $passwordVerified ) {
    accessGranted();
}elsif ( $userVerified && !$passwordVerified ) {
    wrongPassword();
}else {
      accessDenied();
}

sub accessGranted{
    print "Permission has been granted";
}

sub wrongPassword{
    print "You entered an invalid password. Access has been denied.";
}

sub accessDenied{
    print "You have been denied access to this site.";
}
```

The core of the program is the highlighted part in red. It reads the `password.txt` file line-by-line. Each line of the file is in the format: **username**, **password**. The `chomp` function removes any terminating newline characters from its parameter. The `split` function splits each line into two parts and store the result into the array `$username, $password`. The two `if` statements check user name and password, respectively (match the information provided by a user to records stored in a file). The rest of the code is trivial.

Tips for debugging a Perl program

1. Always start from the simplest code and make sure it works well. Add more functionalities one by one. Once you implement a new functionality, save it to a backup folder before you add more functionalities to it.
2. A server side program is supposed to be small. Too complicated programs would slow down the server.
3. A CGI program needs to be everybody executable. However, you should keep other file to be safe, especially the password database.
4. Perl and Unix commands are case sensitive.