

CS 486 – Capstone Project
Software Design Specification
(Revision 1.0)

Submitted to
Dr. Doerry

By
Team Fugu:
Erik Wilson
Ben Atkin
Nauman Qureshi
Thad Boyd

On
March 15, 2004

Table of Contents

1. Introduction.....	1
2. System Architecture Overview.....	1
2.1. OS Tools for OpenBSD.....	2
2.2. Automated Installer.....	2
2.2.1. Disk Image Creator.....	2
2.2.2. Configuration Parser.....	2
2.2.3. Installer Runtime.....	2
2.3. Automatic Patcher.....	2
2.3.1. Installation and Setup Script.....	3
2.3.2. Patching Scripts.....	3
2.4. Runtime Order.....	3
3. Disk Image Builder.....	4
3.1. Design Overview.....	4
3.2. Subsystem Designs.....	5
3.2.1. Disk Image Command-Line Utility.....	5
3.2.2. Disk Image Build Scripts.....	5
3.2.3. Automated Installer Source and Makefiles.....	5
3.2.4. Interactive Installer Build Scripts.....	6
3.2.5. OpenBSD Source Tree.....	6
3.3. Program States.....	6
3.4. Command-Line Interface.....	7
3.4.1. Commands.....	7
3.4.2. General Options.....	7
4. Configuration Parser.....	8
4.1. Design Overview.....	8
4.2. Program States.....	8
5. Installer Runtime.....	9
5.1. Design Overview.....	9
5.2. Subsystem Designs.....	10
5.2.1. System Bootup.....	10
5.2.2. Installer Script.....	10
5.2.3. Pre-Install.....	10
5.2.4. Disk Partitioning and Formatting.....	11
5.2.5. Basic Install.....	11
5.2.6. Post-Install.....	12
5.3. Program States.....	13
5.3.1. Overview and Preinstall.....	13
5.3.2. Disk Setup.....	14
5.3.3. Basic Install from Configuration.....	15
5.3.4. Finalization and Post-Install.....	16
6. Automated Patcher.....	17
6.2. Design Overview.....	18
6.3. Program States.....	19
6.3.1. Main Tpatche Module.....	20
6.3.2. Source Patching Module.....	20
6.3.3. Binary Patching Module.....	20

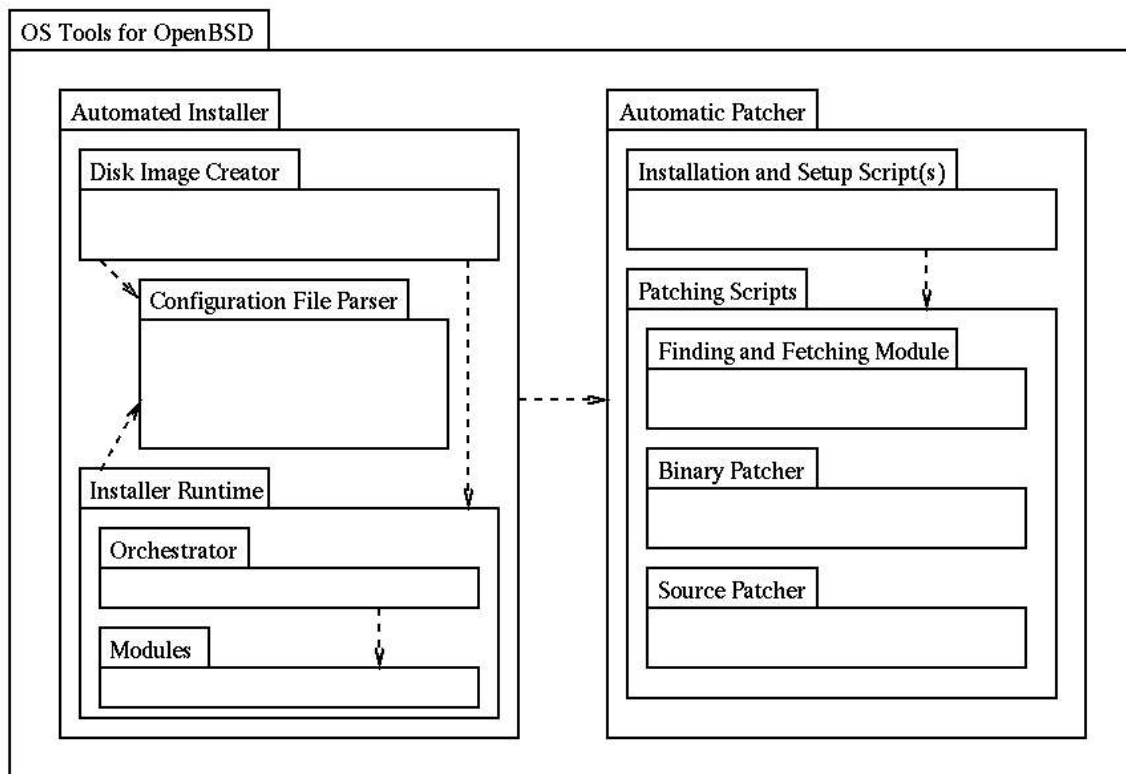
1. Introduction

On the 5th of January 2004 the United States Geological Survey (USGS) approached Northern Arizona University's (NAU) Capstone Project with an idea of developing OS tools for OpenBSD. Team FUGU (<http://www.cet.nau.edu/~fugu/>) was formed in order to develop this project. USGS is a world leader in the natural sciences through their scientific excellence and responsiveness to society's needs. The USGS Astrogeology Program uses OpenBSD due to its renowned security but the costs in time for installation and maintenance of the operating system is a big drawback.

On the 16th of January 2004 Team FUGU was chosen to develop an automated installer and an automated patcher for OpenBSD which when developed would alleviate the USGS of their problems. A Requirements document was accepted on the 18th of February, and a Functional Specification document was accepted on the 10th of March. This document provides a detailed Software Design Specification for the proposed software solution.

2. System Architecture Overview

This section contains a high-level description of our software product's architecture, by organizing the design into packages. A package diagram is shown below as *Figure 1*:



(Figure 1: Package Diagram)

The following subsections contain a description of each package in the above figure, in top-down order.

2.1. OS Tools for OpenBSD

We are designing and creating two *OS Tools* for our client to use with *OpenBSD*, an Automated Installer and an Automatic Patcher. While these tools are largely independent, they interact in that the installer will be able to apply security patches, getting the system “up to date” before it is booted into and system services are started.

2.2. Automated Installer

The *Automated Installer* is similar to RedHat Kickstart and Solaris Jumpstart in that after the configuration file and boot media are properly set up and put into place on the system, a full install will be made on bootup without any user input. Hard disk partitions are set up, hardware is set up, software packages are installed, services such as ssh and print daemons are set up, and any pre-install or post-install scripts specified in the configuration file are run at the appropriate time.

2.2.1. Disk Image Creator

At the core of the installer is the *Disk Image Creator*, which parses and validates the configuration file, and creates a bootable disk image for a given architecture (i386, Sparc, or Sparc64) that can be written to floppy, CD, or other boot media. It is likely that some features of the Disk Image Creator will only work on OpenBSD (such as building from source), but most will work on any type of UNIX workstation.

2.2.2. Configuration Parser

Whenever the validity of a configuration file needs to be tested by the *Disk Image Creator* then the *Configuration Parser* will be invoked to ensure it is in the correct format. Additionally the *Installer Runtime* will utilize the *Configuration Parser* to retrieve a value for a specified Section and Key. Any error messages will be displayed on STDERR.

2.2.3. Installer Runtime

On the boot media generated by the Disk Image Creator will be an *Installer Runtime* that we create, based on the code for the Interactive Install from the OpenBSD website. The Installer Runtime will use an *Orchestrator* script, that will determine which *Modules* are run, and in what order. The modules will include such things as setting up the network and setting up partitions, and anything else that can be done by the Interactive Install.

2.3. Automatic Patcher

The *Automatic Patcher*, our second tool, will have an Installation and Setup (maintenance) script, and a script which fetches and installs binary and source patches. The patching tool will be based on *Tepatche*, which provides part of the tools that our client needs, but the interface will be redesigned to suit our client's needs, and new features, especially binary patching, will be added.

2.3.1. Installation and Setup Script

The *Installation and Setup script* will install the patcher to disk, and add the *Patching Scripts* to the OpenBSD boot scripts and into *cron*, a task scheduling tool that comes with OpenBSD and most other UNIX systems. The frequency at which the tool shall be run will be specified in a configuration file in the installation directory or on the command line.

2.3.2. Patching Scripts

The *Patching Scripts* are comprised of three modules, a *Finding and Fetching Module*, a *Binary Patcher*, and a *Source Patcher*. The Patching Script can be invoked by the user at any time, and will be invoked at regular intervals by *cron*. The script can also be run on each system boot, and by the installer, ensuring that the system is no more than a day out of date at any given time

2.4. Runtime Order

The *Finding and Fetching Module* will be run first, and will find patches on the servers specified in the configuration file and will fetch only the ones which match the criteria specified in the configuration file. The Patching Script will run the Binary Patcher or Source Patcher, as appropriate, for each patch fetched and stored on the disk.

The *Binary Patcher* will apply the patch to a directory, and run a script provided with the patch.

The *Source Patcher* will unpack the patch to a temporary directory, set up configuration files, and run the appropriate make files to build and install the compiled source to the directory.

The *Patching Script* will update the information in the packaging system after the patch is complete. If a patch fails, an error will be sent to a log file and/or an email address, as specified in the configuration file.

3. Disk Image Builder

The Disk Image Builder subsystem provides a command-line tool that can build disk images and verify configuration files. Disk images can be made that include a configuration file or that.

3.1. Design Overview

The Disk Image Builder is a command-line tool that provides an interface to a build process for the automated install. The entities used in this process are shown in the following *Figure 2*:

```
Title:/home/ben/working/Disk_Image_Cla
Creator:Dia v0.92-pre7
CreationDate:Mon Mar 15 09:06:29 2004
```

(*Figure 2: Disk Image Builder Class Diagram*)

The disk image builder will work within a subdirectory of the OpenBSD source directory. This will allow the Automated and Interactive Installer makefiles to build the kernel and any needed programs (such as awk) if they haven't already been built. The Interactive Installer is in the "distrib" subdirectory, so for the Automated Installer we will choose something with a similar name (such as "autodistrib"). Within this directory will be directories for different types of media (such as CD or floppy). To start a build of a CD or floppy, the command-line utility will run "make" in the corresponding directory.

The Makefile will call upon Makefiles for the Automated and Interactive Installers to build them into the a common directory, in a way that the

produced files will not conflict with each other. The Interactive Install build scripts will build in the kernel and the majority of the tools that are needed for our program. The Installer Runtime scripts will be put in the appropriate parts of the RAM disk image and set to run upon boot. After the files have been built, a bootable disk image will be created from the files in the directory.

A command-line utility will provide an easy-to-use interface to create a build script in a given source tree and a man page will be provided to guide the user through the process. The user will be able to create an entire disk image or simply run a file through the parser and check for errors. Options will be provided for adding files to the image directory before the image is created (such as filesets). This script will likely be installed in /usr/local/sbin (the same place as Tpatche).

A detailed design of the command-line interface, modules, and state design follows.

3.2. Subsystem Designs

The basic designs of the subsystems are as follows:

3.2.1. Disk Image Command-Line Utility

The Disk Image Command-Line Utility is a Perl script that will read in options from the user, and call upon other entities to create the disk images and/or check the configuration files for accuracy.

The utility, by default will be installed into the user's path, and the utility's purpose and all of its options will be clearly documented in the accompanying man page. This will allow inexperienced system administrators to create an interactive install.

The command-line utility will need to be run as root if any building is to be done. An underprivileged user can only check a configuration file for correctness.

3.2.2. Disk Image Build Scripts

The Disk Image Build Scripts may be installed into any user defined directory but the directory `/usr/src/distrib/auto` will be provided as the default. The directory structure will closely mirror that of the Interactive Installer Build Scripts to provide uniformity. There will be a general build script in `/usr/src/distrib/auto`, and scripts for particular architectures and image types can be found within subdirectories.

Filenames for the distribution will be chosen in such a way that the Automated Installer and Interactive Installer, in built form, can peacefully co-exist in the same directory. This will allow the user to write a disk containing both installers, selectable immediately upon bootup.

The disk image build scripts will need to be run by a privileged user, because they are in `/usr/src`.

3.2.3. Automated Installer Source and Makefiles

The Automated Installer Source and Makefiles will be installed to the same directory where the Disk Image Build Scripts reside, `/usr/src/distrib/auto`. Makefiles will be provided that will cause any needed tools to be built that aren't built by the interactive installer script.

The design for the source code is described in the Installer Runtime section. This part only describes the way that this code will be fit into the process of creating installer disk images. Patches will be provided to modify the current Makefiles which are currently used to build the distributions for the various architectures, please see the next subsection for more information.

3.2.4. Interactive Installer Build Scripts

The Interactive Installer build scripts are part of the standard OpenBSD distribution, and will be left unchanged. They are used as an integral part of the Automated Installer because the two installers have many common dependencies and we determined that it would be better to start with the Interactive Installer build scripts than to start from scratch.

The Interactive Installer is part of the OpenBSD source distribution (src.tar.gz), and resides in /usr/src/distrib. Inside that directory are subdirectories for different architectures, and within those, subdirectories containing Makefiles for the CD and floppy images available from the OpenBSD ftp server. There are also more general Makefiles that cause the basic kernel and tools to be compiled and placed into a directory structure. It is these that we intend to make use of, rather than writing the Makefiles to build the dozens of command-line tools that are needed.

As shown in the class diagram in Figure 2, there are options to make the entire build and to clean the build. It is also possible to specify a directory in which to place the built distribution. We intend to use this, to get the Automated Installer and Interactive Installer into the same directory tree.

3.2.5. OpenBSD Source Tree

The Interactive and Automated Installer Build scripts rely on the OpenBSD source tree, which contains a number of makeable directories containing programs that ship with OpenBSD. If they have not already been made, they will be made when the object files are requested.

3.3. Program States

The state diagram for the Disk Image Builder is depicted below. The program takes the command-line options, which are the command, the files, and the options. The configuration file option applies to all three commands, while the disk image file and device file apply to the build command and the write command, respectively.

The commands are detailed in the next section.



Title: Disk_Image_States.fig
Creator: fig2dev Version 3.2 Patchlevel 4
CreationDate: Sat Mar 13 21:56:56 2004

(Figure 3: Disk Image Builder States)

3.4.Command-Line Interface

The command-line interface for the Disk Image Creator will utilize some standard commands as well as some general options applicable to all of the commands, these are described as follows:

3.4.1. Commands

Commands will be provided to *build* a disk image, *check* a configuration file for correctness, or *write* a floppy. All three options can take a configuration file as a parameter. All three will check the configuration file for accuracy, if given, and the build and write options will build the configuration file into the disk image.

bsdautoinst build [options] [conffile] outfile

bsdautoinst check [options] conffile

bsdautoinst write [options] [conffile] [device]

<i>Commands</i>	<i>Description</i>	<i>Options</i>
b build	Builds a disk image file and stores it in outfile, or in the build directory if option is not set.	-d -- default-action Sets the default action upon bootup to a given value.
c check	Runs the configuration file through the parser and reports errors.	
w write	Builds a disk image file and writes it to either a default floppy or a floppy specified in <i>device</i> .	-d -- default-action Sets the default action upon bootup to a given value.

3.4.2. General Options

General options that UNIX users have come to expect may be used in our command-line utility, under any of the three modes. These are as follows:

<i>Option</i>	<i>Description</i>
-h --help	Prints usage information.
-v --verbose	Will show more detailed information and warnings when checking the configuration file.

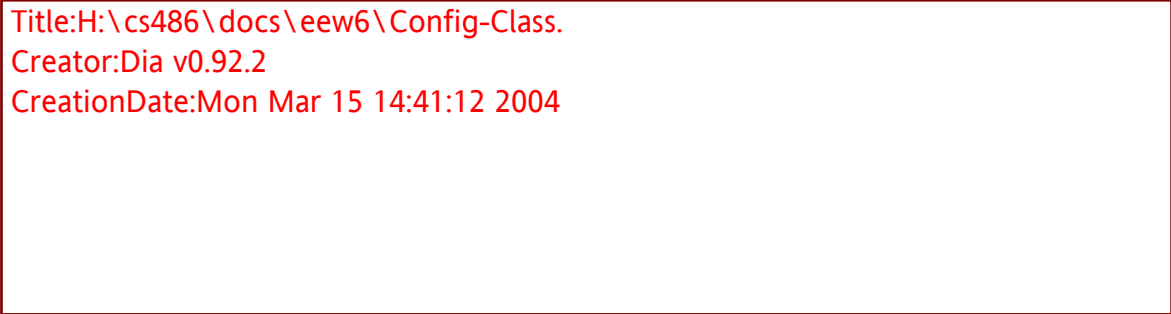
These options will give the utility a more polished command-line look and feel.

4. Configuration Parser

The configuration parser is a fairly simple element of the OS Tool set which will provide for an error checking mechanism when the administrator creates a configuration file. In addition it will be used by the install runtime to obtain a value given a section and key. The following two sections comprise the details of the configuration parser.

4.1. Design Overview

The following Figure 4 depicts the interactions needed with the Disk Image Creator and the Install Runtime which will be used:



Title:H:\cs486\docs\ew6\Config-Class.
Creator:Dia v0.92.2
CreationDate:Mon Mar 15 14:41:12 2004

(Figure 4: Config Parser Class Diagram)

As depicted in the above figure the parser will have functions specifically to parse sections and keys. The parser will be able to display a value for a given section and key to the Installation Runtime when both the Section and Key values correspond to those section and keys given as command line arguments. The next section depicts how these parsing functions are used to create various states.

4.2. Program States

The following Figure 5 describes the states used to create the configuration parser:



Title:config-parser-state.fig
Creator:fig2dev Version 3.2 Patchlevel 4
CreationDate:Mon Mar 15 15:42:34 2004

(Figure 5: Config Parser State Diagram)

As depicted to the left the main routines which encompass the various classes are also used to encapsulate the states of the configuration parser. The grammar will exist statically within the script so there is no need to load a file containing a grammar syntax.

Sample configuration files are still being developed but the next section describes in great detail the install process and information needed to complete the process, and this is where the file will be developed from.

5. Installer Runtime

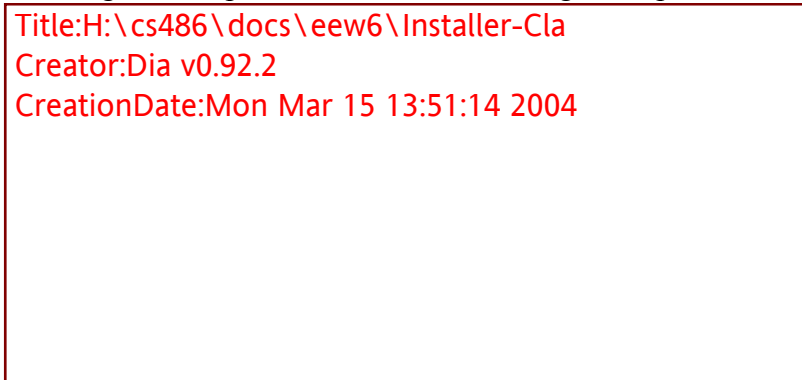
The Installer Runtime is the script that will be run upon boot, to perform an automated install of a system. The Automated Install works by reading options from a configuration file, to determine which action to take in each step, as opposed to asking the user, which is done by the standard Interactive Install.

The Automated Install is designed to allow a class of systems to be installed using only one configuration file. For this to be useful, it must exhibit a certain amount of polymorphic behavior. It will be possible to install a group of systems that are similar but not exactly alike using a single disk image.

The Automated Install closely mirrors the Interactive Install but will be able to determine settings on the fly and will have the capability of running pre- and post-install scripts.

5.1. Design Overview

The Installer Runtime installs an OpenBSD system using information provided in a configuration file, as well as decision trees carefully researched, implemented, and tested by our team, to guide the process. The modular design is depicted in the image below:



(Figure 6: Automated Installer Builder Class Diagram)

As depicted above the *dot.profile* script is used to determine whether to launch the *install.sh*, *upgrade.sh*, or *auto_inst.awk* script. The automated process will use the *config_parser.awk* script to access the values from the configuration files which it finds.

The Installer Runtime has a large number of dependencies. It depends on the kernel to properly recognize devices. It depends on the networking programs to help it to determine the proper network interface and get the network up. It also depends upon the partitioning and slicing tools. The building and adding of these things to the disk images will be taken care of by the disk image build scripts. For the Installer Runtime to work, it is important that the Disk Image Builder compile all the tools and place them in the right locations.

Other than that, creating the Installer Runtime is largely a matter of reading from the configuration file and determining what to do next. It is very well represented by a flow

chart, and as such, the subsystem design sections will contain many very brief function descriptions, and the flow of them will be depicted in the UML state diagrams.

5.2. Subsystem Designs

The subsystems will be designed as follows:

5.2.1. System Bootup

The system bootup is the same as the Interactive Installer's system bootup, with a slight modification: before detecting and loading the devices the installer needs to read in a string for the user, or time out and go with the default option.

5.2.2. Installer Script

The Installer Script calls upon all of the subroutines in the smaller modules, to complete the installation. It keeps track of some variables and passes parameters to the subroutines contained in the modules.

5.2.3. Pre-Install

The Pre-Install finds and loads the configuration file, detects fixed disks, backs up important data, and runs a pre-install script specified in the configuration file. Subroutines for this process are described here:

5.2.3.1. `getconf()`

The subroutine `getconf()` takes a string describing the location of the configuration of the configuration file as its input, brings the network online if necessary, and loads the configuration file into the mounted RAM disk. The configuration file string will usually be an absolute filename or an URL, but it can also be a network location based on the computer's IP address.

5.2.3.2. `backup()`

The subroutine `backup()` takes options describing files that need to be backed up, such as SSH keys, mounts disks, seeks them out, and backs them up to the RAM disk to be put on later.

5.2.3.3. `preinstall()`

The subroutine `preinstall()` reads the pre-install options from the configuration file, finds the script file(s), and executes them.

After the pre-install is complete, the fixed disk(s) are ready to be erased (if a clean install is being done).

5.2.4. Disk Partitioning and Formatting

Disk Partitioning and Formatting is a destructive process, but as part of the Automated Installer it is necessary that it do it without asking. There will be a warning in the disk image builder man page that booting the automated install can cause data loss. Backups will be suggested. The partitioning and formatting functions are as follows:

5.2.4.1. writembr()

The subroutine writembr() scans for available disks, reads the rules in the configuration file (for example “use only the primary disk” or “use all available SCSI disks”), and determines which disk to use. It then creates an OpenBSD partition filling the entire disk that will be sliced using a `bsd disklabel`. Splitting a single disk between two operating systems will not be an part of the automated installer, simply because it is difficult to get it right and it poses the risk of data loss.

5.2.4.2. writedl()

The subroutine writedl() takes the allocated MBR partitions as parameters, creates a disklabel file based on the configuration file, and writes it to the disk. The disklabel partitions' (or slices) sizes and locations are determined mathematically on the fly based on percent values, free space rules, and minimum and maximum sizes. Each disklabel partition is assigned a mount point as specified in the configuration file. The disklabel special files, filesystem types, mount points, and options are written to `/etc/fstab` and `/etc/mtab`.

5.2.4.3. formatslices()

The subroutine formatslices() formats all the slices based on the data in `/etc/fstab` and `/etc/mtab`, in order of their appearance in `/etc/fstab`.

5.2.4.4. mountslices()

The subroutine mountslices() mounts slices in preparation for package installation.

After the disks are partitioned and formatted, the filesets are ready to be retrieved and installed.

5.2.5. Basic Install

The basic install completes everything normally covered by the interactive install. This includes getting on the network and downloading and installing filesets, as well as writing configuration to disk and preparing for first bootup. Functions are briefly described here.

5.2.5.1. filesets()

The subroutine filesets() takes a location for filesets as its argument, temporarily brings the network online if needed, reads the names of needed filesets from the configuration file, and gets the filesets from the specified mounted directory or server. It saves these filesets to a partition for installation.

5.2.5.2. install()

The subroutine `install()` installs the filesets by unpacking them to the correct directories on the fixed disk.

5.2.5.3. restore()

The subroutine `restore()` restores the backed up system files, such as SSH keys, to their proper locations.

5.2.5.4. netconf()

The subroutine `netconf()` permanently sets up the network by reading network configuration settings from the configuration file and trying to connect to the servers. The network will only be brought up temporarily in this step. If the settings in the configuration file do not work, it will make a reasonable attempt to detect the proper network settings. The settings will then be written to disk.

5.2.5.5. finalconf()

The subroutine `finalconf()` reads the final configuration settings from the configuration file, such as temporary root password, time zone, keyboard settings, and locale, and writes them to the disk.

5.2.5.6. bootblock()

The subroutine `bootblock()` writes a boot block to the primary fixed disk, enabling the OpenBSD system to start up upon a reboot.

When the Basic Install is complete, the system is ready to boot into OpenBSD, and only the post-install and patching remains.

5.2.6. Post-Install

The post-install allows for the Automated Installer to do a few extra things that aren't part of a normal interactive install. This includes patching the system and running a post-install script.

5.2.6.1. netup()

Brings the network up, using the network settings written to the hard disk.

5.2.6.2. tepatche()

Creates a chroot environment, allowing perl to be run, and runs our version of `Tepatche`, the OpenBSD automatic patching system, inside of it. Patches the system and logs it.

5.2.6.3. postinstall()

Reads the postinstall options from the configuration file and runs the script(s) specified therein.

5.2.6.4. reboot()

Cleans /tmp and reboots into the installed system.

5.3. Program States

The following sections describe the states used to create an automated install runtime.

5.3.1. Overview and Preinstall

The program's states fit into four different groups, that build on each other, as depicted in Figure 7:

Title:Automated_Install.fig
Creator:fig2dev Version 3.2 Patchlevel 4
CreationDate:Sat Mar 13 16:52:12 2004

(Figure 8: Installer Runtime Pre-Install)

Title:Automated_Install_1.fig
Creator:fig2dev Version 3.2 Patchlevel 4
CreationDate:Sat Mar 13 16:52:23 2004

(Figure 7: Installer Runtime Overview)

The first stage is the pre-install, depicted in Figure 6. The steps involved are as follows:

5.3.1.1. Booting Up

In the Interactive Install, the system goes straight to the Init Scripts from the RAM disk image. For the Automated Installer, before this happens we will request that the user enter the location of a configuration file, and indicate that if one is not entered the default will be chosen in five seconds.

5.3.1.2. Opening the Configuration File

If the action selected is a to run an automatic install on a configuration file, a brief warning will be displayed, saying that an automatic install is selected, and any existing operating systems will be written, and that the installation can be canceled at

any time by pressing Ctrl-C. The configuration file will then be opened, according to “bootstring”. If it is set to be loaded from the network, a temporary connection will be established. The network connection process will closely mirror that in the interactive install.

5.3.1.3. Backup System Data

System data, such as SSH keys, will be backed up, according to the configuration file. This feature was specifically requested by our client.

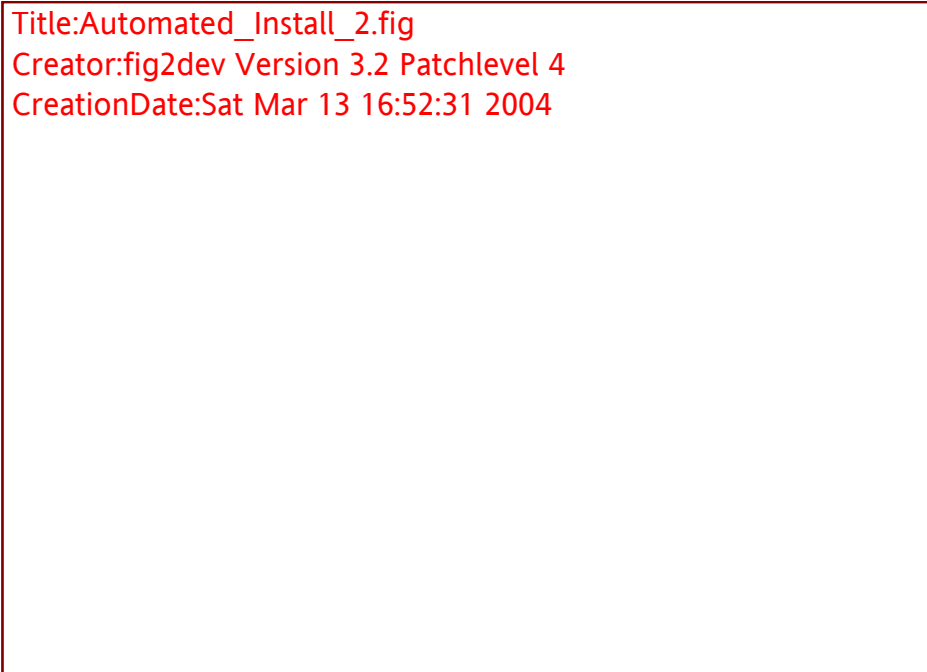
5.3.1.4. Pre-install

Any specified pre-install scripts will be run in the RAM disk environment.

After this is complete, the disk is ready to be written. A message will be displayed saying, “The disk will now be written.”

5.3.2. Disk Setup

In the following Figure 9, the disk geometry is determined and the disks are partitioned and formatted.



(Figure 9: Installer Runtime Disk Setup)

The disk setup steps are as follows:

5.3.2.1. Determine Disk Geometry and Create MBR Partitions

The program will determine disk geometry using the appropriate system administration functions that ship with OpenBSD. It will then write OpenBSD MBR partitions on any disk specified in the configuration file. On all systems, including i386 systems, the installer will assume that the entire disk will be used. This is

because the ability to multi-boot is not desired by our client, and because it is not trivial to implement.

5.3.2.2. Determine Disk Slicing and Create Disklabel Partitions

Disk Slicing will be determined based on rules read from the configuration file. A disklabel file will be written to the MBR partition, and an fstab will be written to the RAM disk.

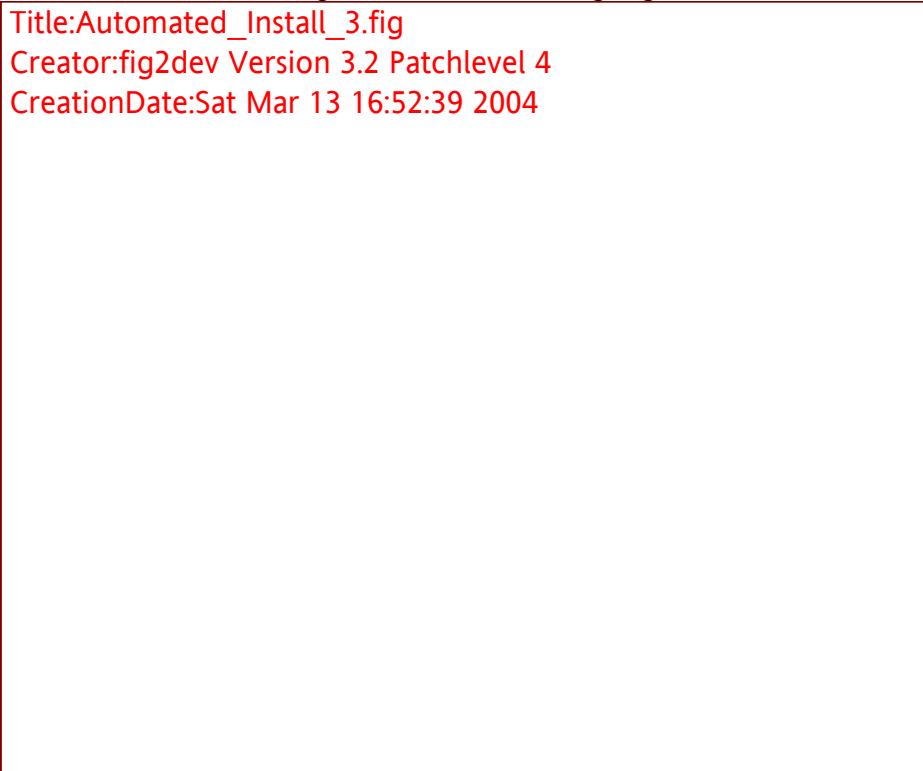
5.3.2.3. Format Slices and Mount Filesystems

The slices will be formatted and the corresponding filesystems will be mounted according to the information in fstab.

After this is done, the basic parts of the permanent installation are ready to be completed.

5.3.3. Basic Install from Configuration

The basic installation is completed in the following Figure 10.



(Figure 10: Installer Runtime Basic Install)

The basic setup steps are as follows:

5.3.3.1. Install Filesets

OpenBSD filesets (the basic installation available from the OpenBSD ftp site) will be installed from the specified location(s). The network will be set up temporarily, if necessary.

5.3.3.2. Restore Backed Up Data

Data backed up during the preinstall will be permanently written to the fixed disk.

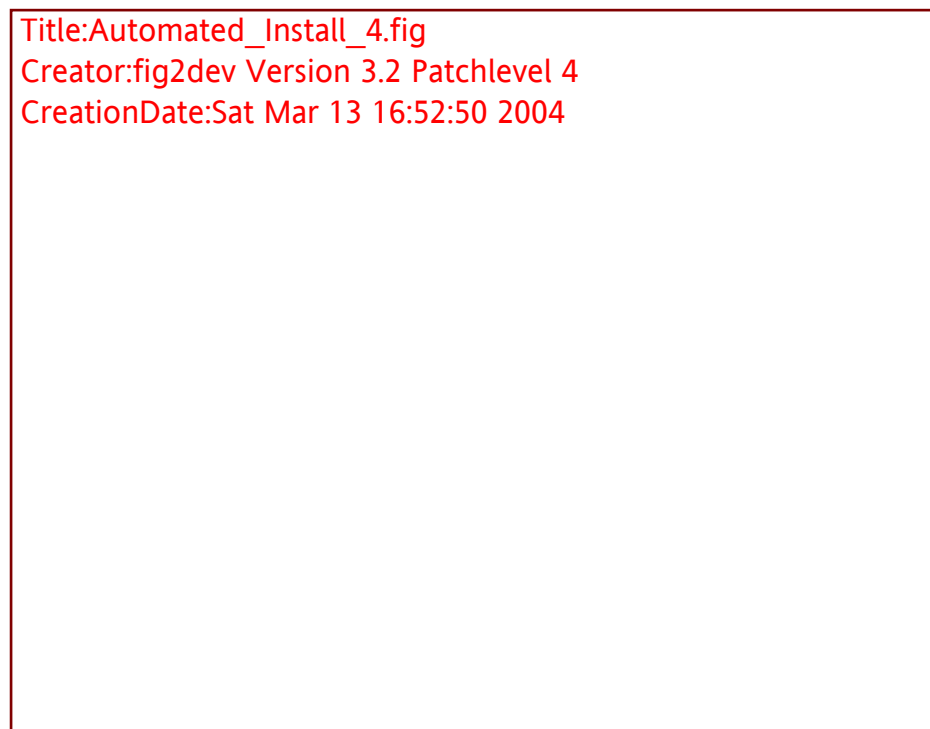
5.3.3.3. Write Basic Configuration to Disk

Settings determined in previous steps, and other settings from the configuration file, will be written to disk. A boot block will also be written, enabling the system to be booted into the newly set up OpenBSD environment for the first time.

After this is done, the system can be booted into OpenBSD or run in a chroot environment.

5.3.4. Finalization and Post-Install

As depicted in the following Figure 11, Any tasks selected for the post-install are completed. This includes installing the latest security patches, using our version of the OpenBSD automatic patcher, Tepoche.



(Figure 11: Installer Runtime Post-Install)

The final steps are as follows:

5.3.4.1. Start Network

A network connection will be started, using the permanent network settings, if it is needed.

5.3.4.2. Install Tepoche

Tepatche will be installed and run, if the option is chosen in the configuration file. This process includes setting up a chroot environment.

5.3.4.3. Run Post-Install

The post-installation specified in the configuration file will be run. By default this will be from the RAM disk environment, but it is desired to provide the option in the configuration file to run in the chroot environment.

5.3.4.4. Reboot System

The system will either automatically reboot or instruct the user to remove boot media and press a key to reboot. This setting is part of the configuration file.

The Installer Runtime shall provide a simple but well-designed unidirectional user interface. If the user is sitting in front of the computer, he or she will see helpful messages about what the installer is doing. Additionally, these error messages can be logged to a file to be read later or emailed to the system administrator during the post-install.

While it is desired that the Installer Runtime not require any input from the user, or ask any questions, except for a configuration file upon system bootup, the installer shall be designed so that it can be interrupted at any time by pressing Ctrl-C. While this situation is unlikely, it is important in to plan for it anyway, in making a robust user interface.

6. Automated Patcher

The automated patcher will be created by revamping and then adding additional features onto the Tepatche Perl script. The script will be divided into several modules to better organize the operations and then binary patching capabilities will be added through the use of the current *pkg* (package) facilities. The following sections provide greater detail on the design.

6.1. Design Overview

The following Figure 12 depicts the overall interactions within Tepoche:

Title:H:\cs486\docs\ew6\Tepoche-Clas
Creator:Dia v0.92.2
CreationDate:Mon Mar 15 15:46:20 2004

(Figure 12: Tepoche Class Diagram)

As depicted above Tepoche is divided into three distinct parts: a Binary module, a Source modules, and the Main module. Several important functions are depicted in the previous classes:

- The ability to download source tarballs if they are needed.
- The ability to download source or binary patches as needed.
- The application to the system of those patches.
- The ability to create a binary patch given that the source is patched.

Tepoche will be installed during the post-install phase listed in Section 5.2.6 and will be run as root during a periodic cron job. A configuration file will also be used for Tepoche in which the script will determine which patching type it will perform (binary or source), if it should host binary patches, as well as various other configuration options such as logging level and email address to send messages to.

The following sections describe in greater detail the process states which will be used by Tepoche in order to create an automated patching solution.

6.2. Program States

The following Figure 13 describes the interaction between the different Tepoche modules:



(Figure 14: Tepoche Activity Diagram)

As denoted in the above figure the three Tepoche modules interact with each other to provide the complete interaction needed for an automated patching system. The activities should be well distributed between the various modules to allow for a well balanced distribution of functionality between the modules. The individual operations of the Main, Source, and Binary modules are provided in the following subsections.

6.2.1. Main Tpatche Module

The Main module will provide the interaction from the shell and orchestrates all of the resulting actions as determined by the configuration file. It follows then that the Main module will perform the following steps during the patching process:

1. Read configuration information from file.
2. Call as appropriate either the Source module or Binary module to perform patching.
3. Send appropriate messages and errors to location defined in configuration file.
4. Schedule a reboot if requested to do so in the configuration file.

The following sections describe in more detail those process which would occur with the Source and Binary modules.

6.2.2. Source Patching Module

The Source module will be used to compile from source any patches which exist from the OpenBSD website, but additionally may act as a server by creating Binary patches from the compiled source. This allows a single source patching computer of a specific architecture to distribute the resulting compilation to the other computers of the same architecture to the rest of the network. The steps needed to perform this are as follows:

1. Download the source tarballs and deflate if source does not exist on system.
2. Download patches from the OpenBSD web server, or other defined location.
3. Apply the downloaded patches to the source.
4. Compile the source and perform any other operations needed to the compiled product (such as copying a newly created Kernel to allow boot).
5. Create a Binary patch if desired from the compiled source patches.

The following section describes the processes used for binary patching in greater detail.

6.2.3. Binary Patching Module

The Binary module is useful for avoiding the slow compiling speeds associated with source patching. The following actions encompass the Binary module:

1. Download binary modules of a specified architecture hosted on network.
2. Apply those binary patches using the *pkg* facility.

Or alternatively the Binary module can create binary patches if the compiled source exists on the system.