Using FreeBSD for a Console Server

By Branson Matheson

Branson.Matheson@ferginc.com

Ferguson Enterprises, Inc.

Abstract:

Managing serial consoles for multiple servers has always been a challenging problem for system administrators. Finding the space and right layout for the multitudes of terminals is difficult. There is no remote connectivity or security besides the access in the computer room. Any important events that are logged to the console are lost or difficult fo find in the fanfold paper for printing terminals. A console server can aleviate these problems and provide other services that make it a cost effective and reliable solution.

1 Introduction

Because most admins use CRT terminals , console only messages (ie.. those not trapped by syslogd on a UNIX host) are not saved anywhere. On printing terminals, these messages end up somewhere down the fanfold. Also if any event occurs on the terminals, there is no notification of those events. These conditions cause the System Administrator to have to go where the consoles are physically located to deal with any issues that require direct intervention. In the middle of the night, this usually means driving into work.

This paper addresses this problem using available software and hardware to create a viable and reliable console server. The parts are all cheaply available and simple to setup. The software is free. If applied correctly, this server maintains the security of the consoles while allowing multiple authorized people to connect to the console from a remote host. It simplifies the setup and lowers the footprint in the computer room. It logs all events that are on the console and can notify the appropriate people when an event occurs using email, syslogging, or paging.

The idea of the console server settles around the FreeBSD command, *watch*(8*)*. This command allows a root privileged person to observe and/or work in tandem with another tty on the machine. This capability is the one that was exploited to allow the console server to function.

There are some dangers with centralizing system consoles. If the console server goes down, access is lost to all the machines it serves. If it is compromised, then the security of the consoles is compromised. This can be especially dangerous with systems like Digital Equiptment Corperation VMS where pressing <ctrl>–P on the console can halt the processor. These issues can be dealt with accordingly to insure that you have the flexibility that the console server provides while insuring the security and availability of the systems.

2 Creating the Server

Here is a short list of necessary components needed to create a console server. These items are the ones that are currently used in the authors production environment.

First a PC that will support FreeBSD. See the HARDWARE file accompanying the FreeBSD distribution for details about what FreeBSD will and will not support. This PC should be fast enough to allow multiple people to connect. It should have enough memory to run X and multiple processes. The lowest amount of memory suggested is 32 megabytes. The disk space on the machine should be big enough for the installation and have room for the logs to be stored away for a long period of time. A multi–port card is useful, especially if there are many of consoles to connect. This example used Cyclades CY–8 boards.

Some knowledge of serial cabling is necessary to connect everything. Information on the different pinouts from the different machines you intend to serve as well as the serial ports which you intend to connect them to is handy to have during the installtion. Also suggested is to use a patch panel to simplify connections and allow flexibility in the event of an emergency. Having a rack to put everything in makes for a neat and clean installation.

A substantial amount of Category 5 cable and serial connectors will be required. In this example, DB25 connectors were used with RJ–45 female receptacles. Then, by using a crimping tools and having RJ–45 crimpets handy, wiring the cable becomes a simple task. Silk, or flat ribbon cable, was also used but is not suggested if creating long cable runs.

Last needed is a good base in system administration knowledge. Understanding of configuration of serial ports, system software, and building packaged source software is helpful to extend the capabilities of the console server. It is suggested to have some common sense ideas about security. Compromising the security of a console server is not difficult to do by making a simple mistake in configuration.

3 Planning

Planning the console server is an important step. All the hosts that are being served will have to have a serial cable run to this central point. That can be ALOT of cables, so the layout must be planned accordingly.

For external connectivity to the console server, plan on a network connection in to your LAN. This connection has the greatest possibility of security breach. Planning for a modem connection (or two) so that dial in access to the server is available in the event of network failure is also a good idea, however it also creates a security hazard. Careful thought must go into accessibility to these devices. Fortunately FreeBSD gives you some options to help make this less of a concern.

Getting the FreeBSD installation ready to go also takes a little forethought. There are some things that make it easy however. Having the configuration manuals and the current jumper(less) configurations of the associated parts available makes the pre and post installation configuration easier. Take time to determine how the disk will be subdivided to handle not only the UNIX installation, but also the logs from the server. There needs to be significant room available for the logs. Even log rotation scripts can very quickly be overwhelmed by some of the more healthy logging that can happen in times of distress.

Lastly, the platforms and OS's that are being served need to be examined for idiosyncrasies. Some hardware platforms, like older Sun systems, execute a reboot if the console drops DTR. Some applications want a special type of console for things and coding around those issues can be difficult. Hewlett Packard(HP)UNIX servers require their own 'hpterm' for the console when you enable the service processor. This can be a problem when interfacing from a vt100 compatible interface like an xterm. Each of these issues must be examined and a determination made as to whether or not the problem can be surmounted.

In our example, separate connections with two CRT terminals that supported the correct emulation were left on the patch panel so that in the event of an emergency, the terminals could be used.

4 System Implimentation

4.1 Building the Host

The console server must be assembled. Care must be taken to keep all the configuration settings of the cards handy. These are necessary when customizing the FreeBSD installation. It is assumed that the installer will have the necessary knowledge to install and configure the system.

Installing FreeBSD is a fairly simple matter for an experienced unix user. The layout must be such that there is plenty of room for console logs. Version 2.2.5–RELEASE has been used steadily now with the example console server. The INSTALLation guide is well detailed in walking a user through the configuration and basic setup. It is suggested that the base release, X11R6, and the 'sys' part of the source are installed so that a new, optimized kernel can be built. Here is a list of other packages that should be considered.

- good scripting language (perl, tk, python)
- tty utilities(screen, less, vile)
- extended shells (zsh, tcsh, bash)
- networking utilities (trafshow, tkined, tcpdump)
- xwindows utilities (fvwm, xvile, xlock, xautolock)
- security tools (runas, ssh, cops, bigbrother)

Once FreeBSD is installed, X11 must be configured. In this example, 1024x768 screen size was used. This allows four xterms to be displayed concurrently on the same screen. X11R6 provides a utility called *XF86Setup* which is a nice gui interface to creating the X configuration. Using fvwm(95) or another window manager with X that support multiple desktops, all screens can be kept open at the same time. Installing and configuring a good window manager makes things easier for non−UNIX types. In this example, fvwm95 was used to allow users familiar with Win95 to be able to use the console server comfortably..

Next is to configure the multiport boards and/or the other kernel facilities you intend to use on the console server. Here are some suggestions of devices in the FreeBSD kernel that should be considered..

- **bpfilter** − this device allows privileged users to snoop the network. It is handy for troubleshooting problems on a non−switched segment of the network. This device is required for the trafshow and tcpdump programs.
- **snp** − this device is required for the watch program.
- **ppp** − this device is required to allow ppp connections into the system.
- **cy0** − this device is required for the Cyclades Multiport Board.

4.2 Creating the Console Facilities

To create the server there are a few pieces that must be assembled. First the configuration file for *tip*(1), /etc/remote(fig 1), must be edited to add the serial ports that will be used for consoles. In the authors example, the machine name was used as ’ key’ for the file.

```
prod1|cg|cuac0g:pa=none:dv=/dev/cuac10:br#9600
prod2|ch|cuac0h:pa=none:dv=/dev/cuac11:br#9600
prod3|cj|cuac0j:pa=none:dv=/dev/cuac12:br#9600
prod4|ci|cuac0i:pa=none:dv=/dev/cuac13:br#9600
```

Figure 1 : Example of /etc/remote

Using *watch*(8), *tip*(1), and the *tee*(1) command combined, the facilities for attaching and logging the console can be created. To create the actual terminal on the console... a script was created called start.console. (fig 2)

```
#!/bin/sh echo -n "ESC]2;$*^G" tip $1 | tee -a
/var/log/prod/log.$1
```

Figure 2 : Example of start.console

This script connects the current tty to the serial port named in /etc/remote file. The echo statement allows a user to specify the titlebar for an xterm.[1] In this case, it is also set to the name of the host that this console serves. The tee statement takes standard out from the tip command and saves it to a file as well as passing standard in to it's standard out. After these are created, to create a console on the server, merely run the command in Example 3.

```
# xterm -e start.console prod1
```

Figure 3 : Example of starting a console

Now everything that comes out of the console port will go to both the screen and a file. Thus allowing anything that comes across the console to be both monitored and saved away for future reference.

To make things easier at boot time, it is suggested that either a 'console' user exists that can be used to login to the system or that in the boot–up scripts, the console user automatically performs a login. In this example we have the console user logged in by hand although it is to be changed in the future. This user must have sufficient permissions such that he or she can run tip. This means having write access to the associated cuac devices. This can be simply done by adding the console user to the ’dialer’ group. The startup scripts for this user auto create all the connections and terminals.

To make it easier for the non UNIX−literate users to access and use the console server, it is suggested to give them a familiar interface such as *fvwm95*(1). Fvwm95 looks and acts a lot like Win95. Creating a default .fvwm95rc makes life easier and can keep those same non−literate users out of trouble. Another good thing to install is xlock, to keep prying hands out; and xautolock to automatically star t xlock when the last user forgets to do it. This helps insure your security and protect your screen from being burned. Examples of .xinitrc and .fvwm95 can be found at the authors home website:

http://www.ferginc.com/~branson/console.server

5 Hardware

5.1 Wiring

When creating the console server, one of the hardest parts is to connect and manage all the wiring that is involved. Terminals, computers, printers, and modems all have a fairly standard way of wiring. However making that connection can sometimes be a lot more difficult than one would think. In general there are two types of connections that will have to be dealt with[2]:

Data Terminal Equipment (DTE)

Pin 2 is Transmit

Pin 3 is Receive

Pin 4 is Ready to Send

Pin 5 is Clear To Send

Pin 6 is Data Set Ready

Pin 7 is Ground

Pin 8 is Data Carrier Detect

Pin 20 is Data Terminal Ready

Data Communications Equipment (DCE)

Pin 2 is Receive

Pin 3 is Transmit

Pin 4 is Clear To Send

Pin 5 is Ready to Send

Pin 6 is Data Terminal Ready

Pin 7 is Ground

Pin 8 is (also) Data Terminal Ready

Pin 20 is Data Set Ready and Data Carrier Detect

Both devices in this case are DTE devices. To make a good connection between the m, the connections must be crossed in such a way as to allow data on the transmit data pin from one side go into the receive data pin on the other side. More simply said, a DTE connector is plugged into to a DCE connector to create a correct connection. The device to do this is called a Null Modem (fig 4).

Pin	to	Pin
2	to	3
3	to	2
4	to	5
5	to	4
6	to	20
7	to	7
8	to	20
20	to	6,8

Figure 4 : Example of a DB25 Null Modem Pinout

There are many ways to create this connection. A crossover cable can be made, an adapter can be used, or the connector can have the crossover wired into it. In this example, the crossing was done at the connector. Proper labeling will insure that visual examination is enough to see how the connection is made. Crossing at the connector also allows flexibility in making the connections. Changing the connection from one type to the other is as simple as replacing the connector.

Standardizing the pinouts that are coming in is another good idea. Having to deal with the different types of pinouts in a centralized area sounds like a good idea at first, however it makes it complex in determining what should be what. For the example configuration, the null modem was added at the end where the machine was. The null modem was an adapter or a specially pinned and labeled connector which connected the 25 pin male coming from the server to the RJ45 connector that lead back to the console server. This makes each incoming connection a DCE type of connection.

For more information on serial connectivity, see Chapter 8 in *The Unix System Adminstration Handbook* or see "An Incomplete guide to Parallel and Serial port pinouts" at http://www.geocities.com/TheTropics/Shores/2250/pinout.html.

5.2 Cabling

The RJ45 cable then leads back to a patch panel. Each cable is labeled as to the machine that it comes from. It then plugs into the receptical for which it is labeled. This makes it simple to relocate and test consoles. It also allows for a central area for all the consoles to be organized. Each receptical on the patch panel leads to a serial cable that goes to the Cyclades board.

There are extra recepticals on the patch panel connected to spare terminals. This is a very good idea to have around in case the console server goes down or the functionality of the specific terminal type is needed for something. For example the HP's "service processor" is geared to using the "hp" terminal type. So by reconnecting a terminal, the service processor can be accessed in its native terminal type. The spare terminals must have a nullmodem installed to negate the nullmodem installed at the server side of the connect ion.

Once the path is made from the server to the patch panel, test each connection on the terminal. This insures a known good connection for troubleshooting.. This will be handy when starting to verify connections into the console server . A good way to test these connections is to make a loopback connector(fig 5). The simplest way to loopback is to short pins 2 and 3 to each other. Typically this is done with a paperclip for female sockets and with a small flat head screwdriver for male sockets. A simple loopback can be wired using the example provided. Any keystrokes on the input side for that connection, either a terminal or a window in the console server, should be visible. Using a loopback connection and working the way back to the terminal or server allows simple and effective troubleshooting.

Pin	to	Pin
2	to	3
4	to	5
6	to	8,20

Figure 5 : Example of a Loopback Pinout

At this point, the rest of the work is grunt work to get all the consoles connected and working. It takes time and patience but is well worth the effort. Generally the problems that will be encountered fall into these categories:

- Baudrate – Inconsistent from the console port on the server to the console server. Generally, most console ports are 9600 unless setup otherwise. The file to examine is /etc/ttys or /etc/inittab on the host side, and /etc/remote on the console server side.
- Misconnection of signals and/or data – A null modem will generally solve these type problems, however there are incidents where a breakout box becomes necessary to find the problem.
- Terminal Type – Most systems default to a specific terminal type. In general a vt100 emulation terminal will work. There are others that need specific types. HP needs an hp type terminal to work with the service processor.
- Bad Cable/Connector – "Cable Happens" says a sign in the support center. Check all cabling and connectors with a known good connection, a multimeter, or best of all, a cable tester.

6 Using the Console Server

Now that the console server is configured and connected. It is time to put it to active use. There are a number of activities that can be done on this server to assist a system administrator in being proactive and better reactive to system problems. Connecting to a console becomes very easy when the need arises. If the console server itself is available, it is merely selecting it from the other windows on the screen. If a history is needed, the logs can be accessed by opening a window on the console server and viewing the logs with *more*(1) or *less*(1).

If the console server is not in the immediate vicinity, attaching to a console becomes very simple using the watch command. The acons script makes this easier.

```
#! /usr/local/bin/perl
($console) = @ARGV; open(W,<AB>w|<AB>);
while(<W>){
   ($user,$tty,$from,$login,$idle,$what)=
     split(/\s+/,$_,6);
   next unless $what =~ /tip/;
   if ( $what =~ /$console/ ){
        exec <BB>watch -W $tty<AB>;
   }
}
print "No console for $console running \n";
```

Figure 6 : acons perl script

To attach to the desired console, it becomes a matter of running a single command as root(fig 7). To do this we use the *runas*(8) command. The authors version of runas is a suid *perl*(1) script that has more precise control of root permissions. It is also available at the authors homepage.

```
branson@cyadris > runas acons prod1
```

Figure 7 : Example of attaching to a console

7 Add on Features

There are some features that can be simply added to allow for more freedom and ability on the console server. These features greatly enhance the console server as a troubleshooting tool.

By adding dial–in access, remote access to the consoles can be made independent of most network problems. Security should be considered allowing this access. There are some utilities on the net that can help with this. Protection like dialback at the dial–in level can be added so that the machine will call the user back to verify the phone number. A security scheme like S/Key can be added for validation of one–time passwords.

By adding tools such as the bpf device and *trafshow*(1), networking problems can be better troubleshot by looking at the actual packets as they transverse the network. Other tools such as *tkined*(1)give a graphical representation of the network and allow a user use network queries to gather information on the machines. *Tkined* can also send alerts based on perceived network problems.

Adding tools such as *swatch*(1) and big brother do proactive examination of system events and can help spot potential problems before they become disasters. Installing paging software allows the console server to alert the appropriate parties of problems just as soon as they happen. This helps shorten the time to a solution.

8 Conclusion

Creating a console server is simple, cost effective, and relatively easy to impl ement. There are multiple benefits associated with a console server including: ease of use, logging of console events, remote access, and increased security. It allows simple and effective consolidation of console cables and terminals. It can be used for proactive troubleshooting by informing the responsible parties of events on the host. It can be used for reactive troubleshooting by the storing of all console events and allowing direct access to the consoles from a remote location. Although there are security concerns, they can be handled to make the console server and effective solution to the problems associated with multiple traditional consoles.

Bibliographie

[1] Eric F Johnson et Kevin Reichard, *Using X*, MIS:Press (a subsidiary of Henry Holt and Company,Inc.), 115 West 18th Street, First Edition 1992.

[2] Evi Nemeth, Garth Snyder, Scott Seebass et Trent R. Hein, *Unix System Administration Handbook, Second Edition*, Prentice–Hall, Inc., Englewood Cliffs, NJ , 1995.

[3] Mahendra, *An Incomplete Guide to Parallel and Serial Port Cable Pinouts*, http://www.geocities.com/TheTropics/Shores/2250/pinout.html, April 1998.

Using FreeBSD for a Console Server