FreePOPs Manual

16th April 2004

Contents

1 Introduction

FreePOPs is a POP3 daemon plus a LUA interpreter and some extra libraries for HTTP and HTML parsing. Its main purpose is translating local POP3 requests to remote HTTP actions on the supported web-mails, but it is really more flexible. For example there is a plugin to read news from a website as if they were mails in a mailbox. You can easily extend FreePOPs on the fly, without even restarting it. You can add a plugin or modify an existing one simply changing the script file since the plugins are written in LUA and are interpreted on the fly.

1.1 Usage situations

FreePOPs can be useful in some situations, here we give the most obvious ones:

- You are behind a firewall that closes the 110 port but you need to read your mail and the web-mail of your mail provider sucks.
- Your mail provider does not allow you to access your mailbox with POP3 protocol, but only through the web-mail service.
- You prefer looking at your mailbox instead of browsing some websites news.
- You have to develop a pop3 server in less than a week and you want it to be reasonably fast and not so memory consuming.
- You are not a C hacker, but you want to benefit from a fast POP3 server frontend written in C and you want not to loose a month in writing the backend in C. LUA is a really simple and tiny language, one week is enough to learn it in a way that allows you to use it productively.

1.2 Features

FreePOPs is the only software I know with these features:

- POP3 server RFC compliant (not full featured but compliant).
- Portable (written in C and LUA that is written in C, so everything is written in the most portable language around the world).
- Small (in the sense of resources usage) and reasonably fast.
- Extremely extensible on the fly using a simple and powerful language.
- Pretty documented.
- Released under the GNU/GPL license (this means FreePOPs is Free Software).

2 History

FreePOPs was not born from scratch. A similar project (only in the main usage situation) is LiberoPOPs.

The ancestor of FreePOPs is completely written in C for some uninteresting reasons. LiberoPOPs supports "plugins" but in a more static and complex way. The POP3 server frontend could be attached to a backend written in C. This means you have to recompile and restart LiberoPOPs each time to change a line in a plugin. Another interesting point is that LiberPOPs was created from scratch in a really short time (you have to be Italian and use a `@libero.it` mail address to understand why), this means it was born with a lot of bugs and FIX-ME in the code.

The LiberoPOPs project had a quick success, because everybody needed it. This means we had a lot of users. In the opensource (and also Linux) philosophy you have to release frequently and this was exactly what we did. We used to release every 2 days. We were working not with Unix users, nor hackers, but mostly with Win32 users. Suddenly we realized that they were lazy/bored of updating the software every 2 days. The ugly Win-world has taught them that software auto-updates, auto-install and even auto-codes probably.

We tried to solve this pulling out of the C engine most of the change-prone code, but this was really hard since C is not thought to do this. After LiberoPOPs had stabilized we started to think how to solve this.

A scripting/interpreted embedded language seemed to me a nice choice and after a long search on the net and on the newsgroup of my university I found LUA.. This is not the place for telling the world how good this small language is and I won't talk more about it here. Integrating the LUA interpreter in LiberoPOPs was not so hard and FreePOPs is the result. Now it is really easier to write/test a plugin and (even if it is not implemented yet) an auto-update facility is really easy to code since there is no need to recompile the C core in most cases.

3 FreePOPs configuration file

FreePOPs doesn't really need a configuration. Most users shouldn't change the configuration file. In case you are a developer or a really curious user the configuration file is `config.lua`, placed in the program directory under win32 or in `/etc/freepops/` in a posix environment.

4 FreePOPs command line parameters

The real FreePOPs configuration is made trough command line arguments. They are described in depth in the man page in Unix environments, here we present only the most useful:

-p <port>, –port <port> By default FreePOPs binds on port 2000. To alter this behaviour just use this switch.

-P <host>:<port>, –proxy <host>:<port> To tell FreePOPs which is your HTTP proxy.

-v, –verbose, -w, –veryverbose This tells FreePOPs to log some interesting info for bug reporting.

-t <num>, –threads <num> FreePOPs is able to manage multiple connections, up to num. Default is 5.

In posix environment like Debian GNU/Linux you can start FreePOPs at boot time as a standard service. In this case the command line switches are stored in `/etc/default/freepops`, in some rpm based systems you should find the same file as `/etc/sysconfig/freepops`.

5 Email client configuration

To configure your email client you must change the pop3 server settings. Usually you must use localhost as the pop3 host name, and 2000 as the pop3 port. In case you install FreePOPs in another computer of your LAN, you should use the host's name instead of localhost. In case you changed the default port with the `-p` switch you should now what you are doing.

6 Creating a plugin

The best way of doing this is to read carefully the `libero.lua` file, that is a quite simple but really commented example of web-mail plugin. Then you should copy the `skeleton.lua` file and start hacking on it. Remember to edit `config.lua` to make FreePOPs associate the right mail-addresses domains to your new plugin. A simpler example is `flatnuke.lua` that is a web-news plugin.

6.1 The interface between the C core and a plugin

As we explained before the C POP3 frontend has to be attached to a LUA backend. The interface is really simple if you know the POP3 protocol. Here we only summarize the meaning, but the RFC 1939 (included in the `doc/` directory of the source distribution) is really short and easy to read. As your intuition should suggest the POP3 client may ask the pop3 server to know something about the mail that is in the mailbox and eventually retrieve/delete a message. And this is exactly what it does.

The backend must implement all the POP3 commands (like USER, PASS, RETR, DELE, QUIT, LIST, ...) and must give back to the frontend the result. Let us give a simple example of a POP3 section taken from the RFC:

```
 1  S: <wait for connection on TCP port 110>
 2  C: <open connection>
 3  S:    +OK POP3 server
 4  C:    USER linux@kernel.org
 5  S:    +OK now insert the pasword
 6  C:    PASS gpl
 7  S:    +OK linux’s maildrop has 2 messages (320 octets)
 8  C:    STAT
 9  S:    +OK 1 320
10  C:    LIST
11  S:    +OK 2 messages (320 octets)
12  S:    1 320
13  S:    .
14  C:    RETR 1
15  S:    +OK 120 octets
16  S:    <the POP3 server sends message 1>
17  S:    .
18  C:    DELE 1
19  S:    +OK message 1 deleted
20  C:    QUIT
21  S:    +OK dewey POP3 server signing off (maildrop empty)
22  C:  <close connection>
23  S:  <wait for next connection>
```

In this session the backend will be called for lines 4, 6, 8, 10, 14, 18, 20 (all the `C:` lines) and respectively the functions implementing the POP3 commands will be called this way

```
user(p,"linux@kernel.org")
pass(p,"gpl")
stat(p)
list_all(p)
```

```
retr(p,1)
dele(p,1)
quit_update(p)
```

Later I will make clear what p is. I hope we'll remove it making it implicit for complete transparency. It is easy to understand that there is a 1-1 mapping between POP3 commands and plugin function calls. You can view a plugin as the implementation of the POP3 interface.

6.2 The interface between a plugin and the C core

Let us take in exam the call to `pass(p,"linux@kernel.org")`. Here the plugin should authenticate the user (if there is a sort of authentication) and inform the C core of the result. To achieve this each plugin function must return an error flag, to be more accurate one of these errors:

Code	Meaning
`POPSERVER_ERR_OK`	No error
`POPSERVER_ERR_NETWORK`	An error concerning the network
`POPSERVER_ERR_AUTH`	Authorization failed
`POPSERVER_ERR_INTERNAL`	Internal error, please report the bug
`POPSERVER_ERR_NOMSG`	The message number is out of range
`POPSERVER_ERR_LOCKED`	Mailbox is locked by another session
`POPSERVER_ERR_EOF`	End of transmission, used in the popserver_callback
`POPSERVER_ERR_TOOFAST`	You are not allowed to reconnect to the server now, wait a bit and retry
`POPSERVER_ERR_UNKNOWN`	No idea of what error I've encountered

In our case the most appropriate error codes are `POPSERVER_ERR_AUTH` and `POPSERVER_ERR_OK`. This is a simple case, in which an error code is enough. Now we analyze the more complex case of the call to `list_all(p)`. Here we have to return an error code as before, but we have also to inform the C core of the size of all messages in the mailbox. Here we need the p parameter passed to each plugin function (note that parameter may became implicit in the future). `p` stands for the data structure that the C expect us to fill calling appropriate functions like `set_mailmessage_size(p,num,size)` where num is the message number and size is the size in bytes. Usually it is really common to put more function all together. For example wen you get the message list page in a webmail you know the number of the messages, their size and uidl so you can fill the p data structure with all the informations for LIST, STAT, UIDL.

The last case that we examine is `retr(p,num,data)`. Since a mail message can be really big, there is no pretty way of downloading the entire message without making the mail client complain about the server death. The solution is to use a callback. Whenever the plugin has some data to send to the client he should call the `popserver_callback(buffer,data)`. `data` is an opaque structure the popserver needs to accomplish its work (note that this parameter may be removed for simplicity). In some cases, for example if you know the message is small or you are working on a fast network, you can fetch the whole message and send it, but remember that this is more memory consuming.

7 Submitting a bug

When you have problems, you think you found a bug, please follow strictly this *iter*:

1. Update to the most recent version of FreePOPs.
2. Try to reproduce the bug, if the bug is not easily reproducible we are out of luck. Something can still be tried: if the software crashed you could compile it from the sources, install valgrind, run freepopsd with valgrind and hope the error messages are interesting.
3. Clean the log files
4. Start FreePOPs with the -w switch
5. Reproduce the bug
6. Send to the developers the log, plus any other info like your system type and how to reproduce this bug.

8 Authors

This manual has been written by Enico Tassi <gareuselesinge@users.sourceforge.net> and revised by Nicola Cocchiaro <ncocchiaro@users.sourceforge.net>

8.1 Developers

FreePOPs is developed by:

- Enico Tassi <gareuselesinge@users.sourceforge.net>
- Alessio Caprari <alessiofender@users.sourceforge.net>
- Nicola Cocchiaro <ncocchiaro@users.sourceforge.net>
- Simone Vellei <simone_vellei@users.sourceforge.net>

LiberoPOPs id developed by:

- Enico Tassi <gareuselesinge@users.sourceforge.net>
- Alessio Caprari <alessiofender@users.sourceforge.net>
- Nicola Cocchiaro <ncocchiaro@users.sourceforge.net>
- Simone Vellei <simone_vellei@users.sourceforge.net>
- Giacomo Tenaglia <sonicsmith@users.sourceforge.net>

9 Thanks

Special thanks goes to the users who tested the software, to the hackers who made it possible to have a free and reliable development environment as the Debian GNU/Linux system.