Design and Implementation Considerations for a Pure Tcl License Manager

By

Gerald W. Lester

HMS Software, Inc.

Overview

- Goals
  - Monitoring vs Enforcement
  - WWW Control Interface
  - Email Notifications
- Why Tcl
- Client Technical Considerations
- Server Technical Considerations
- Server Human Interface
- Summary of Results

Goals

- Monitoring vs Enforcement
  - Monitoring
    - Know license usage
    - Notify administration as usage approaches limits
  - Enforcement – a anti-goal
    - Stop new clients from running when limits exceeded
- WWW Control Interface
  - All server site
  - Set limits, colors and actions
  - Display usage
- Email Notifications
  - Send email to administrators on usage

Why Tcl

- Current product 99% pure Tcl/Tk
  - Computer Aided Process Planning and Manufacturing Execution System
- Person implementing very familiar with Tcl/Tk
- Portable
  - Correct implementation will run on all platforms
- Most of the “work” already done
  - Excellent socket implementation
  - TclHttpd for the server side
  - Tcllib's SMTP and MIME for email notifications

Client Technical Considerations

- Must be non-blocking
- Must not stop operation of client
- Must reconnect if connection lost with server
- Server must know if client goes away

Client Technical Details

```
proc InitiateConnection {} {
    variable managerSocket
    variable managerHost
    variable managerPort
    variable managerConnected

    set managerSocket [socket -async $managerHost $managerPort]
    fileevent $managerSocket writable AcceptConnection
    fileevent $managerSocket readable [list ConnectionError $managerSocket]
    set managerConnected 0
    return
}
```

Client Technical Details

```
proc AcceptConnection {} {
    variable managerSocket
    variable managerConnected

    set message [list HELLO [file tail [info nameofexecutable]]]
    fconfigure $managerSocket -translation binary -blocking 0
    puts -nonewline $managerSocket \
        [format {%10.10d%s} [string bytelength $message] $message]
    flush $managerSocket
    set managerConnected 1
    catch {fileevent $managerSocket writable {}}
    return
}
```

Client Technical Details

```
proc ConnectionError {socket} {
    variable managerSocket

    catch {fileevent $socket writable {}}
    catch {fileevent $socket readable {}}
    catch {close $socket}
    if {[string equal $managerSocket $socket]} {
        InitiateConnection
    }
    return
}
```

Server Technical Considerations

- Embedded in TclHttpd
- Uses Tcl Markup Language (.tml) files for User interface
- Uses Tcllib's STMP and MIME packages

Server Technical Details

```
trace variable ::hmsLicenseMonitor::totalConnectionCount w CheckAlarms

set listeningSocket \
	[socket -server AcceptConnection $portNumber]
```

Server Technical Details

```
proc :AcceptConnection {socket host port} {
    variable socketDataArray

    fconfigure $socket -translation binary -blocking 0
    set socketDataArray($socket,state) header
    set socketDataArray($socket,request) {}
    set socketDataArray($socket,bytesLeft) 10
    set socketDataArray($socket,executable) {}
    set socketDataArray($socket,host) $host
    fileevent $socket readable [list GetIncomingRequest $socket]
    return
}
```

Server Technical Details

```
proc GetIncomingRequest {socket} {
    variable socketDataArray

    if {[llength [file channel $socket]] && ![eof $socket]} {
        ##
        ## We have a valid open channel, so process the request.
        ##
        ## Ignore any I/O error as they will get processed at the end.
        ##
        catch {
            set str [read $socket $socketDataArray($socket,bytesLeft)]
            append socketDataArray($socket,request) $str


            incr socketDataArray($socket,bytesLeft) -[string length $str]
```

Server Technical Details

```
if {$socketDataArray($socket,bytesLeft) == 0} {
    switch $socketDataArray($socket,state) {
        header {
            scan $socketDataArray($socket,request) {%d} socketDataArray($socket,bytesLeft)
            set socketDataArray($socket,state) body
        }
```

Server Technical Details

```
body {
    set reply {}
    set command [lindex $socketDataArray($socket,request) 0]
    set data [lindex $socketDataArray($socket,request) 1]
    switch -exact $command {
        HELLO {
            Connect $socket $data
        }
        DISCONNECT {
            Disconnect $socket
        }
    }

    set socketDataArray($socket,state) header
    set socketDataArray($socket,bytesLeft) 10
}
```

Server Technical Details

```
if {[llength [file channel $socket]] && [eof $socket]} {
    Disconnect $socket
}
```

Server Technical Details

```
proc Connect {socket prgm} {
    variable socketDataArray
    variable connectionByExecutableArray
    variable totalConnectionCount
    variable hostDataArray

    set socketDataArray($socket,executable) $prgm
    if {[string length $prgm]} {
        if {![info exists connectionByExecutableArray($prgm)]} {
            set connectionByExecutableArray($prgm) 0
        }
        set host $socketDataArray($socket,host)
        set hostDataArray($host,socket,$socket) $socket
        incr connectionByExecutableArray($prgm) 1
        incr totalConnectionCount 1
    }
    return
}
```

Server Technical Details

```
proc Disconnect {socket} {
    variable socketDataArray
    variable connectionByExecutableArray
    variable totalConnectionCount
    variable hostDataArray

    catch {fileevent $socket readable {}}
    catch {close $socket}
    set prgm $socketDataArray($socket,executable)
    if {[string length $prgm] && [info exists connectionByExecutableArray($prgm)]} {
        incr connectionByExecutableArray($prgm) -1
    }
    set host $socketDataArray($socket,host)
    if {[llength [array names hocstDataArray $host,socket,*]] != 2} {
        incr totalConnectionCount -1
    }
    unset hostDataArray($host,socket,$socket)
    array unset socketDataArray $socket,*
    return
}
```

Server Human Interface

- Implements multiple alarm levels
  - With deadbands
    - Time based
    - Number based
  - Optional email notification upon entry/exit
- Displays summary of license usage
  - By application type
  - By user
  - Color coded by limit
- Allows drill down to particular user/application's usage

Enhancements needed for License Enforcement

- Compiled code
- Use of TLS
- Enhanced handshake
  - Use of pass/fail with at least one failure
  - Use of key (shared secret)

Possible Future Enhancements

- Use of UDP extension by client to discover license monitor server(s)

Technical Shortcoming

- Changing IP
  - Closes server listen sockets
    - No notification available

Summary of Results

- Client
  - 252 lines (including comments)
- 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)

Questions

?