Pegasus

— another web server for Plan 9 —

Kenji Arisawa, Aichi University
arisawa@aichi-u.ac.jp

ABSTRACT

Pegasus is another web server for Plan 9 operating system. Pegasus enables natural management of web documents: creator of web pages should be able to do everything for the pages. To enable such management in secure manner, the server is running in name space that is separated from other user's documents. This feature comes from novel ability of Plan 9: per process name space configuration. This paper presents briefly the feature of Pegasus.

1. Introduction

World Wide Web has grown to be one of the most important information service today. Common Gateway Interface (CGI) plays important role in the service. Some services accept data that are written in input fields of a browser, and process the data so that the server can respond to the request of the client. Some web documents are created on the fly by the server so that the client can look most recent information constructed from accumulated data in database. These two examples are typical cases that servers use CGI mechanism in servicing web documents.

It is said that writing a secure program for CGI is difficult even for professional programmers. CGI programs working on regular input data without problems have sometimes security halls. You will find many examples of such cases on W3C Security FAQ[3].

The problem comes from the fact: commonly used CGI programs can access whole name space of the system as shown in Fig.1, where "real space", illustrated by outer rectangle, denotes the whole name space of the system.

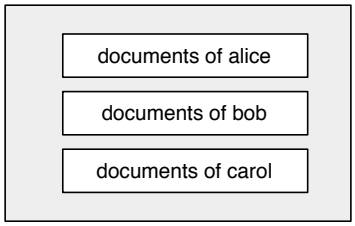

Fig.1: Traditional web server.
real space = httpd space

Httpd space is illustrated by light grey color, where the term "httpd" means a server side program that is running in communicating with the clients. Httpd space is a set of files in which the httpd is servicing to the client. Httpd space is also a set of files that are seen by

CGI programs. Httpd space is exactly equal to real space in traditional web server. Therefore poorly-written CGI programs can be a back door to all files in the system.

We can find some efforts to overcome this problem by adding a wrapper program to an existing httpd: cgiwrapper enables user's CGI scripts to run as the user's privilege so that other user's files can be protected from these script; sbox is a sort of cgiwrapper that optionally restricts access space of user's CGI script to his/her home directory by using "chroot" system call of UNIX.

How does httpd for Plan 9? One of the novel features of Plan 9 is the concept of name space[1]: the name space is dynamically configurable by each process. The configurability is much smarter than that of UNIX. The configured name space of Plan 9 never affects other processes except the child processes. The configured information is not in disk storage; the information is not left even if the httpd or any child process such as CGI process are accidentally terminated. Therefore we expect: we will have much smarter and much safer httpd than that of UNIX world.

The name space of official httpd from Bell-labs is illustrated in Fig.2.

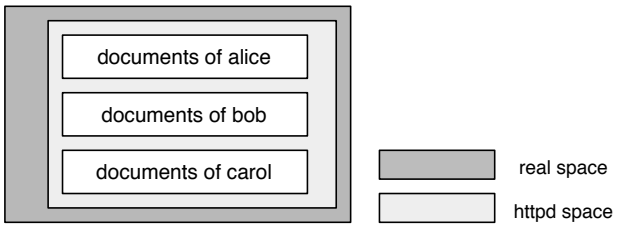

Fig.2: Name space of Plan 9 official web server.

real space $\supset$ httpd space

The advance is in that the httpd is confined in the small subset of the real space. But why the httpd does not go farther? Isn't it possible for Plan 9 to separate user's documents each other as illustrated in Fig.3?

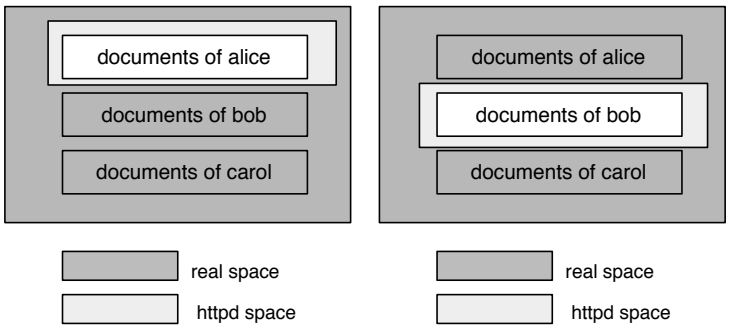

Fig.3: Name space of Pegasus.

Separation by name space is essential for secure CGI service as explained in sbox document[13]: httpd that is running as single user such as nobody makes a problem because if someone make poor coded or malicious coded CGI, the CGI might affect to other users files; on the other hand, httpd that is running as user's privilege of the CGI owner brings other problem because all the files of the user is in risk.

Pegasus is another httpd in Plan 9 world. Pegasus is designed to be a secure and flexible httpd that is based on the novel feature of Plan 9. This paper shows how "per process name space configuration" can be applied to httpd and shows how the users get benefit from the ability.

2. Pegasus Specification

2.1. The history

The first version of Pegasus was shipped to Internet in Jan 2002 as version 0.9 after some preceding experiments. The goal was to achieve comfortable CGI environment with an idea of separated name space. Version 1.0 was released in Feb 2002 with execution handler similar to that of current version. Version 2.0 (Feb 2003) established the current feature of Plan 9: uniformity of httpd name space and document management among real host, virtual host, and regular users.

This section illustrates briefly basic feature of Pegasus 2.2. Most of them are same as previous version. The important advance is in the support of CGI/1.1 and digest authentication.

Currently, Pegasus 2.3 is in testing. Bugs that are found in developing the new version are commented in footnote.

2.2. The concept

Suppose we have four persons : three system users and one system administrator. They are

Alice: a regular user who has web documents in the system

Bob: a user who has web documents for real host

Carol: a user who has web documents for virtual host

David: a system administrator

Alice has files for her web service under the directory "`/usr/alice/web/`". Her documents are under "`/usr/alice/web/doc/`", and files for access controls are in "`/usr/alice/web/etc/`". We refer "`/usr/alice/web`" as httpd root of alice. Unless disabled or changed in "`/sys/lib/httpd.rewrite`", all users are given their httpd root at "`/usr/$user/web/`" where `$user` denotes the user's name.

The location is fixed for each user. On the other hand, httpd roots of real host and virtual hosts are configured in "`/sys/lib/httpd.rewrite`". Let the httpd root for bob be "`/usr/bob/www/`", and the httpd root for carol be "`/usr/carol/www/`".

Basic structure of the name space under httpd root is uniform: each httpd root has a set of common subdirectories. Let $web be a path to the httpd root for a user. Then the directory

```
$web/doc/
```

is document root of the user, and the directory

```
$web/etc/
```

is the place for access control files, and the directories

```
$web/bin/$cputype/
```

(where "$cputype" is `386` for PC compatible machines,) and

```
$web/bin/rc/
```

are the places for executable binaries and scripts that is owned by the user. Following instructions in "`/lib/namespace.httpd`", Pegasus merges these executables with the executables that are offered by david. The merge is virtual, i.e., the result is not in disk storage. The magic

comes from “bind” system call of Plan 9. The result will be seen in “/doc/” and “/bin/” in httpd space.

In servicing documents of alice, Pegasus will see only files under “/usr/alice/web/” and files that are offered by david as illustrated in Fig.4.

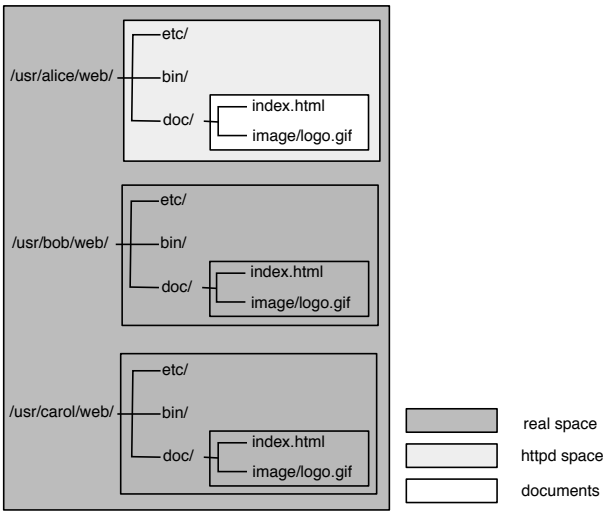

Fig.4: Name space of Pegasus.

The figure shows that the file “/usr/alice/web/doc/index.html” will be seen as “/doc/index.html” by httpd. Thus the files outside of “/usr/alice/web/” are essentially hided from httpd except files that are made visible by david.

Tasks that are required by alice, bob, and carol are same: creating documents, removing documents, moving documents, modifying documents, setting access control, setting redirection, using CGI scripts, configuring CGI handling, etc. Web server should be designed as follows: all that is concerned with their documents should be in their own hands, and furthermore, their tasks should never interfere with each other.

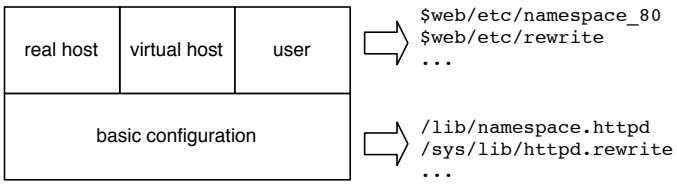

Fig.5: Management layer of Pegasus.

The tasks of david, system administrator, should be minimum: only basic configuration that are common for all users as illustrated in Fig.5.

The concept of “httpd root” of Pegasus is different from that of “server root” of main httpd servers such as Apache. Traditional concept of server root is merely a directory in which httpd is installed. That does not set a boundary of name space for httpd. On the contrary, httpd root of Pegasus has an essential meaning: that sets a boundary of visible name space and invisible name space for httpd. Httpd root of Pegasus is multiple, uniform, and essential. Pegasus is designed to achieve the concept.

2.3. Support level

Pegasus 2.2 supports the following features.

- HTTP/1.1 (RFC2616)
- CGI/1.1 (RFC3875)
- GET, HEAD, POST methods with some tools
- Virtual hosts (IP-based and name-based)
- Basic and digest authentications (RFC2617)
- HTTPS (RFC2818)

2.4. Access control

HTTP clients send a request to the server with path name of a file in the URI, then the following files are inaccessible.

- Files that are not placed under document root
- Files that are unreadable by httpd
- Files with a name that begins with "`.`"

Some files in "`$web/etc/`" enable additional access control.

- "`$web/etc/namespace_$port`" reconfigures the name space that is provided by "`/lib/namespace.httpd`", where "`$port`" is a port number: most commonly 80, for http.
- "`$web/etc/allow`" disallows some clients with specific IP to access some files or directories.
- "`$web/etc/passwd`" requires clients to be authenticated in accessing specific files or directories.

Assume that the requested file exists and is accessible. Then, regular response of the server is just to send the content of requested file. Another type of response is to invoke a script according to the rules in "`$web/etc/handler`". The script has ability to send content of any readable file in httpd space.

2.5. Execution handler

In this subsection, the word "script" refers any executable file including binary files.

The file "`$web/etc/handler`" connects requested path to the action; the sample contents of the file are shown below.

```
# path               mimetype   hctl execpath arg ...
/netlib/*/index.html text/html  0    /bin/ftp2html
*.http               -          1    $target
*/nph-*.cgi          -          1    $target
*.cgi                text/html  +    $target
*.html               text/html  0    $target
*.tt                 text/html  0    /bin/peep $target
```

First field named “path” shows a path pattern that is same as that used in Plan 9 shell “rc” [2] but for two exceptions: slash ‘/’ is not a special character and “/*/” matches “/”. Thus “*.cgi” means a file with suffix “cgi”. The field is compared with the requested path name on URI. More strictly speaking, requested path name from document root is compared. That is, if URI path is “/~alice/foo”, then the first field is compared with “/foo”. All terms “requested path” in this subsection follow to this meaning.

Second field named “mimetype” is the default mime type[1] for the HTTP header “Content-Type”. If ‘-’ is specified and “hctl” is not ‘1’, the “Content-Type” is guessed by the server.

Third field named “hctl” takes three values ‘1’,‘+’, and ‘0’ that mean control level to the HTTP headers by the script; the meanings are as follows:

- 1 Full control by the script
- \+ Partial control by the script
- 0 No control by the script

If ‘1’ is specified the script has responsibility to write all HTTP headers including “Content-Type” and “Content-Length”; the script is called non-pursed CGI in RFC3875. HTTP headers must be separated from HTML headers by a single blank line, i.e., a line that contains only “\n” code.

If ‘0’, the script must not write HTTP header. The header is provided by httpd.

Finally if ‘+’, the script may contain HTTP headers in compliance with CGI/1.1.

The fields greater than third define a command and its arguments that will be invoked when incoming requested path matches the first field. In these fields, $target is the path name prepended “/doc” to the requested path.

The last line in the sample handler is interpreted as follows: If a file, say “/foo/bar.tt”, is requested, then “/bin/peep” is invoked with an argument “/doc/foo/bar.tt”.

Of course files in fourth field must be executable. If the fourth field is $target, then the file must be executable. On the contrary, $target after fourth field needs neither be executable nor be existing.

Note that most browsers do not judge mime type by looking the content but judge only by looking the file suffix. Therefore it is safe to put popular suffix adequate to the content.

Executable file with a suffix “http” and “html” first appeared in Ykhttpd by Charles Forsyth. Sample configuration shown above inherits his idea.

2.6. Character encoding

Plan 9 is based on UTF-8 encoding. Not only the path name is UTF-8, but also the system tools are consistently supporting UTF-8. Thus, we can execute the following commands without seeing any oddness which are still common in UNIX world.

```
term% mkdir こんにちは
term% ls
...
こんにちは
 ...
term% cd こんにちは
```

[1] Pegasus 2.2 had a bug if mine type is not “text”. The problem is fixed in testing version.

Nowadays, UTF-8 encoding is accepted as a standard of Internet world. IRI(International Resource Identifiers) described in RFC3987 claims to stand on UTF-8 for path part of URI and main browsers are already supporting IRI. In Fig.6, You will observe that “こんにちは” is shown as it is in the URL field of the browser.

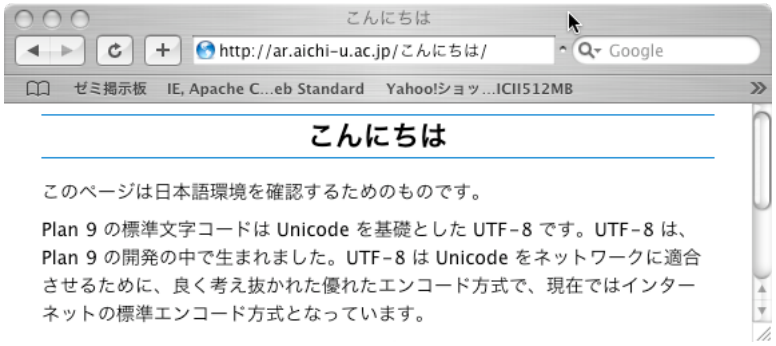

Fig.6: IRI support by Mac/Safari.

As shown in this figure, HTML documents that are placed in the directory “こんにちは” is accessible by the HTTP client.

For Plan 9, the support by httpd is very simple. The feature is already achieved by official httpd of Plan 9. In supporting IRI, Pegasus added only a little modification to official httpd for special feature of redirection of Pegasus. Plan 9 is excellent with IRI.

It is a task of browsers to interpret `href` value such as

```
<a href="http://ar.aichi-u.ac.jp/こんにちは/">こんにちは</a>
```

and to transform it to an appropriate URI in accessing the host. Nowadays, many browsers have the ability. However UTF-8 string in query part of `href` value has still problems in some main browsers. Therefore an auxiliary tool named “hfmt” is provided to compensate the problem.

IDN(International Domain Name) described in RFC3490 is not supported yet, because it seems the needs is very small even in Japan.

3. Administration (1) – system level –

This section explains some works of system administrator that is required in installing Pegasus 2.2.

3.1. Invocation

Pegasus has a couple of commands that invoke httpd.

- httpd
- mon

Httpd of Pegasus can run either as user “none” or as a user who invoked httpd. It has many options as shown below.

```
httpd [-Mfmsuw] [-p port] [-c certificate]  [...]
```

where option "-s" means "server mode", option "-u" means that it runs as a user who invoked httpd, and option "-M" indicates that the httpd is invoked by mon. Other options are not explained in this paper. Readers can find the information from elsewhere[14].

In regular circumstance, httpd is accessed very frequently. In such case, running server mode is recommended.

If httpd is running as user "none", there exists a security problem: private data that is to be read by the httpd would not be protected from system users, because system users can become user "none". Therefore it is necessary for secure web service to run the httpd as some fictional user name, say, "web".

How to invoke httpd that runs as user "web" without password so that httpd can be automatically invoked in starting cpu server?

Mon is provided for this purpose. The ability is based on "devcap" that is introduced in Plan 9 ed.4.

Mon must be invoked by the privilege of host owner so that mon can invoke httpd as a daemon of any user. The command syntax is as follows:

```
mon [-d] [-u user] [-r req] cmdpath argument ...
```

where option "-d" implies daemon mode, and option "-u user" indicates mon to invoke httpd as the given user. The option "-r req" is used for https support, where "req" is factotum format of X.509 certificate signing request[15].

An example of the usage is shown below.

```
b=/usr/local/bin/$cputype
$b/mon -du web $b/httpd -suM
c=/sys/lib/tls/cert
k=/sys/lib/tls/key
$b/mon -du web -r $k $b/httpd -suM -p443 -c $c
```

This example is extracted from "/rc/bin/cpurc" of the author's cpu server that supports both http and https service. The output of "ps" command are shown below. They are live example.

```
ar% ps|grep 'httpd|mon'
bootes      67    0:00   0:00     76K Await    mon
bootes      70    0:00   0:00     76K Await    mon
web      92245    0:03   4:15   1516K Open     httpd
web      92246    0:00   0:00   1516K Open     httpd
web     149886    0:00   0:00   1644K Pread    httpd
ar%
```

We observe three httpds in this example. The first and the second httpds are servicing TCP port 80 and 443 respectively. Both were invoked by mon, and waiting for requests from clients. The third httpd is a child process of the first, and is now in service.

If httpd was not killed, PID of httpd that is invoked by mon would be the next to the PID of mon. However in the above case, the httpd was killed for update. What happens then? Mon monitors termination of the daemon so that mon can immediately invoke next new httpd. This facility is a protection against malicious CGI that tries to stop service by killing httpd.

Note that mon is running as user "bootes", not as user "web". This is because we should protect mon from CGI.

3.2. /adm/users

Add a line in /adm/users:

```
web:web:web:
```

so that Pegasus can run as user "web".

Don't assign password for user "web".

Suppose a user, say alice, wants to have CGI environment, and wants to have files that must be readable only by httpd and by herself. How to do that? Preventing other system users to enter her httpd root will be enough:

```
chmod 750 $web
chgrp web $web
```

though administrator's help is required to change the group[2]. Then alice can safely make her file "foo" readable to user "web" by executing a command:

```
chmod 644 foo
```

3.3. /sys/lib/httpd.conf

A file "http.conf" in "/sys/lib" configures basic parameters of Pegasus: server name, the location of "base", etc. The contents in the distribution are shown below. They are all comments.

```
# myname     www                    # server name (default is your sysname)
# base       /usr/web               # base directory for Pegasus
# namespace  /lib/namespace.httpd   # name space configuration
# rewrite    /sys/lib/httpd.rewrite # system rewrite file
## parameters below might be required tuning
## unit of time is second
# allowbasic    0           # 1: allow basic authentication
# parsetimeout  15          # timeout to parse header
# waittimeout1  15          # wait timeout for a non-authenticated client
# waittimeout2  900         # wait timeout for an authenticated client
# alivetimeout  15          # keep alive timeout
# cgitimeout    5           # timeout for CGI
# posttimeout   900         # timeout to get POST data
# connectlimit  300         # limit of total open tcp connections
# maxpost       10          # post data size restriction (in unit of MB)
# obstime       3           # observation time to detect burst access
# maxaccess     20          # allowed max access in obstime
# lockouttime   180         # lockout time for maxconnect and maxaccess
# contmax       100         # max persistent continuation count for safty
```

Blank lines and lines beginning with '#' in this file are ignored. Following the '#' , attribute and the default value are shown as a guide. Only first and second fields are consulted. Text that follows second '#' shows the meaning.

[2]It is more convenient and natural if alice could change the group of "$web" without help of administrator. However Plan 9 does not allow the operation. Plan 9 does not have SUID nor SGID concept in UNIX, therefore it seems better to allow a user to change group of his/her files to any group.

3.4. /lib/namespace.httpd

Name space is configured in "/lib/namespace.httpd" by default. The typical example that supports CGI is shown below.

```
bind -a /usr/web/bin/$cputype /bin
bind -a /usr/web/bin/rc /bin

bind /sys/lib /usr/web/sys/lib
bind /lib /usr/web/lib
bind /bin /usr/web/bin
bind /rc/lib /usr/web/rc/lib
bind -c #e /usr/web/env
bind #p /usr/web/proc
bind #c /usr/web/dev

# change the line below subject to your needs.
bind /usr/bob/netlib /usr/bob/www/doc/netlib
```

Bind operation in the first line is to the "/bin" which contents are already configured in "/lib/namespace". We will see all executables of Plan 9 in the "/bin". Therefore, if david, system administrator, does want to restrict executables for CGI only to files in "/usr/web/bin/$objtype" and "/usr/web/bin/rc", then he would rather adopt

```
bind /usr/web/bin/$cputype /bin
```

instead of the first line. The operation replaces the contents in "/bin" by the contents in "/usr/web/bin/$cputype".

The third line

```
bind /sys/lib /usr/web/sys/lib
```

exports "/sys/lib" to httpd space. Be careful in exporting "/sys/lib". There are files that must be secret; the example is "/sys/lib/tls/key". Examine the permission bit.

3.5. /sys/lib/httpd.rewrite

The "/sys/lib/httpd.rewrite" has two roles:

- Redirects incoming HTTP requests to other URI,
- Specifies the location of httpd root of real host and virtual host.

The example is shown below.

```
# car.shop.com is the virtual host for carol
# the uri is http://car.shop.com
http://car.shop.com    */usr/carol/www
# emili has moved to bell-labs
/~emili    http://plan9.bell-labs.com/~emili
# real host must be the last
/    */usr/bob/www
```

Lines beginning with '#' are comments. Other lines consist of two fields.

The first field is a URI pattern. It begins with '/' or with scheme name followed by ':'. Request to the real host or the user is compared with the pattern that begins with '/'. The matching rule is very simple. The request that begins with the first field is regarded as matched. For example, if the request is "/~emili/foo/bar.html" then the request matches "/~emili".

The comparison is name-wise in name tree. Therefore "/~emilia/foo/bar.html" does not matches "/~emili" as it should be. The same matching rule is adopted to user level redirection in "$web/etc/rewrite" and access control in "$web/etc/allow" and "$web/etc/passwd".

The second field beginning with '*' shows the httpd root for real host or virtual host or regular user; the default httpd root for regular user is "/usr/$user/web/" where "$user" denotes the user name. Others are URI to which request is redirected. The reminder of the requested URI (i.e., "/foo/bar.html" in case the request is "/~emili/foo/bar.html") is appended to the second fields so that the request jumps to "http://plan9.bell-labs.com/~emili/foo/bar.html" and browsers can open the page there.

3.6. /usr/web/

The directory "/usr/web/" is default base directory for Pegasus on which name space of Pegasus is configured.

For regular documents, the following three directories are required.

```
/usr/web/doc/        # empty
/usr/web/etc/nonce/ # empty
/usr/web/mnt/        # empty
```

where directory "nonce" must have full permission (i.e., read/write/executable) to httpd daemon. The directory is a database to support digest authentication.

For CGI, more directories will be required as shown below.

```
/usr/web/bin/386/
/usr/web/bin/rc/
/usr/web/env/        # empty
/usr/web/proc/       # empty
/usr/web/dev/        # empty
/usr/web/rc/lib/     # empty
/usr/web/tmp/        # empty
```

You can add more directories depending what to do in CGI script. The followings are the example.

```
/usr/web/sys/lib/    # empty
/usr/web/lib/        # empty
/usr/web/net/        # empty
```

3.7. /sys/log/

Pegasus uses the following two log files:

```
http
blacklist
```

in "/sys/log/", where "http" is a regular log file for monitoring tasks of httpd, and "blacklist" is a log file to detect overloaded clients that might be an attack. Then service to the client is stopped during 3 minutes[3] and the IP address is put into the "blacklist". However some overloaded clients might not be an attack. Then parameters in "/sys/lib/httpd.conf" should be adjusted.

4. Administration (2) – user level –

Imagine again three system users alice, bob and carol having web documents in "/usr/alice/web/doc/", "/usr/bob/www/doc/", and "/usr/carol/www/doc/" respectively.

We denote their httpd root by "$web". The value is "/usr/alice/web/" for alice, and "/usr/bob/www/" for bob, etc. Then "$web/doc/" is the document root for the user.

4.1. $web/doc/

The directory "$web/doc/" is the document root for the user. Httpd will see web documents under "/doc/" as if "$web" is nonexistent. For example, "$web/doc/index.html" is seen as "/doc/index.html" by httpd.

4.2. $web/bin/

The directories under "$web/bin/" are the place for user's own executable files. They are

```
$web/bin/$objtyep/
$web/bin/rc/
```

where "$cputype" stands for the type of CPU; the value is 386 for PC compatible machines. Files in these directories will be added using bind operation to "/bin".

4.3. $web/etc/

The directory "$web/etc/" is the place for access control files. The files are listed below. They are all optional.

```
$web/etc/allow
$web/etc/passwd
$web/etc/rewrite
$web/etc/handler
$web/etc/namespace_$port
```

The files "$web/etc/allow" and "$web/etc/passwd" control the access by IP address and by password respectively. Thus, access control is in user's hand as it should be.

The "$port" stands for a port number; the value is 80 for regular httpd service. Other port numbers are also acceptable.

Users can configure name space in the file "$web/etc/namespace_$port" standing on the name space offered by system administrator. The configuration is applied not only to CGI scripts but also to regular documents. Note that any CGI script can additionally configure name space according to it's own needs in starting service.

[3] The time can be changed in "/sys/lib/httpd.conf".

Users can write their own rewrite rules in "$web/etc/rewrite". This feature is important for document management, because an access to some documents should be redirected to a secure scheme such as https.

Note that CGI handling is not the task of administrator. Users have a large variety of needs in constructing their web pages and the CGI scripts are connected to the web pages. Therefore it is desirable that users are allowed to have CGI scripts. In Pegasus, all about CGI are completely in user's hand. See "Execution handler" for more details.

4.4. $web/etc/rewrite

Suppose carol wants to redirect a request, say "http://car.shop.com/private", to a secure scheme such as "https://car.shop.com/private", where "car.shop.com" is her virtual host. Then carol will write a line

```
/private    https://car.shop.com/private
```

in "/usr/carol/www/etc/rewrite". This line implies that incoming request to "/private/foo" is redirected to "https://car.shop.com/private/foo", where "foo" stands for any path including empty path. Then, what happens if the request is already based on https?

Pegasus protects itself from redirection loop such as

```
/bar    /baz
/baz    /bar
```

or more complicated redirection loops by counting loop counts if they appeared in the same TCP connection. The mechanism will work if the client is based on HTTP/1.1 and keeps connection on redirection to the same host and port[4]. Note that the redirection loop is safe also for clients based on HTTP/1.0 because these clients do not support automatic redirection.

The first example is the simplest one: incoming URI is same as redirected URI if incoming scheme is https. Then Pegasus does not redirects the request. This feature makes the first example meaningful and useful.

Note that first field is a path from document root, and the second filed is a URI. Therefore alice, a regular user, must write as follows[5]:

```
/bar    /~alice/baz
```

if she wants to redirect "/bar" to her document "/baz".

4.5. $web/etc/allow

Pegasus manipulates allow/deny control in a single file "$web/etc/allow". The example is shown below.

```
/foo
        192.168.*.*
/foo/bar
        ~192.168.1?.*
        *
```

[4]Some browsers reconnect on redirection even if it is to the same host and port. Then other mechanisms such as maxaccess protect Pegasus from the redirection loop. In addtion, some browsers also detect redirection loop to stop the loop.

[5]Pegasus 2.2 had a bug in redirection for regular user. The bug is fixed in testing version.

A line that begins with '/' denotes path name from document root. The name "/foo" matches "/foo/" and any deeper path than "/foo" such as "/foo/bar".

IP pattern block follows the path name. A line that begins with space or tab shows IP pattern. The pattern is shell style matching. A symbol '~' at the beginning of IP pattern means negation.

Thus files under directory "/foo/bar" is accessible only with the IP of the pattern "192.168.*.*" except "192.168.1?.*".

Note that a pattern "~*" is set implicitly at the end of each IP pattern block.

4.6. $web/etc/passwd

Pegasus supports both basic and digest authentication. Basic authentication is disabled by default but can be enabled in "/sys/lib/httpd.conf".

In Pegasus, any user can control accesses to his/her files by authentication. A file "$web/etc/passwd" is provided for the authentication. The content is, for example,

```
alice 3a58b912829a2e4b4720c3a41e58dd29 /private alice@hera
```

The single line supports both basic and digest authentication. The line consists of four fields: user, key, path, and realm.

Key is produced by using shell commands of Plan 9 as follows:

```
echo -n 'user:realm:password' | md5sum
```

For example, assume that alice wants to set a password "black cat" to the realm "alice@hera", then alice will get the key as follows:

```
echo -n 'alice:alice@hera:black cat' | md5sum
```

which will produce the key in the example.

Old format which supports only basic authentication is also supported for compatibility.

4.7. Special directory in document root

Directory "$web/doc/~" is special. This feature came from "/favicon.ico" problem: certain browser tries to get "favicon.ico" of the real host every time in accessing user's(say, alice's) document. Then, old Pegasus closed the connection to direct the browser to the name space of real host. The browser came back again to the name space of alice with new connection. The rash of redirections and reconnections will reduced the efficiency.

Pegasus 2.2 admits the needs for the user to refer some of files in real host keeping the connection and keeping the name space.

Now, Pegasus manipulates the problem as follows. Suppose bob, a owner of real host, has a file "foo" in "/usr/bob/www/doc/~/", and suppose Pegasus is running in the name space of alice. Then Pegasus can see the file in "/doc/~/"(in name space of alice), so that Pegasus can respond directly to the request with URI path "/foo"(a request to real host) by referencing "/~alice/~/foo" (path "/~/foo" in name space of alice) keeping the connection and keeping the name space. Here, "foo" is explained as a file for simplicity, but any path is acceptable.

5. Some Notes in Writing Documents

5.1. Special letters in path name

It is wise not to use some special ASCII letters in path name[6]. They have special meaning in HTTP, HTML, or query. Some typical examples are listed below.

```
'%', ';', '?', '#', '"', '&', '<', '>', '=', '+', ' '
```

Current version of Pegasus does not support to use three letters ‘;’, ‘?’, ‘#’ in path name. Other letters are probably one of the main origin of troubles, and Pegasus have not been tested sufficiently in supporting these letters.

If they were used in path name and the path name is referenced in href value in HTML document, then they must be encoded using hexadecimal `%HH` style[7].

5.2. Href value for regular users

A regular user, say alice, might be prompted to write `href` value in action tag in her document like this:

```
<a href="/foo.html">...</a>
```

However according to HTML specification, the file “`/foo.html`” followed by “`href=`” does not mean the file in “`/user/alice/web/doc/foo.html`” but means the file in the document root of real host.

The correct expression is

```
<a href="/~alice/foo.html">...</a>
```

if the “`foo.html`” is in her document root.

5.3. URI syntax

Pegasus stands on the following URI syntax

```
http://host[:port][/path][;params][?query]
```

based on HTTP/1.1 defined in RFC2069. However the RFC was made obsolete by RFC2617 and the new syntax have been defined as

```
http://host[:port][/path][?query]
```

which is different from old syntax in “`;params`” part. In the RFC, semicolon is kept as reserved character in path field but the semantic is unclear.

Pegasus has been utilized the “params” for CGI script: the parameters in “params” are passed as arguments to the script. That still works in regular situation. However if the path is in the area that requires authentication, Mac/Safari fails to be authenticated.

Therefore it is not recommended to use “`;`”, though very handy in writing script.

---

[6] You will find more letters and the reason in RFC2396.

[7] A command tool named “`hfmt`” is included in the package of Pegasus 2.2. The tool converts path name to appropriate URL style.

5.4. CGI script

In case that "hctl" in "$web/etc/handler" is '1' or '+', you need not include "\r" code at the end of header lines. The server takes care for the "\r" in communicating with HTTP client. However for compatibility with old scripts, the "\r" codes are accepted.

In case that "hctl" is '+', Pegasus automatically attaches headers that are required in regular response to the request. Thus, it is enough for the script to write only special headers: if the header name is absent in regular HTTP headers, then the header is added; if the header name is already present in regular HTTP headers, then the header is replaced by new one in the script.

An example of output of this types of CGI is shown below.

```
Set-Cookie: cookie=something; expire=Sun, 6-Aug-2006 11:43:57 GMT;
domain=ar.aichi-u.ac.jp; path=/test4; secure

<html>
<head>
<title>Cookie sample</title>
</head>
<body>
...
</body>
</html>
```

where the second line is a wrapped line.

In case that "hctl" field is '+', HTTP header "Content-Length:" in the output of script will be ignored because server cares the header. On the contrary, if the "hctl" field is '1', all header must be provided by the script including "Content-Length:".

In writing scripts, you can use ramdisk. The disk is always provided at "/tmp" by the server, and output of script is once stored in the file named "/tmp/...", and then sent to the client as if it is a regular file.

You need not worry about if the files in "/tmp" are peeped by others, because Plan 9 ramdisk is private to a process and its children. And also you need not worry about if the files in "/tmp" are left somewhere when the script is accidentally terminated, because the ramdisk vanishes as soon as the execution of script is finished or terminated. These features make the use of temporal files safe and reliable.

5.5. CGI environment variables

Pegasus provides all environment variables described in CGI/1.1. See RFC3875 for details.

Subject to the specification, environment variable QUERY_STRING is set even if query is not present; then the value is null string. However Plan 9 shell rc does not recognize the $QUERY_STRING as a null string but recognizes as shell variable with no components. The substitution in rc script

```
QUERY_STRING=$"QUERY_STRING
```

will produce the value requested by the RFC. All the null string environment variables have same problem.

Scripts written in rc do not need `QUERY_STRING`. Pegasus decodes the query internally and builds variables suitable for rc. Environment variable names that are created from the query begin with "`QS_`".

Nameless data such as "`alice+bob&carol`" in the query produce shell variable `QS_` with the value

```
('alice bob' carol)
```

which enables an assignment

```
$*=$QS_
```

in rc script to produce arguments of rc.

Query data such as "`name=alice&name=bob`" will produce shell variable `QS_name` with the value

```
(alice bob)
```

which is again a one-dimensional array. Thus, no query data is lost.

6. Conclusion

Pegasus enables natural management of web documents: creator of web pages should be able to do every thing for the pages. This feature comes from name space virtualization technology of Plan 9, i.e., per process name space configuration. The technology enabled flexible design of Pegasus standing on safe, secure and reliable implementation.

Acknowledgment

I thank the anonymous author who wrote official httpd of Plan 9. Writing HTTP response headers is really painful. I could not have written up Pegasus if the source codes were not available to me. The codes manipulating responses and many other useful codes were imported to Pegasus with a little modification.

References

[1] R. Pike, D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom, “The use of name spaces in Plan 9”. *Operating System Review*, 27(2):72–76, Apr. 1993.

[2] Tom Duff, “Rc – The Plan 9 Shell”, *Plan 9 – The Documents – Volume Two*, Computing Science Research Center, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

[3] Lincoln D. Stein & John N. Stewart, “The World Wide Web Security FAQ”, `http://www.w3.org/Security/Faq/` February 2002

[4] RFC2068, “Hypertext Transfer Protocol – HTTP/1.1”, January 1997.

[5] RFC2069, “An Extension to HTTP : Digest Access Authentication”, January 1997.

[6] RFC2396, “Uniform Resource Identifiers (URI): Generic Syntax”, August 1998.

[7] RFC2616, “Hypertext Transfer Protocol – HTTP/1.1”, June 1999.

[8] RFC2617, “HTTP Authentication: Basic and Digest Access Authentication”, June 1999.

[9] RFC3490, “Internationalizing Domain Names in Applications (IDNA)”, March 2003.

[10] RFC3875, “The Common Gateway Interface (CGI) Version 1.1”, October 2004.

[11] RFC3986, “Uniform Resource Identifier (URI): Generic Syntax”, January 2005.

[12] RFC3987, “Internationalized Resource Identifiers (IRIs)”, January 2005.

[13] Lincoln D. Stein, “sbox: Put CGI Scripts in a Box”, `http://stein.cshl.org/software/sbox/`, December 2005.

[14] Kenji Arisawa, “Pegasus 2.2”, `http://plan9.aichi-u.ac.jp/pegasus/eman-2.2/`, August 2006.

[15] Kenji Arisawa, “Running https service”, `http://plan9.aichi-u.ac.jp/pegasus/eman-2.2/https.html`, August 2006.