A Rolling Gopher Gathers No Moss

Jerry Berkman
Programmer
217 Evans Hall, U. C. Berkeley
Berkeley, California 94720
jerry@uclink.berkeley.edu
Tel: (510)642-4804 Fax: (510)643-5385

Abstract

At U.C. Berkeley, we have recently started using Gopher for on-line documentation and publishing. This paper describes how we decided to use Gopher, first for our new free e-mail and information access service, and then for our campus information servers.

It then discusses the specific features in a Gopher server for student jobs listings, how to invoke Gopher in a restricted environment, how searches work with the extended WAIS code, how departments install their data in the e-mail system's Gopher, and other aspects of Gopher at U.C. Berkeley.

Paper

At U.C. Berkeley, as at many other Universities, we have found Gopher has made a significant improvement in our ability to publish information on-line. This paper describes a few Gopher projects in which I have been involved, and some notes on my experiences with Gopher.

Choosing Gopher for UCLink

In early 1993, U.C. Berkeley's Information Systems and Technology began offering free accounts for e-mail and internet access to UCB students and employees. We wanted this service, called UCLink, to be easier to use and more user friendly than our traditional UNIX services.

One of our first actions was to decide what program to use to supply on-line information on UCLink. We could port our home-grown help" system from our other UNIX systems or go with Gopher or some other information server.

One factor was that we wanted a way to provide on-line information on UCLink so that it would be accessible both to users who logged in and those who accessed their mail on UCLink via POP (Post Office Protocol) clients such as Eudora for the Mac and NUPop for DOS systems. Gopher provides that ability while our home-grown UNIX help system does not.

It is also difficult to get the documents in the home-grown system distributed to all the machines on which they may be useful. For example, I maintain a set of help files for FORTRAN users. Often people would send me questions, and I would want to reply "the details are in the help system". But I couldn't since they were on departmental systems or workstations and had no access to our help system. That is not a problem with Gopher.

For me, switching to Gopher served another purpose. It serves as an indicator that this is a new system and that we are addressing a new constituency. Our UNIX help system is oriented to people who know UNIX; for example it uses the notation ".." to mean parent directory and the documents assume the user know how to modify files with an editor. Many of our UCLink users do not know how to do this. Using Gopher and starting with a fresh approach was important.

Information Server Task Force

At about the same time as we were developing strategies for UCLink, a computing center task force was organized to study information servers. We had to decide whether to continue local efforts to develop a server called Infocal, or to switch to a public domain server such as WAIS, Gopher, or World Wide Web.

As part of our deliberations, we surveyed a group of comparable universities to see what they were doing. We found that 10 of the 15 universities surveyed were using or switching to Gopher for their campus wide information systems (CWIS). Several had local systems which were being converted to Gopher, and several expressed interest in WAIS or WWW, but no-one seemed to be running a CWIS based on WAIS or WWW.

It became clear that Gopher was the only system which had working client for a wide variety of platforms, had sufficient functionality, would be easy to support, and had an ongoing commitment from the Internet community.

Departmental Publishing via Gopher

Once we began the UCLink project, we discovered not only did people want to have access to e-mail and the Internet, but that many people wanted to publish information electronically for the campus. As Gopher is an easy way to do this, we decided that departments and recognized student groups could use UCLink to publish material via the UCLink Gopher.

However, I do not have a lot of time to devote to helping them. Many of them had already downloaded the Gopher software and documentation, but had trouble figuring out what to do.

I developed a short write-up which tells people exactly what steps to take to run Gopher with their own data on UCLink to develop their menus and documents. Once they get a little experience and have information ready to be published, they then contact me and we copy it into the main UCLink Gopher system. This approach has proven quite popular. Currently a wide variety of departments and units, such as the residence halls, telecommunications, research units, and academic departments, are working on developing information to be published via Gopher.

The Student Jobs Gopher

I recently helped set up student job listings in Gopher. This has two unique features compared to my other Gopher work.

First, since the listings include both on-campus and off-campus jobs, we want the listings to be restricted to UC Berkeley students. We do not want high school students, non-students, or Stanford students reading the listings and applying for the jobs.

Gopher is not yet able to restrict access to a particular set of people, but it is easy to restrict access to the Berkeley Domain, "berkeley.edu", via the Gopher configuration file. This means it is not restricted to students, but that faculty and staff can also read the listings; this is acceptable to the information providers.

The second special feature is that we want to allow the easiest possible access to these listings. To provide this, we have set up a special account named "jobs" on UCLink for simplified access.

Any student or employee can connect to UCLink and use the "jobs" id to login to UCLink. Then the user is asked to supply his or her last name, identification number, and birthday (month and day). After verifying the information, the program invokes Gopher to look at the jobs listings.

Of course any student can get an account on UCLink or access Gopher from other campus systems, so a special access method for the jobs Gopher is not strictly necessary.

However some may not want to open an account just to look at the job listings. They may have an account on another system which is temporarily unavailable, or they may be at home and find it easier to connect directly to UCLink rather than go through another system.

On our part, we don't want to encourage people to open accounts they will not be using regularly because then the people often forget their password, leading to extra work for us when they decide to use the account in the future. It will be interesting to see how much this special access method is used. I'll have a better feel for this by SIGUCCS.

Since we are providing this access specifically for the student jobs listings, I tried to make the access secure. If the user wants more general access to UCLink, he or she can open an account on UCLink.

The UNIX Gopher client normally lets you save files, print files, and select alternate versions of "telnet" and "lpr". The "-s" (secure) option on the UNIX client eliminates these capabilities. In addition, it restricts mail addresses to normal looking addresses, and excludes addresses which have odd syntax which could have obscure purposes.

In normal use of Gopher, you may invoke "telnet" and the "more" pager utilities. These both allow escape to the user's shell. The Gopher software distribution directory has versions of "telnet" and "more" that do not allow shell escapes (See "Gophertools" under "UNIX" under "Gopher Software Distribution").

The secure "telnet" supplied by the Minnesota seems ok, but the "more" seems very simple minded. I decided instead to take the Ultrix source for "more" and make a few minor changes: I commented out the code for the "!" and ":!" shell escapes, I commented out the "v" editor escape, and I supplied a different "help" file. I invoke the Gopher client at the end of the "jobs" shell program,

```
setenv( "PAGER",
"/usr/local/gopher/secure/more", 1 );
setenv( "GOPHER_TELNET",
"/usr/local/gopher/secure/telnetsecure", 1 );
execl( "/usr/local/gopher/bin/gopher",
"/usr/local/gopher/bin/gopher",
"-s",
"uclink.berkeley.edu",
"1601",
(char *) 0 );
```

Since the "jobs" account is set up with this program as its logon shell, exiting Gopher results in logging off. The program logs the identity of each person using it; if they do manage to escape from the program and try to do something out of line, we will have a record of their identity.

Searches in Gopher

The UNIX Gopher Server uses the code from the WAIS (Wide Area Information Server) system for searches. The official WAIS search code comes from Thinking Machines, Inc. It is also possible to use the WAIS code as modified by Don Gilbert, Indiana University. It is available via anonymous ftp from ftp.bio.indiana.edu .

The original WAIS code only indexes words; the modified code adds boolean operators "and" and "not", partial word matches indicated by "*" at the end of a word, literal phrases enclosed in quotes, and other features.

I installed the modified code in our Gopher server. Although there is a short "readme" file, it is not all that clear how the searches work. The following description is based on experiments.

For the experiments, I used a small data base of job listings. If you specify a single word in a search, e.g.:

```
Search pattern: berkeley
```

the files which contain that word are presented in a new Gopher menu:

```
Search Jobs Listings: berkeley

 1. job11
 2. job1
 3. job10
 4. job4
 5. job8
```

The WAIS system returns a weighted list of files which contain the search word. In this case, "job11" contains several instances of "berkeley" so it is listed first; the other entries each contain only one instance of the word "berkeley", so they are listed in alphabetical order. The search ignores case.

If you search for a simple word and then display one of the selected files, the word is highlighted where it occurs in the file.

If you list several words in the search field, the search looks for files which contain any of the specified words. Thus:

```
Search pattern: programmer berkeley
```

will look for all files which contain programmer or berkeley. These will be presented in weighted order. In my experiment, a file which contained both berkeley and programmer once was listed before a file containing berkeley three times, which in turn was listed before files which contained berkeley only once.

The implied or can cause trouble. For example:

```
Search pattern: San Francisco
```

This is supposed to be a search for jobs in San Francisco. However it is really a search for files containing either San or Francisco and could include jobs in San Jose. You could try using the boolean "and":

```
Search pattern: San and Francisco
```

Even so, this would include a listing for a clerk at San Leandro Graphics at 110 Francisco Street in San Leandro and this listing could easily come out as the top weighted listing since it contains "San" twice and "Francisco" once.

To get the desired search, use the literal form (either single or double quotes may be used):

```
Search pattern: "San Francisco"
```

This excludes the listing for San Leandro Graphics. However, when you display the files found, there is no highlighting. Highlighting is used only when searching for words, not for literals. In small files, such as the job listings, this is not very important. In longer files, this is a severe disadvantage.

You can exclude files using the "not" operator:

```
Search pattern: programmer not
  berkeley
```

could be an attempt to look for a programmer job outside of berkeley. However it could also exclude a listing containing something innocuous like "Berkeley students welcome" or "Berkeley course credit possible".

The order in which the operators are applied is not the same as the order in which they are written. All "not" words are saved and applied at the end to the list of files which contain the sought after words.

The other words are applied starting from the left. To quote from the "iubio-wais.readme" file:

"For example, this query

```
red and blue or yellow but not green
  and orange or black but not white
```

will be interpreted like this (the parentheses just show the implicit left-to-right interpretation):

```
((((((red and blue) or yellow) and
  orange) or black) not green) not
  white)"
```

As mentioned earlier, you can search using the asterisk as a wild card, e.g.:

```
Search pattern: comput*
```

will match files with "computer", "computes", "computing", "computation", etc. Again, there will not be any highlighting.

Don Gilbert's patches to WAIS give us a much more powerful tool. However, I would use them with some caution. Sometimes the result is not what I would expect. For example, I would expect the following two commands to be equivalent:

But they are not. The first works as expected and found several jobs; the second did not find anything.

The "iubio-wais.nes" file states that booleans and literals can "generally be mixed in a query". But

```
Search pattern: analyst not 'san
  francisco'
```

excludes a job in San Carlos.

Gopher Problems

We are using Gopher on an Ultrix system. The "vi" editor uses the termcap facility, but Gopher uses the terminfo facility. This means someone can logon, use the editor, but not be able to use Gopher:

```
% gopher
Sorry, I don't know how to deal with
  your 'xterm48' terminal.
Segmentation fault (core dumped)
```

Not only did Gopher not work, it aborted with a core dump! This leads to fairly vitriolic mail to our consult account. Hopefully, someone will fix this soon. The minimal fix would be to exit politely with a comment but without the segmentation fault and core dump. A better fix would be to prompt the user for a terminfo compatible terminal name or to convert on the fly the termcap entry to terminfo.

Believe it or not, if you get the message:

```
Sorry, this isn't a WAIS index...
```

on an attempt to use a WAIS index, your index may be fine. The problem is probably that you forgot to compile or link in the WAIS code.

Menu Titles and Ordering

Gopher has a facility to title and order menu items via a parallel set of files in a directory named ".cap". For example, if you have a file named "ucbinfo", you can give it a title and a position in the menu by creating a file ".cap/ucbinfo" and putting in it lines such as:

```
Name=UCB Campus Information
Numb=1
```

This specifies the title and location of the entry in the Gopher menu. I found this feature to be inconvenient to use. I could not see what the menu was going to look like, and I had to edit several files for minor changes in ordering of the menu. Gopher also seems to be very particular about the placement of the "Numb=" line.

I create a file called ".order" which contains both the file names and titles in the order I want them,

```
e.g.:
FAQs          FAQs: Frequently Asked Questions about UCLink
email         Information on E-mail ( Pine, forwarding,...)
Netnews       Information about Net News
accounts      Accounts: Eligibility, Passwords, ...
help          Where to Go for Help and More Information
internet      About the Internet: Goher, FTP, News, ...
```

FAQs F AQs: Frequently Asked Questions about UCLink email Information on E-mail (Pine, forwarding, ...) Netnews Information about Net News accounts Accounts: Eligibility, Passwords, ... help Where to Go for Help and More Information internet About the Internet: Goher, FTP, News, ...

The first token is the name of the file; the rest of the line is the title to appear in the menu. The following shell script creates the ".cap" directory and files. I find this approach must easier to use than creating ".cap" files directly.

```
#!/bin/csh -f
#
# if .order file exists:
# Convert .order into .cap files
# currently ignores any .Links file
# removes old .cap directory
# if no .order, do nothing
#
if( ! -e .order ) exit
/bin/rm -fr .cap
mkdir .cap
set awkf = `mktemp`
cat << "EOT" >! $awkf
BEGIN { skip[" "] = "yes"; skip["\t"] = "yes"; }
{
 n = length( $1 ) + 2;
 while( skip[substr( $0, n, 1 )] == "yes" ) n = n + 1;
 print "cat << XXYY > .cap/" $1 ;
 print "Name=" substr( $0, n );
 print "Numb=" NR;
 print "XXYY";
}
"EOT"
awk -f $awkf .order | csh
/bin/rm -f $awkf .cache
```

Our Gopher Configuration

Since many departments are preparing information for our Gopher, and some of them may install shell scripts or perl programs, we have decided to run our Gopher server on UCLink not as root, but as a special user who has no special privilege. Since making that conversion, I have not been able to get chroot to run with the WAIS server and other options. However, that does not seem important with as the user id has no special permissions.

Our Gopher servers are run out of "inetd.conf" with a command of the form:

```
gopherd -I -c -u gopherm -o
  .../config -l .../log ...
```

However, commands in "inetd.conf" can only have 4 arguments, so I wrote a short C program to "exec" "gopherd".

Acknowledgements

In concluding, I would like to thank Tamara Sturak for introducing me to Gopher and Ed Moy and Darek Milewski for helping me with Gopher projects. Without their help, those Gopher servers would still be in the debugging stage.