Using FreeBSD to Render Realtime Localized Audio and Video

John H. Baldwin
The Weather Channel
Atlanta, GA 30339
jhb@FreeBSD.org, http://people.FreeBSD.org/~jhb

Abstract

One of the largest selling points for The Weather Channel (TWC) is the ability to generate localized content for its subscribers, specifically local weather forecasts. To facilitate this localized content, TWC deploys smart devices in cable head ends. These smart devices are called STARs (Satellite Transmitter Addressable Receivers) and are responsible both for collecting weather data and displaying the data to the user in an audio and video presentation.

TWC decided to produce a next generation STAR that was more flexible than the current generation as well as cheaper. FreeBSD was chosen as the platform for these devices. Although FreeBSD was largely suitable for this application, a few modifications were required and several workarounds were employed. The end result is a PC built mostly of off-the-shelf components that is able to render broadcast quality audio and video in realtime.

1 Introduction

Most cable channel operators provide national or regional feeds from a single source to cable head end operators across the country and world. These feeds are transmitted via satellite from the broadcasters to the head ends. The head ends then distribute these feeds to viewers over cable. The Weather Channel is a unique broadcaster in that part of its presentation to viewers is localized to provide local weather observations and forecasts. It is not feasible to generate the content of these local forecasts at TWC and then distribute them over satellite to each of the head ends. There is just not enough bandwidth. Instead, a smart device known as a STAR is deployed in each head end. In addition to a national audio and video feed, TWC sends weather-related data to the STARs out in the field including local observations, thirty-six hour text forecasts, daily extended forecasts, radar images, and severe weather alerts. In addition, TWC can send software updates and other non-weather-related data over the satellite.

For example, every hour thousands of automated observation stations all over the United States sample the weather conditions at their location and transmit this data to the National Weather Service (NWS). Most STARs in the field are assigned an observation point. TWC receives this data from NWS and then sends it over the satellite where it is received by each of the STARs. Each STAR saves any data that is relevant to its configuration and discards the rest. The STAR may then display those observations either on top of the national feed in a small bar at the bottom of the screen, or it may render a full-screen presentation.

Currently TWC broadcasts its feed as an analog NTSC signal for the core network. TWC has started migrating to a digital MPEG signal to meet the demands of the cable industry as it begins the migration to digital. As part of this move, TWC is deploying a next generation STAR device known as the IntelliStar.

2 What is an IntelliStar and What Does It Do?

An IntelliStar must perform two major tasks concurrently. First, it must collect weather data from the satellite feed and store the data relevant to the location it is configured for. This data includes current observations, radar images, satellite images, and several different forecasts. Second, the device must display this weather data to viewers at broad-

cast quality that compiles with FCC regulations. This includes rendering both full screen video as well as being able to render graphic objects on top of a live video feed. When a box is down, it can not deliver local content on the analog output and can not deliver any digital output, so down time is very visible to customers and must be avoided.

In addition, since other networks do not deploy devices to head ends to generate localized content, an IntelliStar must operate as a "black box" from a cable head end technician's perspective. For example, head end technicians are accustomed to using the power button to turn things off, so the device must handle powering off via the power button gracefully. Also, the device must operate without having a keyboard, mouse, or monitor connected. Finally, the device should not require any interaction from a head end technician beyond using the power button to power the device on and off.

In addition to these requirements, TWC wants to take advantage of recent technology innovations. The current generation of STARs, known as the WeatherSTAR XL, are built from SGI boxes running IRIX with additional proprietary hardware. The IntelliStar, on the other hand, is built largely from commodity PC hardware with limited proprietary hardware. This has resulted in a cheaper and more compact device that is easier to upgrade in the future. For example, the actual rendering of video content is done using the hardware acceleration of an off-the-shelf AGP graphics adapter. Should the need for more rendering horsepower arise in the future, the graphics adapter can be replaced with a newer adapter without any changes needed in either proprietary hardware or software. To accomplish this goal, much of the realtime requirements for audio and video have been offloaded into proprietary hardware.

3 Why FreeBSD?

For the software platform on the IntelliStar, TWC also wants to stick with largely off-the-shelf software. The main device needs a stable and reliable general purpose OS that is easily maintainable. The TWC applications are written in a mixture of C++ and Python and are multithreaded, so decent support for those platforms is required. Finally, TWC needs a platform that can be easily updated in the field over the one-way satellite feed. The final decision on the OS for the IntelliStar was made in the summer of 2001. Prior to that, the development platform was Red Hat Linux 7.0. FreeBSD was chosen over Red Hat 7.0 for three primary reasons: stability and maturity of the virtual memory subsystem, a current and working tool chain, and a complete and self-consistent distribution from one source.

3.1 VM Stability

As mentioned before, down time is very noticeable. If a box is down, then thousands of viewers may not receive their local forecast. At the time of the platform decision, TWC knew that the development of the IntelliStar would take some time. Thus, TWC wanted an OS branch that was stable and mature but would not be obsolete by the time the IntelliStar was deployed. The 2.2.x Linux kernel series was nearing its end of life cycle as the 2.4.x Linux kernel series was just ramping up. However, the virtual memory subsystem of the early 2.4.x series had severe stability problems. In addition, Linux changed over to an entirely new virtual memory subsystem during the early 2.4.x series. FreeBSD, on the other hand, employed a mature, stable, and well-tested virtual memory subsystem in its 4.x branch. TWC developers noted that even on the Linux developer mailing lists, FreeBSD's virtual memory subsystem was held up as the benchmark to test the new Linux subsystems against. As a result, TWC developers were much more comfortable with FreeBSD's virtual memory subsystem than with Linux's. FreeBSD 4.x was also not nearing its end of life cycle during 2001, so TWC developers were not worried about having to perform major OS upgrades in the middle of their development cycle or immediately after the initial deployment.

3.2 Compiler

The software in TWC's STAR group largely consists of multithreaded C++ applications, and TWC required a compiler toolchain that would work with these applications. The compiler that shipped with Red Hat 7.0 was version 2.96 of the GNU C Compiler (gcc). This version of gcc was not an official release of gcc by the GNU Project. Instead, it was a snapshot of the 3.0 development branch of

GCC with additional fixes from Red Hat developers. Out of the box this pre-release compiler was unable to compile simple multithreaded C++ applications. Numerous patches were required from Red Hat before the compiler could compile this simple test case. Other Linux distributions used releases from the gcc 2.91 series which were unable to compile TWC's applications due to their more limited C++ support. FreeBSD, on the other hand, shipped with supported releases from the gcc 2.95 branch. This compiler was able to handle multithreaded C++ applications out of the box without the need for further patches and included sufficiently recent C++ support to compile TWC's applications.

3.3 Self-Contained OS

IntelliStar devices will be deployed all over the continental U.S., and TWC wanted to be able to upgrade the base OS if necessary in addition to TWC's custom applications. The only data link that is reliable for all STARs is the one-way transport from TWC out to the STARs via satellite. Thus, update systems that require a two-way link are not suitable. Since FreeBSD is an entire OS from one source rather than a collection of packages from different sources like Linux distributions, it is easier to generate an arbitrary release that is self-consistent and use that complete distribution for either installs or to upgrade units in the field. OS components can be upgraded merely by sending a tarball out to the deployed units without having to worry about keeping a local package database up to date. Formal packages are only used for third party libraries and applications that are not part of the base OS and TWC applications.

Remotely upgrading the kernel is also safer and easier with FreeBSD than with Linux. With Linux one has to ensure that the lilo boot blocks are updated after a new kernel is installed or one may end up with an unbootable box. FreeBSD's bootstrap simply looks for the loader by name in a UFS filesystem, so simply updating the actual kernel file is all that is needed to upgrade the kernel. Recent versions of Linux now offer the GNU GRUB boot loader which does not suffer from the same limitations as lilo. However, GNU GRUB is still under development and was not in use when the final OS decision was made.

Finally, TWC could easily build a custom release of FreeBSD including customized installation scripts. TWC currently maintains a small set of local patches. Using the `LOCAL_PATCHES` and `LOCAL_SCRIPTS` features of FreeBSD's release process as detailed in [release.7], TWC is able to build a full release with these patches that can be installed either from a CD or over the network like any other FreeBSD release. By using PXE network booting and FTP install over a LAN, the installation of a new machine simply involves plugging the box in and turning the power on.

4 Changes from stock FreeBSD

The current generation of IntelliStars are based on release 4.7 of FreeBSD. Although the stock FreeBSD release met many of the requirements, several modifications were required. These modifications included backporting support for the Advanced Configuration and Power Interface (ACPI) from 5.0, modifying the `sio` driver to share PCI interrupts, adjusting the maximum size of receive socket buffers, and several fixes for sysinstall to allow for more flexible scripted installations.

4.1 Backporting ACPI

As mentioned earlier, one of the requirements for the IntelliStar was that it require little interaction from head end technicians once it is deployed. Specifically, head end technicians should be able to safely power off the device by flipping the power switch. With the advent of ACPI, implementing this feature becomes possible as an ACPI system will inform an ACPI-aware OS when the power button is pressed. The OS can safely shut down and then ask the computer to power itself off via ACPI. The details of how this works can be found in the ACPI specifications which are available at [ACPI].

The 5.x development branch of FreeBSD contained a mostly functional ACPI driver including support for power button events. Thus, the same functionality could be brought to the version of FreeBSD used in the IntelliStar by backporting a stripped down ACPI driver to the 4.x branch. The majority of the ACPI driver consists of Intel's ACPI Component Architecture (ACPICA) which is OS independent and freely available at [ACPICA]. Thus the

only code that needed to be backported was the OS shim that wraps around ACPICA as well as some supporting code in other areas of the kernel used by the shim.

The changes made to the OS shim included using proper synchronization primitives and avoiding the use of a private taskqueue. 5.x uses mutexes to protect data structures while 4.x still uses the spl mechanism to protect top-half kernel code from being interrupted. Thus, macros were added that use `splhigh` and `splx` for synchronization with the ACPI interrupt handler on 4.x and a mutex on 5.x. FreeBSD 4.x also does not support the same software interrupt API used by the ACPI code in 5.x to implement a private taskqueue. Thus, the backport to 4.x simply uses the system taskqueue for ACPI events instead. These changes were wrapped in appropriate precompiler conditionals and committed to the 5.x branch prior to 5.0-RELEASE.

Changes made to other parts of the kernel include backporting entire subsystems and drivers as well as modifications to existing subsystems and drivers. The ACPI driver required support for the `bus_set_resource` and `bus_get_resource` methods in the `nexus` driver, so these changes were backported and committed to the 4.x branch prior to 4.8-RELEASE. The ACPI driver also used the `resource_list_print_type` helper function from the resource manager, so that function was backported and committed prior to 4.8-RELEASE. The ACPI driver also depended on the new power subsystem and `pmtimer` driver which debuted in 5.0-RELEASE. The actual code for both of these subsystems compiles directly on 4.x but does require some simple changes to the i386 clock and low level interrupt code. In addition, the use of the power subsystem requires several changes to the `apm` driver. Several of the changes made to the `apm` driver in the 5.x branch were merged prior to 4.8-RELEASE to minimize the size of the local patches TWC maintains.

The other significant difference between ACPI support in 5.x and ACPI support for 4.x is that the backported ACPI driver does not include the PCI support code. This means that the backported ACPI driver does not route PCI interrupts using the `_PRT` tables or enumerate host to PCI bridges. The ACPI PCI code depends on large changes to the PCI driver made in 5.x that are too large to backport to 4.x. The rest of the functionality provided by the ACPI driver is present in the backport including power button events, suspend and resume, battery status, AC adapter status, CPU throttling, thermal zones, and the ACPI timer.

4.2 Sharing `sio` PCI Interrupts

In FreeBSD, there are two main types of interrupt handlers: fast interrupt handlers and non-fast interrupt handlers. Fast interrupt handlers execute with slightly less latency to the original interrupt request and than non-fast handlers. Also, all interrupts are blocked while executing a fast interrupt handler. Fast handlers cannot share an interrupt source such as an interrupt request (IRQ) line with other interrupt handlers. Non-fast interrupt handlers, on the other hand, can share an interrupt source with other non-fast interrupt handlers. This is enforced in `bus_setup_intr` by having attempts to register a fast interrupt handler on an interrupt source that already has an interrupt handler and attempts to register a non-fast interrupt handler on an interrupt source that already has a fast interrupt handler fail.

The purpose of fast interrupt handlers is to minimize latency for devices that require very low latency. A prime example of such devices are the serial ports found in PCs which have very small data buffers. As a result, if interrupt latency is high, characters will be dropped. Thus, the driver for serial ports, `sio`, uses fast interrupt handlers.

When not using I/O APICs to manage interrupts on a PC, there are only sixteen interrupt sources in the form of ISA IRQ lines. Most of these IRQ lines are reserved for ISA devices. ISA devices cannot share interrupts, so PCI devices are restricted to using the IRQ lines not used by any ISA devices. Due to the limited number of free IRQ lines and and the increasing number of PCI devices in PCs, PCI devices are usually required to share whatever IRQ line is allocated to them with other PCI devices. Thus, if a PCI device driver uses a fast interrupt handler it can either block other devices from registering non-shared interrupt handlers on the same interrupt source if it is the first driver to register a handler for that source, or it can fail to register its handler if another driver has already registered a handler.

The `sio` driver handles the second case but does not handle the first case. To handle the second case, the

`sio` driver first attempts to register its handler as a fast interrupt handler. If that fails, it tries to register it as a non-fast interrupt handler. However, the `sio` driver currently has no way of detecting the first case and properly handling it.

The IntelliStar uses a PCI modem managed by the `sio` driver that just happens to be the first PCI device to register its interrupt handler for its interrupt source during the boot phase. As a result, other PCI devices using the same interrupt source such as an Ethernet adapter are unable to register their interrupt handler and fail to attach. The solution that TWC developed was to add a flag to the `sio` driver's global attach routine to specify whether or not the driver should attempt to register a fast interrupt handler or if it should only use a non-fast interrupt handler. The attachments for different busses can then set this flag to force `sio` to share its interrupt source with other devices. For example, the PCI and PCCard busses force the `sio` driver to share its interrupt source in TWC's patch.

The patch was submitted for review to the FreeBSD developers but was rejected as being too much of a hack. Several of the developers still wished to allow the `sio` driver to use fast interrupts when possible and wanted to fix the problem TWC had in the drivers for the busses themselves such as the PCI bus device driver instead of in `sio`. However, no progress was ever made on even how to go about doing that, so TWC continues to maintain this bug fix as a local patch.

4.3 Increasing Size of Socket Receive Buffers

As mentioned earlier, an IntelliStar receives not only live video over the satellite, but also data including weather data and software updates. For the IntelliStar, this data is transmitted over the satellite in a multicast UDP data stream alongside the video stream. The bundling of the data stream with the video stream is managed by an external integrated receiver/decoder (IRD) which provides the data stream as the original UDP multicast stream to one of the Ethernet ports on the IntelliStar. Since this communication transport is only one-way, TWC has no way of knowing if an IntelliStar has lost data due to the socket buffer overflowing. If the link were two-way, then a reliable protocol such as TCP could detect that data was lost and request a retransmission, but with the one-way link that option is not available. Since rendering the on-screen graphics is higher priority than reading data off the socket, there can be enough latency between reads of the receiving socket for the standard socket buffer size of 41600 bytes to overflow. To fix this, TWC uses `setsockopt` to increase the size of the receiving sockets buffer to 512 kilobytes. At our current data rate this allows up to one full second of data in the buffer. The default maximum size for a socket buffer in FreeBSD is 256 kilobytes, however, so TWC added an entry to `/etc/sysctl.conf` to increase this limit to four megabytes per socket buffer via the `kern.ipc.maxsockbuf` sysctl.

4.4 Scripting Enhancements for sysinstall

One of the benefits of FreeBSD mentioned in an earlier section is the ability to easily build a customized installation process. This can be done with the default installation utility, sysinstall, via installation scripts. This custom installation allows TWC to quickly and easily install all of the necessary software on IntelliStars before they are deployed into the field.

While working on the scripts for TWC's custom release, a few limitations and gaps were found in sysinstall's scripting support. TWC extended sysinstall's scripting support to address these shortcomings. All of these changes to sysinstall were committed prior to 4.7-RELEASE. To support more flexible installation scripts, TWC also made some changes to the release Makefile.

The first two changes made to sysinstall allowed scripted installs to optionally be more interactive. The first change added the `diskInteractive` variable to the disk layout editors. If this variable is set when invoking the `diskPartitionEditor` or `diskLabelEditor` commands, then the interactive disk layout editors will be used instead of requiring a fully scripted disk layout. The second change added the `netInteractive` variable to the network interface setup dialog. If this variable is set when invoking the dialog via the `tcpMenuSelect`, `mediaSetFTP`, or `mediaSetNFS` commands, then the user is asked if they wish to use DHCP or IPv6 rather than assuming that neither is desired.

The next change to sysinstall fixed a bug in the han-

dling of the `noError` variable. As documented in [sysinstall.8], the `noError` variable causes sysinstall to ignore failures from the next command executed. Usually sysinstall will abort and stop executing a script if a command fails. The `noError` variable allows a script to continue if a non-fatal error occurs. The bug was that the `noError` variable was only cleared if a command failed. Thus, if the non-fatal command immediately after `noError` was set succeeded, the variable remained set and the subsequent failure of a later command would be bogusly ignored. The fix was simply to always clear `noError` after executing a command.

The fourth change to sysinstall involved the addition of the `mediaClose` command. This command simply executes the internal function by the same name. For an install using a CD as the installation media, this will unmount the CD allowing it to be ejected from the drive. TWC uses this at the end of CD installations to unmount the CD prior to displaying a dialog box prompting the user to eject the CD.

The final changes made were to the release Makefile and not the installation utility itself. The first change was to add the dialog program to the memory filesystem used as the root filesystem during installations. This made the dialog program available to shell scripts executed by the installation scripts. This allows for more complex installation scripts that can interact with the user using the various tools described in [dialog.1].

For example, the TWC install begins with a menu box prompting the user for the type of machine to install: an IntelliStar or a development machine. Depending on which option the user selects, different parameters are used. This is accomplished by having the top level install script execute a shell script. This shell script uses the dialog command to display the menu and obtain the user's choice. The shell script then generates a configuration script on the fly. After the shell script finishes, the top level configuration script loads the configuration script generated by the shell script and executes it.

To support this change, TWC added a `NO_FLOPPIES` variable to the release Makefile to disable building of boot floppies. Adding the dialog program to the memory filesystem made the memory filesystem too large to fit on floppies. By defining the `NO_FLOPPIES` variable during the release build, TWC's custom release completed without an error. TWC does not use floppies for any of its installations, so the loss of floppies as a installation boot media was not a problem. The addition of the `NO_FLOPPIES` variable was committed prior to 5.0-RELEASE and will be merged to the 4.x branch prior to 4.9-RELEASE.

5 Workarounds for FreeBSD Problems

In addition to the problems above, FreeBSD is lacking in two other areas as well that TWC chose to work around in its own software instead of modifying FreeBSD. The first area involves negative nice priorities and is worked around fairly easily. The second area consists of a couple of problems with the userland thread implementation employed in FreeBSD 4.7-RELEASE.

5.1 Nice is too Unnice

The TWC software that runs on the IntelliStar consists of several applications of varying importance. For example, rendering the video presentation is very important and receiving data is slightly less important. Most other tasks are not all that important as far as latency is concerned. Thus, negative nice values are applied to the important applications to ensure that they are not starved by any background processes.

Initially a nice value of -20 was used for the most important process. However, during development an infinite loop bug was encountered and the box locked up. Some simple tests of a program that executed an infinite loop at a nice value of -20 verified that the looping process starved all other user processes on the box. This was surprising since it was expected that the CPU decay algorithm of the scheduler would sufficiently impact the priority of the important process so that other userland processes would receive some CPU time. As a matter of fact, the CPU decay algorithm will not decay a nice -20 process enough to allow normal processes with a nice value of zero to execute. The explanation can be found in a simple examination of the code.

The priority of a userland process is calculated by the following code snippet from the `resetpriority`

function:

```
newpriority = PUSER +
    p->p_estcpu / INVERSE_ESTCPU_WEIGHT +
    NICE_WEIGHT * p->p_nice;
newpriority = min(newpriority, MAXPRI);
p->p_usrpri = newpriority;
```

The `p_nice` member of `struct proc` holds the nice value and `p_estcpu` holds an estimate of the amount of CPU that the process has used recently. This field is incremented every statclock tick in the `schedclock` function:

```
p->p_estcpu = ESTCPULIM(p->p_estcpu + 1);
```

The `ESTCPULIM` macro limits the maximum value of `p_estcpu`. Its definition along with the definition of other related macros follows:

```
#define NQS     32
#define ESTCPULIM(e) \
    min((e), INVERSE_ESTCPU_WEIGHT * \
        (NICE_WEIGHT * PRIO_MAX - PPQ) + \
        INVERSE_ESTCPU_WEIGHT - 1)
#define INVERSE_ESTCPU_WEIGHT   8
#define NICE_WEIGHT     2
#define PPQ             (128 / NQS)
#define PRIO_MAX        20
```

Thus, the maximum value of `p_estcpu` is 295.

Since `p_estcpu` is never less than zero, a process with a nice value of zero will have a userland priority greater than or equal to `PZERO`. For a process with a nice value of -20, the total nice weight ends up being -40. However, the maximum weight of the CPU decay is 36. Thus, with a nice value of -20, the CPU decay algorithm will never overcome the nice weight. Thus, a lone nice -20 process in an infinite loop will starve normal userland processes with a nice value of zero. In fact, since the maximum CPU decay is 36, any nice value less than -18 will produce the same result.

However, according to a comment above `updatepri`, `p_estcpu` is supposed to be limited to a maximum of 255:

```
/*
 * Recalculate the priority of a process after
 * it has slept for a while.  For all load
 * averages >= 1 and max p_estcpu of 255,
 * sleeping for at least six times the
 * loadfactor will decay p_estcpu to zero.
 */
```

If this is the case, then the maximum CPU decay weight is merely 31, and any nice value less than -15 can starve normal userland processes.

One possible solution would be to adjust the scheduler parameters so that a nice -20 process did not starve userland processes. For example, `INVERSE_ESTCPU_WEIGHT` could be lowered from eight to four. However, increasing the strength of the CPU decay factor in the scheduling algorithm might introduce other undesirable side effects. Also, such a change would require TWC to maintain another local patch to the kernel. TWC decided to keep it simple and stick to nice values of -15 and higher.

5.2 Userland Threads

Two of the larger problems TWC encountered were due to limitations in FreeBSD's userland threads implementation. As mentioned earlier, TWC's software consists largely of multithreaded C++ applications. Both problems stem from all userland threads in a process in FreeBSD sharing a single kernel context. First, when one thread calls a system call that runs in the kernel, all of the threads in that process are blocked until the system call returns. Secondly, all the threads in a process are scheduled within the same global priority. Both of these problems are demonstrated in one of the TWC applications that contains two threads. One thread is responsible for rendering frames, and the other thread loads textures into memory from files. The rendering thread is much more important than the loading thread since the loading thread preloads textures and can tolerate some latency whereas the rendering thread must pump out at least thirty frames every second.

When the loading thread is loading a large file into memory, it can temporarily starve the rendering thread. The internal implementation of the `read` function in the thread implementation uses a loop of non-blocking `read` system calls. However, if the entire file is resident in memory already, then the non-blocking `read` will copy out all of the file to

userland. Especially for large files, this data copy may take up enough time to delay the rendering thread by a few frames. To minimize the effects of this long delay, reads of large files are broken up into loops that read in files four kilobytes at a time. After each read, `pthread_yield` is called to allow the rendering thread to run. If the two threads did not share their kernel context, then when the rendering thread is ready to run it could begin execution on another CPU immediately rather than having to wait for the copy operation to complete.

The second problem is that all threads within a process share the same global priority. In the application in question, the rendering thread is the most important user thread in the system. Therefore, its process has the highest priority. The loading thread, however, is less important than threads in some of the other processes executing TWC applications. Since the two threads share the same global priority, the loading thread ends up with a higher priority than the more important threads in other processes. If the two threads had separate kernel contexts, then the rendering thread could keep its high priority without requiring the loading thread to have a higher priority than threads in other processes. TWC currently does not employ a workaround for this problem and so far no real world anomalies have been attributed to it.

TWC considered using an alternate thread library to work around these problems. Specifically, the thread library contained in the LinuxThreads port described at [LinuxThreads]. However, there are binary incompatibilities between the structures defined by FreeBSD's thread library and the LinuxThreads' thread library. Thus, any libraries used by a multithreaded application that use threads internally must be linked against the same thread library as the application. As a result, for our applications to use LinuxThreads, all of the libraries they link against that use threads internally would also have to link against LinuxThreads. As a result of those libraries using LinuxThreads, other applications that use those libraries would also have to link against LinuxThreads. This would require TWC to custom compile several packages including XFree86, Mesa, Python, and a CORBA ORB as well as other applications depending on those packages rather than using the pre-built packages from stock FreeBSD releases. Since the workarounds for FreeBSD's thread library were not too egregious, they were chosen as the lesser of two evils.

Looking to the future, TWC is very excited about the ongoing thread development in FreeBSD's 5.x series. The more flexible threading libraries in that branch should eliminate most of the current problems with FreeBSD's current thread library. At the moment, however, TWC is uncomfortable with deploying 5.x until it is more proven and mature.

6 Conclusion

While FreeBSD may not have been a perfect fit out of the box for the IntelliStar, it was successfully adapted to the IntelliStar's needs with relatively minor effort. The first IntelliStar units began generating and delivering content to live viewers in March of 2003. As of the time of this writing 24 units are deployed across the continental U.S. All of these units deliver the Weatherscan Local network which delivers 24/7 localized weather programming. More Weatherscan units are schedule to roll out during the rest of the year, and IntelliStars should begin replacing older STARs on the main TWC channel in 2004.

7 Acknowledgments

Thanks to The Weather Channel for funding the writing of this paper, the FreeBSD development described in this paper, and numerous other FreeBSD improvements. Thanks also to all the of the contributors to the FreeBSD Project. Without their efforts, there would not be an OS to build upon. Special thanks to those who reviewed and critiqued this paper including Chris McClellen, Sam Leffler, and Gregory Shapiro. A special thanks is due as well to the FreeBSD ACPI developers who added the initial ACPI support to FreeBSD. Without that contribution, implementing soft power-off would have been much more difficult and very likely much more of a hack.

8 Availability

All of the changes to FreeBSD that have not already been committed to the tree are freely available at

the following URL:

`http://www.FreeBSD.org/~jhb/patches/`

Specifically, there are three patches of interest in that directory. The first two patches are for the backport of ACPI to 4.7-RELEASE and 4.8-RELEASE and are contained in the files `acpi_4.7.patch` and `acpi_4.8.patch`, respectively. In addition to those patches, the following files and directories must be checked out from the 5.x branch on top of an appropriate kernel source tree:

- `sys/contrib/dev/acpica`
- `sys/dev/acpica`
- `sys/i386/acpica`
- `sys/i386/include/acpica_machdep.h`
- `sys/i386/isa/pmtimer.c`
- `sys/kern/subr_power.c`
- `sys/sys/power.h`

The other patch is in the file `sio_shareirq.patch` and includes the changes to fix the `sio` driver to share interrupts with other PCI devices.

More information about FreeBSD and the FreeBSD Project is available at [FreeBSD]. For more information about The Weather Channel and its products, please see [TWC].

References

[ACPI] ACPI - Advanced Configuration and Power Interface, `http://www.acpi.info`

[ACPICA] Instantly Available Technology - ACPI, `http://developer.intel.com/ \ technology/iapc/acpi/downloads.htm`

[dialog.1] *Dialog*, FreeBSD General Commands Manual, `http://www.FreeBSD.org/cgi/ \ man.cgi?query=dialog&sektion=1& \ manpath=FreeBSD+4.8-RELEASE`

[FreeBSD] FreeBSD Project, `http://www.FreeBSD.org`

[GNU GRUB] GNU GRUB, `http://www.gnu.org/software/grub`

[LinuxThreads] LinuxThreads, `http://www.freebsd.org/cgi/ \ url.cgi?ports/devel/linuxthreads/ \ pkg-descr`

[NWS] NOAA - National Weather Service, `http://www.nws.noaa.gov`

[Python] Python Programming Language, `http://www.python.org`

[Red Hat] Red Hat, `http://www.redhat.com`

[release.7] *Release*, FreeBSD Miscellaneous Information Manual, `http://www.FreeBSD.org/cgi/ \ man.cgi?query=release&sektion=7& \ manpath=FreeBSD+4.8-RELEASE`

[sysinstall.8] *Sysinstall*, FreeBSD System Manager's Manual, `http://www.FreeBSD.org/cgi/ \ man.cgi?query=sysinstall&sektion=8& \ manpath=FreeBSD+4.8-RELEASE`

[TWC] The Weather Channel, `http://www.weather.com/aboutus`

[XFree86] XFree86, `http://www.xfree86.org`