Mothballing Notes

P92000.PinkFloyd User Space OO Filesystem Server

John W. Fawcett ([jfawcett@microsoft.com](mailto:jfawcett@microsoft.com))

12/20/2016

This work is an offshoot of the work done by Charles Forsyth and captured at <https://github.com/zchee/u9fs>. That work itself was an offshoot of work done by Bob Flandrena, Ken Thompson, Rob Pike, and Russ Cox at Lucent. That work is copyright 1992-2002, but the license is all-use.

The original code is a user-space, network-based server for the P9 filesystem that works in user space and can serve any disk that has a POSIX-compliant programming interface. The code was originally written to provide access to UNIX disks that supported POSIX and worked through inted. The user would compile the program, set up inetd.conf/xinetd.conf entries as appropriate, then mount the disk with a few “-u “ options.

I needed a server that could do this, but would work with a Windows-based filesystem, might use a different transport, and might have different authorization methods. Further, since my work was experimental in nature, I wanted the ability to easily switch between different transports and/or underlying filesystems easily and quickly. Finally, I wanted something that didn’t run under inetd, since that made working with the debugger difficult.

This code was uniquely suited to these tasks. It was written well enough that conversion to objects was possible and, with that, abstraction and polymorphism. My plan was to abstract the transport, authorization, filesystem access, and any internal objects that presented themselves, implement pure virtual interfaces for those I wished to morph, then get it all working again. The first step would be Linux <-> Linux, with Windows <-> Linux next.

I recently had an epiphany with respect to the work this project was intended to inform and/or prototype, and so will be abandoning this for the time being. Before I do, this is where the code sits:

1. Abstraction
   1. Largely complete. The primary abstracted objects are ITransport, IFileSystemUserOps, and IAuth
   2. Work on the Windows port indicates the need for IFileSystemSupportOps, but that may not be necessary
   3. The Windows work in very much in progress. I will probably check that in as a subdirectory to this repo
2. Concrete classes
   1. Auth
      1. None
      2. P9Any (untested)
      3. Rhosts (untested)
   2. Transport
      1. StdinTransport – for inetd (untested)
      2. TCPTransport – multithreaded TCP system
   3. IFileSystemUserOps
      1. PosixUserOps – Access to a POSIX-compliant filesystem
3. P92000 Standards Compliance
   1. Original code was compliant to P92000
   2. I implemented \*most\* of the P92000.u extensions (mostly for symlink)
   3. I know that create with auth None will create a file owned by root instead of the attached user. This is kind of a design feature, since the user on the remote system will probably not have credentials on the system with the server. What I really need to do is create the file as root, then chown it to the UID of the attached user, even if they don’t exist on the local system. The problem with this is that the GID of the owner is not presented in the protocol, so we’d have to capture that from the Attach message and persist it. Once that’s done, permissions should work properly.
   4. My plan was that once I had the code working properly with P92000.u, I would extend that further for some additional functionality I was planning. This extension would be tentatively named P92000.PinkFloyd, because I’m more Dark Side Of The Moon than anything U2 ever did. The extension would have been fairly straightforward and captured in abstraction through the ITransport layer.
4. Porting Activity
   1. I’ve made this work Linux-to-Linux, and Linux-to-Cygwin
   2. Was working on a Windows port when mothballed
   3. Windows port would rely on the TCPTransport object, but would require a new IFileSystemUserOps implementation

Of special note in this code is the TCPTransport object. This code is implemented as a set of three threads synchronized using semaphores and protected by thread recursive mutexes. The threads are:

1. Acceptor: Binds to the port and waits for connections to hit. When one hits, the new socket is added to a queue and the Validation thread is called
2. Validation: Accepts sockets from the Acceptor and does the Auth. If the user is acceptable, their socket is added to the Listener
3. Listener: The main thread loop for this transport. Waits for new data on any connected socket, except the one bound to the acceptor port

What needs to happen to this code:

1. Finish the P92000.u extensions
2. Finish the Windows port
3. Write a buttload of tests

John W. Fawcett

[jfawcett@microsoft.com](mailto:jfawcett@microsoft.com)

December, 2016