
Computer and Automation Research Institute
Hungarian Academy of Sciences

# *Web Scripts, Gateways and Forms*

**P. Kacsuk**

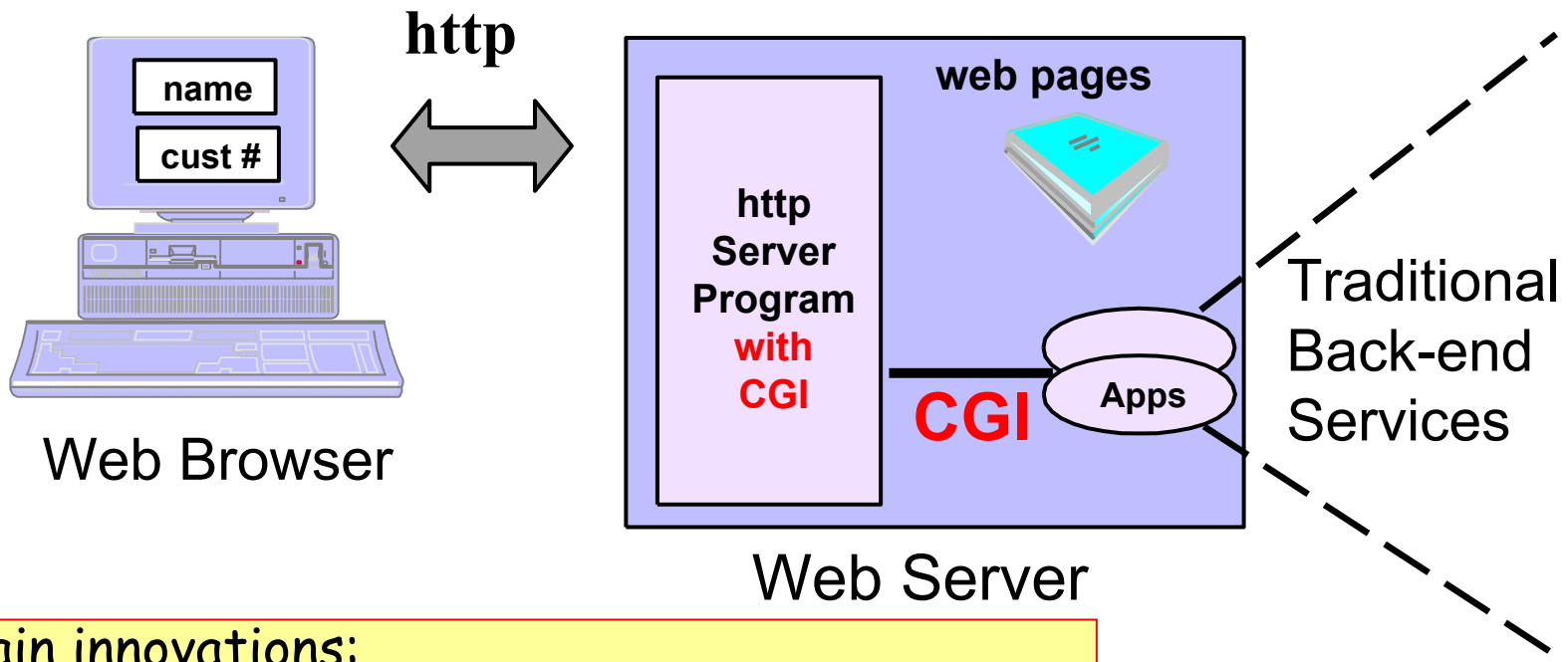
***Laboratory of Parallel and Distributed Systems***

***MTA SZTAKI Research Institute***

**kacsuk@sztaki.hu**

**www.lpds.sztaki.hu**

# Script-based Web




Main innovations:

1. Fields, forms - dynamic page content
2. **Server-side processing** - CGI
3. Access to legacy data

1. **A program on a server, triggered by input from a browser**
2. **The CGI program links the HTTP server to another program, such as a database or transaction system**

# What is a script?

- A web script is a program that can be executed by the web server in response to a web request.
  - Any program can be a web script
  - It can call other programs or contact other servers
- Goal of a web script:
  - To act on the web server on behalf of the web client
  - To extend the capabilities of the ***`httpd`*** **daemon**

# How does a script work?

- The ***httpd*** **daemon** starts the script, passes the data from the browser to the script and vice versa
- The ***httpd*** **daemon** should do the following steps:
    1. determine that the request is really a program and not a document
    2. locate the program and determine if it is permissible to execute
    3. start the script program and ensure that the input from the client will be passed to the script
    4. read the output from the script and pass it back to the client
    5. send an error message back to the client if something goes wrong with the script program
    6. close the network connection when the scipt completes


# What files are executable scripts?

- The server software will execute scripts only according to specific rules, as configured by the system administrator. These can be:
  - Scripts must be located in a particular directory and must have UNIX execute permission
  - The server can be configured to allow any file with a particular name, such as files ending in .cgi


# Making the script run: The Common Gateway Interface

- CGI is standard protocol for how scripts are to be called and how data is passed between the httpd daemon and the script.
- The CGI standard is actually a suite of standards, one for each operating system. E.g. for UNIX:
  - the script executes as a process
  - input data from the browser are passed through standard input and UNIX environment variables
  - results passed back through standard output
- CGI is one of the many possible standards:
  - Netscape servers use NSAPI (Netscape Server Application Interface)
  - Microsoft Windows platforms use OLE or DLL


# A User's view of a script

- Assume the client user would like to know how many users currently logged on in a web server.
- There are two ways to do it:
  - If she has an account on the web server, she can login and call the **`uptime`** UNIX utility
  - If she has no account on the web server, she can run a script on the server using her web browser and this script will call the **`uptime`** UNIX utility on her behalf
- Assume the name of the script: **`how_busy_are_you`**
- The client should request:

```
GET /scripts/how_busy_are_you HTTP/1.0
```

- And the httpd server would execute the script and return the result:

```
1.29pm up 21days, 4:35, 5 users, load average:0.00
```


# The how_busy_are_you Perl script

```
#!/usr/local/bin/perl
# Perl script to execute UNIX ‘uptime’ program
$UPTIME = ‘usr/ucb/uptime’;
select(STDOUT); $| =1; # make the output unbuffered so it
                       # is written immediately to httpd
# print the MIME type of the data tp follow
print “Content-type: text/plain\n\n”;
# run the real “uptime” and send the output to httpd
if (-x $UPTIME) {
    exec($UPTIME);
} else {
    print “Cannot find uptime command.\n”;
}
```


# A web server's view of scripts

**STEP 1:** Wait for a new request
**STEP 2:** A request arrives from a client
**STEP 3:** The server parses request
**STEP 4:** Read other information from network
**STEP 5:** Do the method requested
**STEP 6:** Close file, close network
**STEP 7:** Go to step1


# STEP 1: **Wait for a new request**

- The httpd program waits for a request to arrive from a client somewhere on the Internet

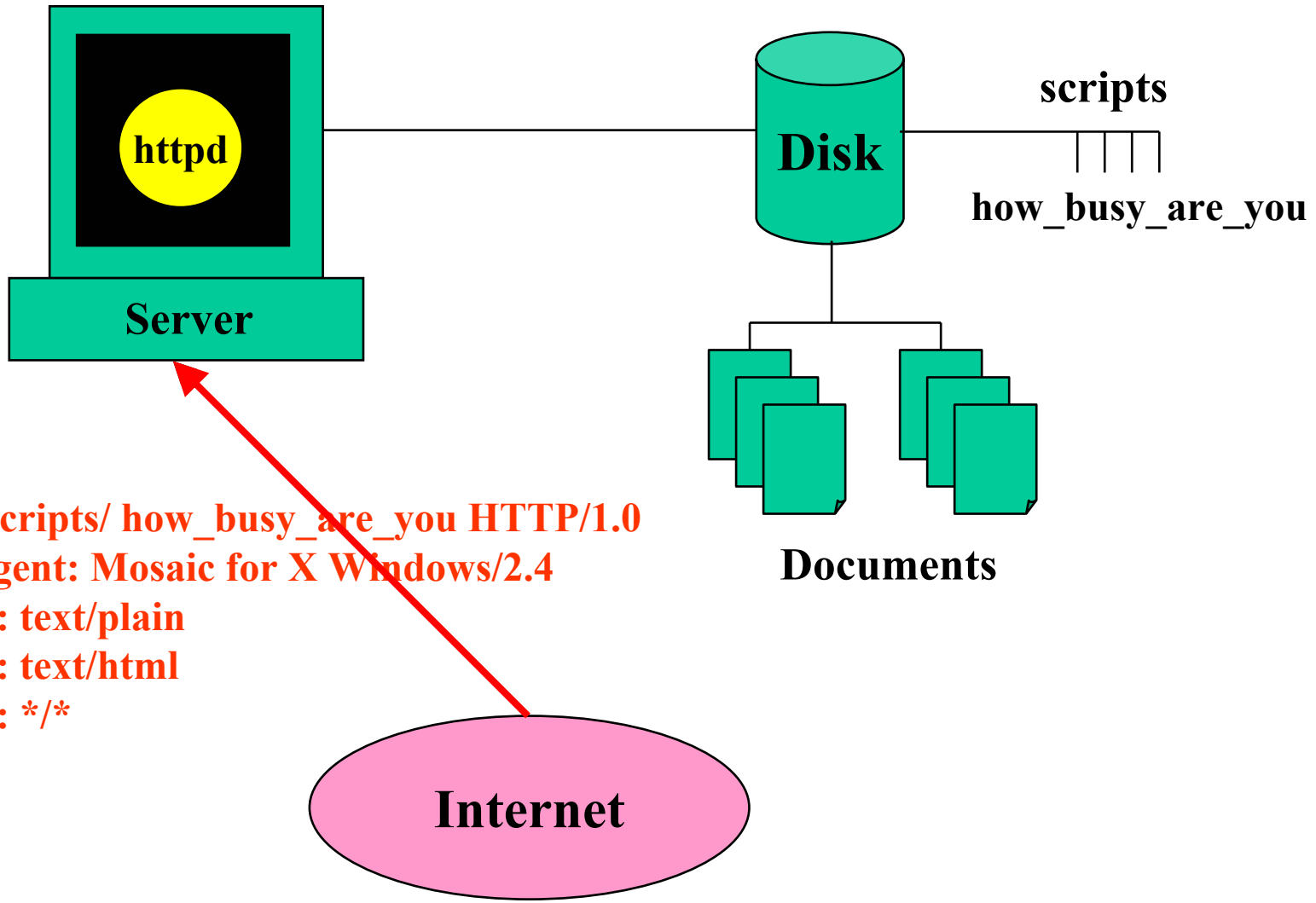
# STEP 2: *A request arrives from a client*

- The action begins, as usual, when someone requests a documents selecting a URL:

  http://www.server.org/scripts/how_busy_are_you
- The client locates the server and initiate a connection
- Once the connection is established, the client sends the request according to the HTTP protocol:

  GET /scripts/how_busy_are_you HTTP/1.0
- Notice that Step 1 and 2 are exactly the same as in case of a normal document request

# STEP 2 - 4


httpd
Server
Disk
scripts
how_busy_are_you
Documents
GET /scripts/ how_busy_are_you HTTP/1.0
User-agent: Mosaic for X Windows/2.4
Accept: text/plain
Accept: text/html
Accept: */*
Internet


# STEP 3: The server parses request

- The httpd server decodes the request and discovers that

  scripts/how_busy_are_you

  is supposed to be a script.
- To return the result it should use version 1.0 of the HTTP according to the GET command:

  GET /scripts/how_busy_are_you HTTP/1.0
- There is no need for the server to find the client on the Internet since it will use the same connection on which the request arrived.


# STEP 4: Read other information from network

- The httpd daemon reads the rest of the request as in the case of normal documents:

  **User-agent: Mosaic for X Windows/2.4**

  **Accept: text/plain**

  **Accept: text/html**

  **Accept: */***


# STEP 5: Do the method requested (in UNIX)

- The httpd server must
  - locate the script

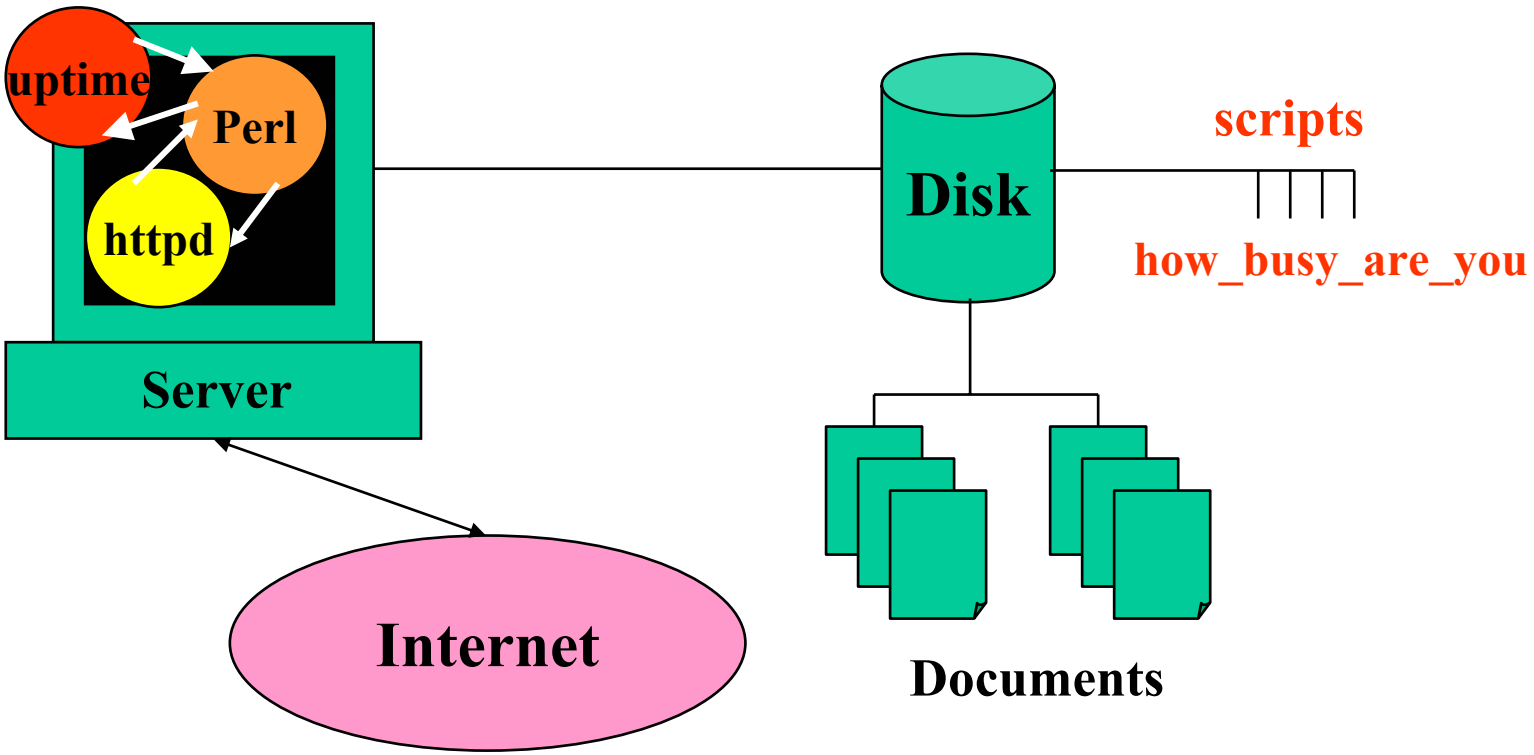
    scripts/how_busy_are_you
  - confirm that it is an executable program
  - discover that it is a Perl script
- The httpd daemon must create the correct execution environment:
  - with approriate UNIX shell environment variable set
  - with standard input and output
  - starting the Perl program
- The Perl program executes each line of the script as if it were typed from a keyboard:
  - starts the ‘uptime’ program
  - the results of the uptime program is relayed to httpd

# STEP 5: Do the method requested

a. Find script, determine that it is OK to execute; discover that it is a Perl script

b. Set up the execution environment: standard input/output and environmental variables; start the Perl program

c. The Perl program starts /usr/ucb/uptime

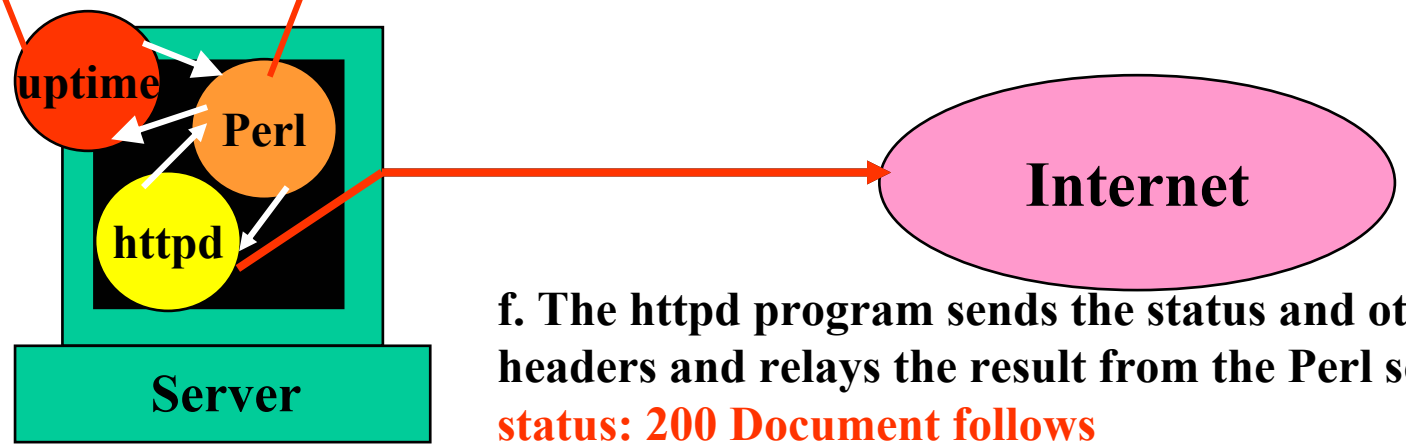



The input and output of the programs are connected, so data flows between them.

# STEP 5: Do the method requested

**d. uptime prints out the result to the Perl script:**
**11:35am up 7 days, ...**

**e. The Perl program prints out the Content-type and relays the result to httpd:**
**Content-type: text/plain**
**11:35am up 7 days, ...**




**f. The httpd program sends the status and other headers and relays the result from the Perl script:**
**status: 200 Document follows**
**Server:NCSA/1.4**
**Date: Thu, July 20 1999 17:35:18 GMT**
**Content-type: text/plain**
**11:35am up 7 days, ...**


# STEP 6: **Close file, close network**

- **A.** When the output of the script is completely sent, the script terminates and closes all the connections to the httpd program
- **B.** The httpd program closes the network connection

- **STEP 7:** The httpd server is now ready for another request.


# STEP 5: Do the method requested (in case of error)

- If the file requested is not found or could not be executed, the request cannot be satisfied:
  - the status code will be something other than 200
  - if the name of the script was misspelled, the status code: 403
- It is also possible that the script executes but encounters an error.
  - In such case the uptime program sends a textual error message
  - since the program is returning information the status code: 200


# Thank you