Design and Implementation Considerations

for

A Pure Tcl License Manager

By

Gerald W. Lester

HMS Software, Inc.

Abstract

This paper discusses design and implementation considerations for a pure Tcl License Manager that is platform neutral for the management of current use licenses. It covers a minimum effort implantation that is layered on top of the TclHttpd to provide administrative control.

Also discussed are certain limitation imposed by the current Tcl implementation of server sockets.

Goals

Monitoring vs Enforcement

Out main goal was to monitor the usages of concurrent license, not to enforce license limits. Most of our clients are concerned about accidentally exceeding the number of licenses. They requested help from us on knowing when they needed to purchase additional licenses. But because of the criticality of the application to their business, neither us nor they wanted to prevent the application from running.

WWW Control Interface

To easy installation, we decided that we wanted all controls, configuration of limits and display of usage to be via a web browser without use of a plugin, Java or Javascript. This would simplify "selling" the solution to the network and security administrators at our client sites.

Why Tcl

Our current application, which involves Computer Aided Process Planning and Manufacturing Execution. The application is almost all pure Tcl/Tk – excpetions being some extension such as oratcl. Thus it was natural to chose Tcl to implement the License Manager. Additionally the implementer has extensive (10+ years) experience with Tcl/Tk.

Of course the platform independence that Tcl provides when a Tcl/Tk application is implemented correctly, along with the fact that a large part of the "work" was already available in the public domain assisted in making this an easy decision. In particular we, the following features made Tcl very attractive to implement the License Manager in:

1. Excellent socket implementation
2. TclHttpd for the server side
3. Tcllib's SMTP and MIME for email notifications

Client Technical Considerations

Since our application is mission critical for a large number of our customers, we did not want the application to not start, or later cease to function, if either the License Manager was not available at startup, the connection to the License Manager was lost or it the number of licenses was exceeded – yet at any given time we wanted as accurate as possible a view of the current license usage. This drove us to the use of non-blocking (asynchronous connect and exclusive use of non-blocking reads) TCP/IP connections. This had the added benefit that the client was notified, via an EOF on the socket channel, if connection was lost to the server and thus could initiate a reconnect. It also provided notification to the server, again via an EOF on the socket channel, if the client was unexpectedly terminated (such as the machine crashing).

Server Technical Considerations

Since it was our desire for the client interface to be via simple web pages, the use of the TclHttpd web server (written in pure Tcl) was a natural choice. We added code to handle the incoming socket connections for license usage and to track what "alarm limit" the system was in at any given point. This was done via the use of the "library" directory that the TclHttpd sources in at startup. The web pages were coded using the Tcl Markup Language (.tml) files.

Server Human Interface

The following features were desired:

1. Ability to define an arbitrary number of alarm limits
2. Ability to define which color to display when the system is within the bounds of an alarm limit
3. Ability to define deadbanding, both time and numerical, to prevent the system oscillating between two alarm limits
4. Ability to define email notifications associated with entry and exit of alarm limits
5. Ability to display current and historic usage
6. Ability to "drill down" and see exactly which people are using how many licenses.

Enhancements needed for License Enforcement

For license monitoring one is going in with the assumption of a non-hostile environment, i.e. the customer is concerned about accidentally exceeding the number of licenses; however when dealing with license enforcement one has to assume a hostile environment, i.e. the users may be actively attempting to circumvent the licensing policy. This causes several changes in both the design and deployment.

One change is that compiled code, preferably delivered as a executable, should be used. While some argue the "security through obscurity" is not valid security, it does add a level of defense. IMHO. good system for a hostile environment should take a multi-layer security approach.

The next change would be to use TLS or another hardened pipe to pass information. This layer makes it harder, although not impossible, for the user to implement their own server or a man in the middle attack that always grants a license.

To make it harder on the potential hostile user, a more complex handshake could be implemented that uses a shared secret, embedded time and multiple request for the license – with at least several (a random number between 1 and N) requests that should fail and at most one that should succeed. If a success is returned for a request that should fail, the application would assume that a invalid server or a man in the middle attack is in progress.

Possible Future Enhancements

One enhancement for our License Monitor that is strongly being considered is the use of an initial UDP handshake for the client to automatically discover the server(s). As well as for the server(s) to discover each other. This would eliminate the requirement for the client application to have a configuration parameter for the address of the server. It also opens the possibility to implement redundant and/or distributed servers.

redundant servers would have all clients go the a primary server then to a backup server if the primary fails. Distributed servers would allow a client to go to any server to obtain a license. The difference is mostly in how the servers have to distribute their database, for redundancy the master has to distribute to the saves, for a distributed server system all servers have to distribute to all other servers. Using the TIE package in Tcllib, makes the coding of a distributed server system not much more work than the coding of a redundant server system.

Technical Shortcoming

While Tcl has been almost ideal with its simple abstraction of a socket interface, one shortcoming has become evident, namely the current (Tcl 8.4) implementation does not handle a node having its IP address changed while a listen socket (i.e. server socket) is open. The socket becomes dead, in other words it will accept no new connections, and the application will not receive any notification that something has gone horribly wrong.

This is a know issue that has been periodically discussed on the comp.lang.tcl newsgroup.

Summary of Results

The project was implemented in a little over two days time and has been deployed at multiple customer sites. Below is a summary of the amount of code required to implement the system:

Client

- 252 lines (including comments) of Tcl Code

Server

Tcl Code:

Basic Server: 415 lines (including comments)

Alarm Management: 682 lines (including comments)

Display Support: 211 lines (including comments)

TML pages:

Eight files totaling 532 lines (including comments)

Total Code: 2092 lines (including comments)