OPAALS PROJECT

Contract n° IST-034824

Task 10.13:

agropedia Appliance (IITK)

Deliverable 10.15

A KM appliance – for deployment in an agricultural knowledge management setting

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number: IST-034824

Project Acronym: OPAALS

Deliverable N°: D10.15

Due date: M30

Delivery Date: M33

Short Description:

A Software appliance is a device that provide a narrow range of functions that are normally run on a dedicated hardware platform. The objective of this task is to build a storage appliance accessible over http.Essentially it is a content management system enhanced with a knowledge model for organizing, browsing and searching the content according to a knowledge model. The knowledge model itself is a component that can be easily plugged in and enhanced in "operation". It also has the necessary infrastructure for collaborative enhancement and sharing of knowledge. We would like to design, build and test-deploy such a system in the domain of agriculture.

Author: Mitesh Gupta

Partners contributed:

Made available to:

Versioning		
Version	**Date**	**Name, organization**
0.1	26-12-2008	Mitesh Gupta, IIT Kanpur
1.0	21-2-2009	Mitesh Guta, IIT Kanpur
1.0	22-2-2009	T.V.Prabhakar, IIT Kanpur

Quality check

Internal Reviewers: Jesus Gabaldon, Paolo Dini

Dependences:

Achievements*	We have successfully built a KM appliance, which contains a self customised Linux, able to run agropedia website on top of drupal content management system. We have used asus eee pc as the hardware base for our appliance. The appliance built will be deployed in the Krashi Vigyan Kendra's (KVK), in India. The appliance when booted up can be used as a server and other computer's can connect to it through wireless and wired media.
Work Packages	An appliance could also be the way an OKS node is deployed and hence connected to WP10. The agropedia appliance is expected to be installed in Village Knowledge Centers and hence generate case studies and data for WP11.
Partners	The OKS Team
Domains	Computer Science Domain: This report covers knowledge on building a computer appliance which contains a custom self build Linux system. It contains a general methodology which is to be followed by which we can build a computer appliance. A special case of building an agropedia appliance is discussed in detail.
Targets	Village Level Krishi Vigyan Kendras (Agriculture Knowledge Centers)
Publications*	--
PhD Students*	No PhD student. One Master's Student – Mitesh Gupta
Outstanding features*	The appliance built is very specific to the task and contains no overhead other than the function it has to perform. The Operating System is custom tailored and is very specific to the hardware it is being built for - in this case an Asus Eee pc. The system build is minimum in size and maximum in performance.
Disciplinary domains of authors*	Mitesh Gupta, T.V.Prabhakar, Computer Science

The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report

Table of Contents

Table of Contents 4
1. Introduction 6
1.1 Introduction to Agropedia 6
2. Requirements of an Appliance 7
3. Approach 7
3.1 The Top-down Approach to build a computer appliance 7
3.2 The Bottom-up Approach to build a computer appliance 7
4. Choosing the C library 8
5. Cross-Compilation 8
5.1 What is cross-compiling? 8
5.2 Building a cross platform development tool chain 9
5.2.1 Manually building a toolchain 9
5.2.2 Automated cross-toolchain build systems 10
6. Linux Kernel 10
6.1 Configuring the Kernel 10
6.2 Compiling the Kernel 11
7. Building the file system 11
7.1 Installing Libraries 12
7.2 Installing Kernel Modules and Kernel Image 12
7.3 Device Files 12
8. Main System Applications 12
8.1 BusyBox Configuration 13
8.2 Installing BusyBox 13
8.3 Custom Applications 13
9. Building a KM Appliance 14
9.1 Background Knowledge and other requirements 14
9.2 How we built a Cross-Compiling toolchain? 15
9.3 Building the Linux kernel 16
9.4 Busy box Installation 16
9.5 Making the root file system 17
9.6 Installing the HTTP server 18
9.7 Installing the Database Server 18
9.8 Installing PHP 19
9.9 Other Necessary files 20

9.9.1 Boot scripts 20
9.9.2 mdev.conf 20
9.9.3 /etc/profile 21
9.9.4 /etc/inittab 22
9.9.5 Configuring the network card 22
9.9.6 Device File 23
9.9.7 Finishing up 24
9.10 Booting up the System 24
10. Performance Tuning 24
10.1 Changing Kernel Parameters 24
10.2 Tuning Process Priority 25
10.3 Tuning Virtual Memory Subsystem 25
10.4 Tuning the disk subsystem 25
10.5 Tuning the network subsystem 26
10.5.1 Increasing network buffers 26
10.5.2 Tuning window sizes 26
10.5.3 Some recommendations [6] 26
11. Available tools 27
12. Summary 28
13. Conclusion 28
14. Resources 28

1. Introduction

An **appliance** refers to a device with a narrow function. A device or control that is very useful for a particular job. A **computer appliance** is generally a separate and discrete hardware component specifically designed to provide a specific functional resource, and which often resides on a dedicated hardware platform. Examples can be http server appliance which is a computer that works as an http server and nothing else. Other example can be a search appliance designed for indexing corporate websites and returning these results.

In this work we try to explore methodologies to build a computer appliance. In section 9 of this report we look at agropedia and the requirements of an agropedia (www.agropedia.net) appliance and build one. The approach discussed here is general and can be used to build any custom tailored computer appliance.

OPAALS and Agropedia are both open and open source projects with no Intellectual property rights concerns. There is very good synergies between them that allows the OPAALS funding to go much farther and interact with a broader community of research and adoption.

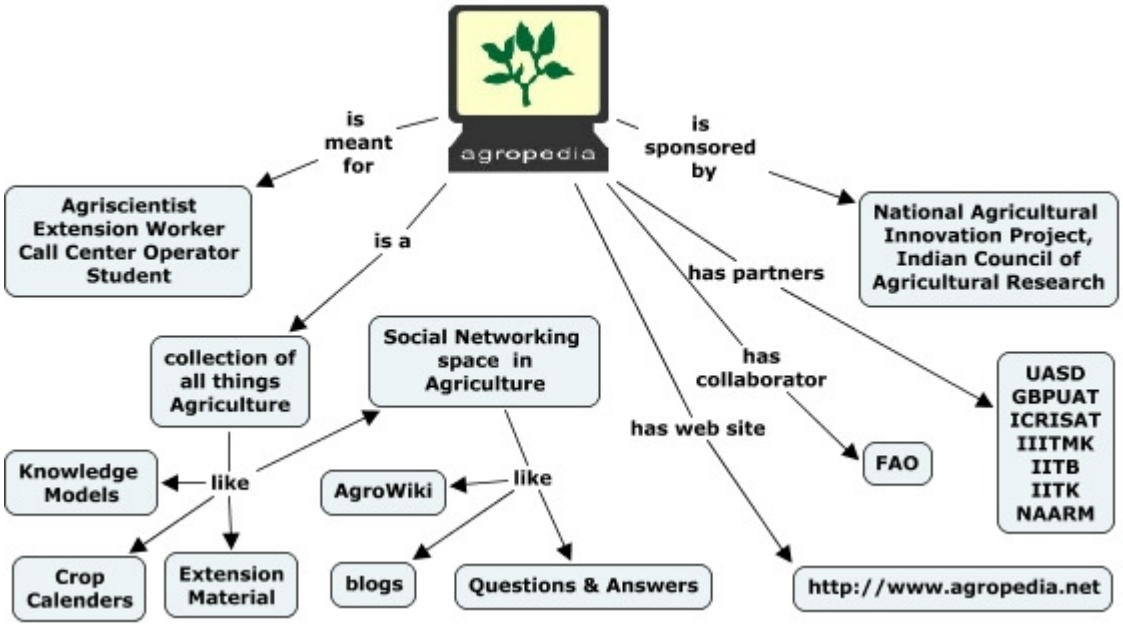
In the following section we provide a detailed introduction about Agropedia.

1.1 Introduction to Agropedia

Agropedia is an agriculture knowledge repository of universal meta models and localized content for a variety of users with appropriate interfaces built in collaborative mode in multiple languages. Agropedia aims to develop a comprehensive digital content framework, platform, and tools in support of agricultural extension and outreach. In other words, it aspires to be a one stop shop for any information, pedagogic or practical knowledge related to extension services in Indian agriculture – an audiovisual encyclopedia, to enchant, educate and transform the process of digital content creation and organization completely.

Using state of the art practices and techniques of the semantic web, agropedia is a platform where both specialists in the agriculture research and education domain and students and others interested in agriculture can make lasting contributions to the vast knowledge base. The specialists have a choice to contribute towards the gyan dhara (certified content) or participate in the interaction space to contribute to janagyan (emergent knowledge). All other registered users are co creators of janagyan (emergent knowledge) through their participation in the agrowiki, agro-blog, agro-forum and agro-chat like interaction spaces. Thus, the users of agropedia are the architects of the knowledge, which is the lifeblood of agropedia, and they do this through an easy to use, entertaining and intellectually stimulating web interface.

Agropedia project is funded by the World Bank and had been developed by the Consortium for 'Redesigning the farmer extension agricultural research/education continuum in India with ICT mediated Knowledge Management' with the NAIP-KM team of IIT Kanpur building the technology platform. The motive of the project is to develop highly integrated approaches between agricultural research and the education sector with the Krishi Vigyan Kendra (KVK), the emerging actors in private sector and with the organisations promoting rural information access centre. The following concept map decribes agropedia in greater details: -



CMap describing Agropedia (Courtesy www.agropedia.net)

2. Requirements of an Appliance

The requirements for the computer appliance can be listed down as:-

1. Should provide a desired functionality.
2. Should not have any unnecessary functionality causing an overhead in the system.
3. Should be fine tuned and optimised to deliver the required functionality
4. Should be as small as a foot print as possible in terms of size of the final system.

3. Approach

Broadly speaking there are two approaches one can follow to build a computer appliance. They are:

1. Top-down approach.
2. Bottom-up approach.

3.1 The Top-down Approach to build a computer appliance

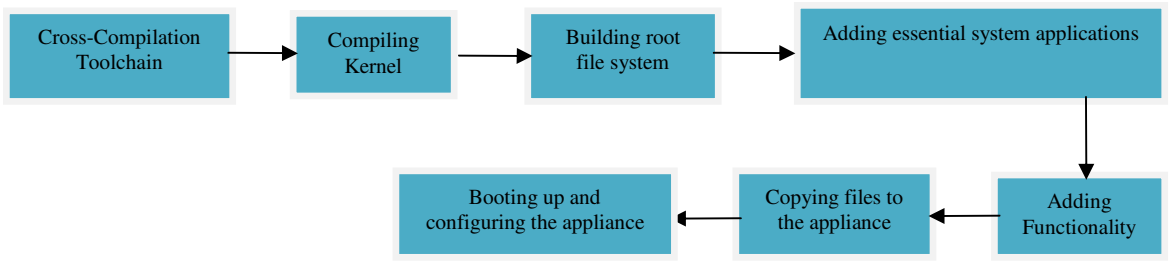
In this approach, one starts with a complete desktop GNU/Linux distribution (Debian, Fedora, Ubuntu etc.) and removes unneeded stuff. The advantage of this approach is that one can start with an available distribution; there on, one can start by first adding the functionality that is required and then remove unnecessary stuff. The disadvantage of this approach is, that this is a very tedious job - one has to go through a huge number of files and packages. There is a need to understand what each file and package is about before removing it. Even then one might end up with a not so small a system as most distributions use standard desktop toolsets and libraries and lots of shared libraries end up in the final configuration.

Section 11 gives a list of some of the tools available to customise a standard distribution.

3.2 The Bottom-up Approach to build a computer appliance

This approach expects the user has a complete understanding of the system he is building and gives him full control over the system. The user starts with an empty or minimalist root file system, incrementally adding only the things that are need. This approach is much easier to control and maintain and the end result of this approach can be extremely small, all the more as one may use lightweight toolsets instead of standard toolsets. While the details of the procedure necessarily vary from one target to another, the same general principles apply. The steps described in the later sections can be used to build a customised appliance using this approach. Typically one could achieve savings in three main areas- the kernel, the application programs and the libraries.

The following figure shows the workflow to build an appliance



Steps in Building an Appliance

First we start by building a cross-compilation *toolchain*, more of which is explained in the following sections, then we build a root filesystem which will later be copied to the appliance, The Linux kernel is cross-compiled depending upon the appliance hardware; an essential system application busy box which contains all the required system applications is compiled and installed in the root filesystem. Then all the required functionality is added one by one to the system and the complete filesystem is then copied to the appliance making it ready to be used.

For building a cross-compilation toolchain we have to choose a C library against which all our applications will be compiled. We will discuss the aspects to be kept in mind while choosing the C library in the following section.

4. Choosing the C library

The GNU C library which is used almost in general is quite large in size and occupies much space, if used, in the appliance. uClibc is a good alternate to the GNU C library. The uClibc library originates from the uClinux project, which provides a Linux that runs on processors lacking a memory management unit (MMU); the library, however, has since become a project of its own and supports a number of processors, including one's that have an MMU. Although it does not rely on the GNU C library, uClibc provides most of the same functionality. uClibc is available for download from the project website at http://www.uclibc.org. We have used uClibc library in building the agropedia appliance.

5. Cross-Compilation

5.1 What is cross-compiling?

"*Cross-compiling refers to using a compiler on one system to develop code to run on another. The system on which a target system is build is typically called the host and for which the system is being developed is called target. Even when host and target are the same architecture, it is*

necessary to distinguish between their compilers, they may have different versions of libraries, or libraries built with different compiler options, so that something compiled on host compiler could fail to run, or could behave unexpectedly, on the target. [3]" To build an Appliance we have to make a cross-compiler as we will be developing our system to run on the target machine which may be different from host in terms of architecture.

5.2 Building a cross platform development tool chain

"*A tool chain is a set of software tools needed to build computer software. A cross-platform tool chain is built on one development platform but build programs that run on another platform* [1]" We present a build overview for building the tool chain. As mentioned in [1], the five main steps involve setting up:

1. Linux headers
2. Binary utilities (binutils)
3. The bootstrap compiler
4. The C library
5. The full compiler.

The first thing that one probably notice is that the compiler seems to be built twice which is normal and required, as some languages supported by GCC, such as C++, require C library support. Hence, a bootstrap compiler is built with support for C only, and a full compiler is built once the C library is available.

Each of the steps involves many iterations of its own. Nonetheless, the steps remain similar in several ways. One may follow the following general steps in order to build a toolchain: -

1. Unpack the package.
2. Configure the package for cross-platform development.
3. Build the package.
4. Install the package.

It is not necessary that all packages will follow all the steps mentioned above, for example the Linux headers do not require one to build or install the kernel.

Building a tool chain can be very error prone and is a delicate and complicated process. One has to take care about versions, patches, and tweaks of the various tool chain components for various architectures – knowledge about which is not only scattered at various different locations, but also changes from one version to another.

5.2.1 Manually building a toolchain

To build the toolchain manually one can follow http://cross-lfs.org/view/clfs-embedded/x86/cross-tools/introduction.html as reference for the target x86 cross-compilations. To manually build a toolchain for other architectures one has to explore and find the various parameters needed to be changed in other to develop a cross-toolchain. Manually building a cross-toolchain can be very error prone and takes a lot of time. To simplify the things there are various automated tools available which will build the cross-toolchain for the architectures supported by the tools. We will now explore the automated cross-toolchain in the next section.

5.2.2 Automated cross-toolchain build systems

A number of commercial sources for working cross-compilers for various architecture combinations are available, as well as several free cross-compilation toolkits. We list out a couple of cross-compilation toolkits that can be used, and there may be many more:-

1. Crosstool-ng: Dan Kegel's crosstool collects a variety of expertise and a few specialized patches to automatically build toolchains for a number of systems. Crosstool has not been updated in a while, but the new crosstool-ng project builds on this work. Can be downloaded from the distribution site http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool.
2. Buildroot: Buildroot is a set of Makefiles and patches that makes it easy generate a cross-compilation toolchain and root filesystem for our target Linux system using the uClibc C library. Buildroot has recently grown into a large project and it is possible to construct a complete linux system with user defined functionality using buildroot.
3. Scratchbox: Scratchbox is a cross-compilation toolkit designed to make embedded Linux application development easier. It also provides a full set of tools to integrate and cross-compile an entire Linux distribution.

We have used crosstool-ng open source cross-compiling toolchain for building the Appliance.

After setting up the cross-compilation tool one can start building up the system. We will begin by cross-compiling the Linux kernel for our target system, and then will build/install an application program named Busybox [8], and finally the root file system. This would give us the minimal system which is ready to boot up and bring up the shell. Later we can add the functionality to the appliance as desired for the purpose.

6. Linux Kernel

To start building the Linux kernel one need to get the copy of the latest stable Linux kernel source downloadable from http://www.kernel.org. To build and install a Linux kernel one first needs to configure the kernel for the target.

6.1 Configuring the Kernel

Configuration is the initial step in the build of a kernel for the target. This is an important step and should be done with care as one has to choose all the required kernel functionality in this step like choosing the correct network card drivers, filesystem type to be supported, hard disk drivers etc. There are many ways to configure the Linux kernel, and there are many options from which to choose. Regardless of the configuration method one uses or the actual configuration options one chooses, the kernel will generate a *.config* file at the end of the configuration and will generate a number of symbolic links and file headers that will be used by the rest of the build process.

The Linux kernel build system (Kbuild) includes support for a variety of configuration methods, including the following:

`$make config`
Provides a command-line interface where one is asked about each option one by one. If a *.config* configuration file is already present, it uses that file to set the default values of the options it asks one to set.

$make oldconfig
Feeds *config* with an existing *.config* configuration file and prompts one to configure only those options one have not previously configured. This contrasts with *make config*, which asks one about all options, even those one have previously configured. Developers often use this option to update their configuration as upstream configuration options change, without having to reconfigure the entire kernel.

$make menuconfig
Displays a curses-based terminal configuration menu. If a *.config* file is present, it uses it to set default values, as with *make config*.

$make xconfig
Displays a Tk-based X Window configuration menu. If a *.config* file is present, it uses it to set default values, as with *make config* and *make menuconfig*.

6.2 Compiling the Kernel

Compiling the kernel involves a number of steps. These include building the kernel image and building the kernel modules. Depending upon the chosen architecture, one might also need to specify what kind of image needs to be produced. For example, in the case of an ARM platform, one could use the following command to create a compressed image:

```
$make ARCH=arm CROSS_COMPILE=arm-linux- zImage
```

With the kernel image built, next step is to build the kernel modules:

```
$make ARCH=arm CROSS_COMPILE=arm-linux- modules
```

To install the Linux kernel modules in an alternate directory, use this command:

```
$make ARCH=arm CROSS_COMPILE=arm-linux-
INSTALL_MOD_PATH={Alternate Directory} modules_install
```

With both the kernel image and the kernel modules built they just have to be copied to the target to be used.

7. Building the file system

One of the last operations conducted by the Linux kernel during system start-up is mounting the root filesystem. We will discuss here the basic root filesystem structure. Then we will explain how and where to install the system libraries, the kernel modules, kernel images, device nodes, main system applications and custom applications. Finally, we will discuss how to configure the system initialization scripts.

The rules to build a root filesystem are contained in the Filesystem Hierarchy Standard (FHS) document. Wikipedia provides a good description about FHS describing the essential directories which can be read at http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

Follow the following instructions in order to create a basic root filesystem skeleton.

1. **$mkdir rootfs (Create a directory name rootfs.)**
2. **$cd rootfs**
3. **$mkdir bin dev etc lib proc sbin sys tmp usr var**

```
4. $chmod 1777 tmp
5. $mkdir usr/bin usr/lib usr/sbin
6. $mkdir var/lib var/lock var/log var/run var/tmp
7. $chmod 1777 var/tmp
```

After creation of a root filesystem skeleton various software components should be placed in their appropriate locations.

7.1 Installing Libraries

While building the tool chain we build a C library that will need to be copied on the target's root file system that has been created so that the applications can use them at runtime. The directory where one has previously installed the C library contains the entire list of libraries installed during the package's build process. One can selectively copy only selected files which the application needs, can be found out by ldd command, or the entire directory to the root file system rootfs/lib directory created above.

7.2 Installing Kernel Modules and Kernel Image

Once the build kernel modules and kernel image has been built, one can copy them to the root filesystem. The kernel image is to be copied into the rootfs/boot directory and the modules to the rootfs/ directory.

7.3 Device Files

All the device files in a Linux root file system are located in the /dev directory. The device file can be built by using mknod utility. The figure below lists the essential entries one needs in the /dev directory:-

Filename	Description	Type	Major number	Minor number	Permission bits
Null	Null device	Char	1	3	666
Console	System console	Char	5	1	600
Mem	Physical memory access	Char	1	1	600
tty0	Current virtual console	Char	4	0	600
tty1	First virtual console	Char	4	1	600
tty	Current tty device	Char	5	0	666

To create a device file using mknode issue the following command:-

```
$mknod –m 0600 console c 5 1
```

The above statement will create the 2nd entry of the table described above.

8. Main System Applications

Once the file system is created and the kernel, system libraries are installed on it, one can start to build the application. We will describe here the busybox application which is a multi-call binary that combines many common UNIX utilities into a single executable. One will also save a lot of time and find it easier to implement a simple system, as one doesn't have to configure and build the sources of each tool separately. One can obtain the latest source code of the Busybox application from the project website http://www.busybox.net. One has to extract the package and move into the

extracted directory to install the package and follow the steps as mentioned in the INSTALL file which comes along with the source code of the BusyBox.

8.1 BusyBox Configuration

As mentioned in INSTALL file, one starts by configuring the busybox for various tools to include in the system. BusyBox uses exactly the same configuration tools as the Linux 2.6 kernel. Likewise, all configuration settings are stored in a *.config* file in the root source directory, which can be created with the same configuration commands $ make xconfig or $ make gconfig or $make menuconfig or $ make defconfig, which a user can choose according to his needs. A short description about each configuration option is provided in section 6.1. BusyBox can be configured with only the set of applets needed for each environment.

8.2 Installing BusyBox

To build BusyBox with uClibc instead of the GNU C library, use the following command:

```
$make ARCH=ppc CROSS_COMPILE=i586-geode-linux-uclibc- \
> CONFIG_PREFIX=rootfs install
```

BusyBox will now be installed in the rootfs directory of the system and is ready to be used.

Another important utility that can be installed on the system is Tinylogin which is also a single binary like BusyBox which implements addgroup, adduser, delgroup, deluser, getty, login, passwd, su, sulogin and vlock.

8.3 Custom Applications

This section contains some guide line which a user can use to install other applications according to his needs on the target system.

Many software packages for Linux come as compressed archives of source files. The same package may be "built" to run on different target machines. A single distribution of a software package may thus end up running, in various incarnations, on an Intel box, a DEC Alpha, a RISC workstation, or even a mainframe. One has to carefully compile with appropriate build options given at configuration or compile time and install the packages in order to do so.

Listed below is a general procedure (as mentioned at http://www.tldp.org/HOWTO/Software-Building-HOWTO-3.html) one can use to follow to install the desired application on the target: -

- Obtain the source of the application and extract it in a directory.
- Read the README/INSTALL file and other applicable docs included in the source. Particularly keep focus on instructions to cross-compile the application.
- Run configure script with the necessary flags set for cross-compilation, host and target system.
- Check the makefile.
- If necessary, run **`make clean`**, **`make Makefiles`**, **`make includes`**, and **`make depend`**.
- Run **`make`**.
- Check files permissions.
- If necessary, run **`make install`**.

Note:-
After the **`make`** creates the binaries, one may wish to **`strip`** them. The **`strip`** command removes the symbolic debugging information from the binaries, and reduces their size, often drastically.

9. Building a KM Appliance

Using the knowledge described above we were able to successfully build an agropedia Appliance which can be readily deployed in an Agriculture Knowledge Management setting. For the appliance hardware we have chosen Asus EeeBox (http://www.asus.com/news_show.aspx?id=11854). Asus EeeBox comes with the following specification: -

CPU: - Intel Atom N270 (1.6 GHz)
RAM: - 1 GB
HDD: - 80 GB
Wi-Fi: - 802.11n
LAN: - 10/100/1000

Task was to build an agropedia appliance which can successfully run the www.agropedia.net website with Maximum Performance. The appliance built can act as server and can be deployed in the remote location where no Internet is Available. Users can connect to the appliance directly through Wi-Fi or through wired cable and access the www.agropedia.net website.

9.1 Background Knowledge and other requirements

Agropedia server runs on acquia Drupal (http://acquia.com). Acquia is a "commercial open source software company" providing products, services, and technical support for the open source Drupal (www.drupal.org) social publishing system. Requirements for running acquia Drupal are: -

1. HTTP Server
2. PHP
3. Database Server

In our appliance we have focused on providing the above requirements and have installed/added only the above mentioned packages thus keeping the functionality limited and very basic.

For building the agropedia appliance following packages were used: -

1. **HTTP Server**: - Light HTTP server lighttpd (http://www.lighttpd.net/) was used. Lighttpd server gives a small memory footprint compared to other web-servers, effective management of the cpu-load, and advanced feature set (FastCGI, SCGI, Auth, Output-Compression, URL-Rewriting and many more) making it an ideal choice to be used in a appliance.

2. **PHP**: - Php (http://www.php.net/) release version 5.2.8 was used and built with the process described in the following section.

3. **Database Server**: - agropedia databases are stored in MySql (http://www.mysql.com/) database. Although other database choices exist with us were PosgreSql (http://www.postgresql.org/), Sqlite (http://www.sqlite.org/ which would be best in an

appliance due to its small memory footprint), we have chosen MySql database server to be compatible with agropedia.

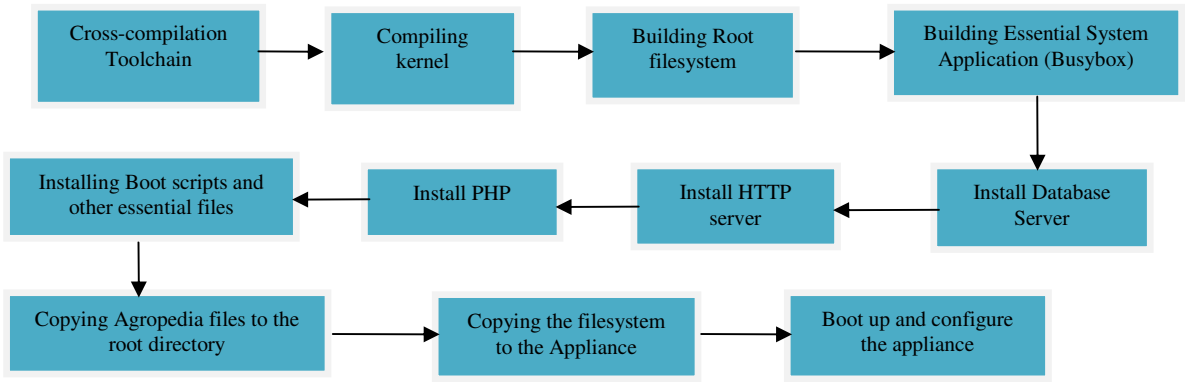
4. **System Utilities**: - To provide the minimum necessary utilities like ls, vi, ping etc. in the appliance Busybox 1.13.2 was used.

5. **Linux Kernel**: - Linux kernel version 2.6.28 was build and installed in the appliance.

6. **C Library**: - uClibc library was chosen as it provides all the needed functionality and also keeps the library size small thereby reducing the overall appliance size.

The appliance that we built contains the above packages and kernel installed. There were no extra packages installed keeping the appliance specific to a particular task and small in size. The section below describes steps followed to build the agropedia appliance. The appliance was build on top of Ubuntu 8.04 i686-linux-gnu machine, the target system was Asus Eee Box i586-geode-linux-uclibc.

The figure below describes the step by step process needs to follow to build an agropedia appliance.



Steps in Building an agropedia Appliance

9.2 How we built a Cross-Compiling toolchain?

Our previous discussion on cross-compilation toolchain gives a broad overview and general techniques one can follow to build a cross-compilation toolchain, in this section we will give the steps followed by us to build one, we have used crosstool-ng (http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool) to build the cross-compilation tool chain.

1. Download and install crosstool-ng.
2. Copy the crosstool and uclibc configuration files from the samples directory located in the source code for i586-geode-linux-uclibc to the current directory.
3. Run **`$ct-ng menuconfig`**. This will open a configuration menu which is used to configure the crosstool depending on the target system architecture in our case it is Asus Eee box. Keep the install path as default.
4. To start building the cross-toolchain run the following command **`$ct-ng build`**. This command will download and build all the packages needed to build the cross-compilation toolchain.

Once the toolchain is successfully build set the environment to use the cross-compiled gcc as the default C compiler. Add the path where cross-tool chain was installed to PATH variable. Use the following command if the toolchain was installed in the default path.

```
$export PATH = ~/x-tools/i586-geode-linux-uclibc/bin:$PATH
```
After adding the bin to the path set the environment variable CC, CXX etc by using the following commands
```
$export CC = i586-geode-linux-uclibc-gcc
$export CXX = i586-geode-linux-uclibc-g++
$export AR = i586-geode-linux-uclibc-ar
$export AS = i586-geode-linux-uclibc-as
$export LD = i586-geode-linux-uclibc-ld
$export RANLIB = i586-geode-linux-uclibc-ranlib
$export READELF = i586-geode-linux-uclibc-readelf
$export STRIP = i586-geode-linux-uclibc-strip
```

Once the above things are setup we can start by building our appliance, we will start first by building the linux kernel.

9.3 Building the Linux kernel

In section 6 of this report we discussed about the aspects related to compile a Linux kernel from source in general, this sections reports the configuration changes and specific steps followed by us to build and install Linux kernel for the appliance. Source code for Linux kernel 2.6.28 was obtained and unpacked. To build the linux kernel run the following command in the source directory.

`$make defconfig` (This command will make the default config file for x86 processor)
`$make menuconfig` (This will open a menu where we need to select the appropriate drivers to build.)

For Asus Eee box select ATA/ATAPI in device driver menu, from network card drivers menu choose r8169 driver. Wireless card driver was identified as ralink 2790, this driver till date was not included in the kernel and has to be built and loaded separately.

After choosing the appropriate drivers we can start compiling the kernel. Following command was used to compile the kernel.

```
$make ARCH=x86 CROSS_COMPILE=i586-geode-linux-uclibc- bzImage
```

The above command will compile the kernel and will make a compressed image of the built kernel in arch/x86/boot/bzImage file.

9.4 Busy box Installation

Section 8.1 and 8.2 contains the description about busybox and installing instructions followed in general, this section describes the steps we followed to install Busybox on our Appliance. Obtain the busybox [8] source code from the website [8] (we have used busybox 1.13.2), extract and cd into the source directory. Follow the following commands to build and install busybox: -

`$make menuconfig`. (This will give a graphical menu from which we can choose which all ulility we want in our system. Choose the appropriate utilities we want to use in the appliance.)
`$make ARCH=x86 CROSS_COMPILE=i586-geode-linux-uclibc-` (This will compile the source)
`$make install` (This will installed the busybox utilities in _install/ subdirectory).

9.5 Making the root file system

Section 7 contains a general filesytem description and minimal filesytem present on a Linux system, this section contains the steps we followed to build the filesystem hierarchy present in the agropedia appliance. We will now make the root file system for our appliance which will be later copied directory into the appliance hard disk. We will install all the packages on this root filesystem. Use the following commands (as mentioned here http://cross-lfs.org/view/clfs-embedded/x86/final-preps/creatingdirs.html) in order to build a root filesystem

```
$mkdir /mnt/rootfs
$cd /mnt/rootfs
$mkdir -pv
{bin,boot,dev,{etc/,}opt,home,lib/{firmware,modules},mnt}
$mkdir -pv {proc,media/{floppy,cdrom},sbin,srv,sys}
$mkdir -pv var/{lock,log,mail,run,spool}
$mkdir -pv var/{opt,cache,lib/{misc,locate},local}
$install -dv -m 0750 root
$install -dv -m 1777 {var,}tmp
$mkdir -pv usr/{,local/}{bin,include,lib,sbin,src}
$mkdir -pv usr/{,local/}share/{doc,info,locale,man}
$mkdir -pv usr/{,local/}share/{misc,terminfo,zoneinfo}
$mkdir -pv usr/{,local/}share/man/man{1,2,3,4,5,6,7,8}
$mkdir -pv cross-tools{,/bin}
$for dir in usr{,/local}; do
  ln -sv share/{man,doc,info} ${dir}
done
```

Next step is to create the passwd, group, and log files. Create the /etc/passwd file by running the following command (as mentioned at http://cross-lfs.org/view/clfs-embedded/x86/final-preps/creatingfiles.html) :

```
cat > etc/passwd << "EOF"
root::0:0:root:/root:/bin/ash
EOF
```

Create the /etc/group file by running the following command:

```
cat >etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF
```

The **login**, **agetty**, and **init** programs (and others) use a number of log files to record information such as who was logged into the system and when. However, these programs will not write to the log files if they do not already exist. Initialize the log files and give them proper permissions:

```
$touch var/run/utmp var/log/{btmp,lastlog,wtmp}
$chmod -v 664 var/run/utmp ${CLFS}/var/log/lastlog
```

After the root filesystem is ready with the directory hierarchy, we will now copy the linux kernel and busybox we have build above to the appropriate directory.

- Go to the source directory of the kernel, copy bzImage from arch/x86/boot/bzImage to /mnt/rootfs/boot directory.
- Now go to the busybox source directory and use the following command to copy the busybox from the _install subdirectory to the root filesystem
  $rsync -a _install/ /mnt/rootfs/

By following the steps above we have a built a bootable system having kernel and busybox installed. For the init script we will use the default init script which comes with busybox. Next we will start by building and installing other essential packages we have identified above for the agropedia Appliance.

9.6 Installing the HTTP server

Now we will install the http server in our appliance root filesystem. For the http server we have chosen lighttpd server for the reasons mentioned above. Lighttpd server 1.4.20 was used. Follow the following steps to install lighhtpd server to the root filesystem: -

- Download and extract the source code from the lighhtpd (http://www.lighttpd.net/) official website, cd into the source directory.
- Execute the following command in order to configure lighhtpd to install on the root filesystem:
  ```
  $./configure --host=i586-geode-linux-uclibc \
  --prefix=/home/mitesh/lighthttpd-install \
  --without-zlib --without-bzip2 --disable-ipv6
  ```
- Run **$make**
- Run **$make install**

The steps mentioned above will install the lighttpd server in our /mnt/rootfs directory, which contains the root filesystem of our appliance.

9.7 Installing the Database Server

Having installed the http server we will now start installing the database server which will be used by our appliance. We have chosen MySql as our database server, version used was 5.1.30. We will cross-compile our MySql server now, to do this one should follow the steps mentioned below: -

- Download and extract the mysql source code from the mysql website (http://www.mysql.com/), cd into the source directory.
- In order to make mysql successfully cross-compile we have to make the following changes in the source:
  - Edit sql/Makefile.in, find the following lines :
    ```
    done
    gen_lex_hash$(EXEEXT): $(gen_lex_hash_OBJECTS)
    $(gen_lex_hash_DEPENDENCIES)
    @rm -f gen_lex_hash$(EXEEXT)
    ```

```
$(gen_lex_hash_LINK) $(gen_lex_hash_OBJECTS)
$(gen_lex_hash_LDADD) $(LIBS)
mysql_tzinfo_to_sql$(EXEEXT):
$(mysql_tzinfo_to_sql_OBJECTS)
$(mysql_tzinfo_to_sql_DEPENDENCIES)
@rm -f mysql_tzinfo_to_sql$(EXEEXT)
$(mysql_tzinfo_to_sql_LINK)
$(mysql_tzinfo_to_sql_OBJECTS)
$(mysql_tzinfo_to_sql_LDADD) $(LIBS)
```

- Change the line
  $(gen_lex_hash_LINK) $(gen_lex_hash_OBJECTS) $(gen_lex_hash_LDADD) $(LIBS) to
  gcc $(CXXFLAGS) -I../include -I/usr/include -L/usr/lib/mysql -lmysqlclient -o gen_lex_hash gen_lex_hash.cc

- Save and close the file.

- Now edit include/config.h file and change the line **#define STACK_DIRECTION** to **#define STACK_DIRECTION 0**

- After making the above mentioned changes in the source code we can now start cross-compiling the mysql server, issue the following command in order to configure mysql source for cross-compilation

```
$./configure --host=i586-geode-linux-uclibc
ac_cv_sys_restartable_syscalls=yes \
--prefix=/mnt/rootfs \
--without-ndb-binlog \
--without-docs \
--without-man \
--with-readline \
--enable-local-infile \
--with-mysqld-ldflags=-static \
--with-client-ldflags=-static \
--disable-assembler \
--with-big-tables \
--with-extra-charsets=complex \
--enable-thread-safe-client
```

- After configuring the source run **$make**
- Run **$make install** to install the mysql server in our root file system.

Now we have a mysql server installed in our root filesystem which is cross-compiled using our cross-compiler to run against uClibc C library.

9.8 Installing PHP

We will now install PHP identified as the requirement to run agropedia. We need to cross-compile php and install it on our root filesystem. Complete the following steps in order to install php: -

- Obtain and extract the php source code from the website (http://www.php.net/), cd into the source directory.
- Configure php for cross-compilation by issuing the following command: -

```
$./configure --host=i586-geode-linux-uclibc \
 --prefix=/mnt/rootfs/ --enable-fastcgi --disable-ipv6 \
 --disable-libxml --disable-xml --disable-xmlreader \
 --disable-xmlwriter --disable-dom --disable-simplexml \
```

```
--without-iconv --without-pear --with-mysql=/mnt/rootfs/ \
--without-pdo-sqlite --with-config-file-path=/usr/local
```

- Run **$make**
- **$make install**

The above procedure will successfully build and install php into the root filesystem.

9.9 Other Necessary files

We will now create the necessary files which are used by a linux system in order to boot and configure itself the necessary content for the file used here can be found at http://cross-lfs.org/view/clfs-embedded/x86/bootscripts/chapter.html. We have followed the steps mentioned at the above mentioned link to create the necessary files our system.

9.9.1 Boot scripts

Some essential boot scripts can be obtained from http://cross-lfs.org/files/packages/embedded-0.0.1/clfs-embedded-bootscripts-1.0-pre4.tar.bz2
Extract and move into the directory of the file downloaded above, and install it by issuing the following command:-

```
$make DESTDIR=/mnt/rootfs install
```

9.9.2 mdev.conf

We will now configure the mdev which is a busybox replacement of udev. For this we will create /etc/mdev.conf for use with our system: -

```
$cat > /mnt/rootfs/etc/mdev.conf<< "EOF"
# /etc/mdev/conf
SLEEP=10

# Symlinks:
# Syntax: %s -> %s

MAKEDEV -> ../sbin/MAKEDEV
/proc/core -> kcore
fd -> /proc/self/fd
mcdx -> mcdx0
radio -> radio0
ram -> ram1
sbpcd -> sbpcd0
sr0 -> scd0
sr1 -> scd1
sr10 -> scd10
sr11 -> scd11
sr12 -> scd12
sr13 -> scd13
sr14 -> scd14
sr15 -> scd15
sr16 -> scd16
sr2 -> scd2
sr3 -> scd3
sr4 -> scd4
sr5 -> scd5
```

```
sr6 -> scd6
sr7 -> scd7
sr8 -> scd8
sr9 -> scd9
stderr -> fd/2
stdin -> fd/0
stdout -> fd/1

# Remove these devices, if using a headless system
# One will see an error mdev: Bad line 35
vbi -> vbi0
vcs -> vcs0
vcsa -> vcsa0
video -> video0
# Stop Remove for headless system

# Devices:
# Syntax: %s %d:%d %s
# devices user:group mode
null 0:0 777
zero 0:0 666

urandom 0:0 444

console 0:5 0600
fd0 0:11 0660
hdc 0:6 0660
kmem 0:9 000
mem 0:9 0640
port 0:9 0640
ptmx 0:5 0660

sda* 0:6 0660
sdb* 0:6 0660
hda* 0:6 0660
hdb* 0:6 0660

tty 0:5 0660
tty0* 0:5 0660
tty1* 0:5 0660
tty2* 0:5 0660
tty3* 0:5 0660
tty4* 0:5 0660
tty5* 0:5 0660
tty6* 0:5 0660

ttyS* 0:20 640
EOF
```

9.9.3 /etc/profile

We will now build /etc/profile file for our system, it is the file that specifices how the environment will function.

```
$cat > /mnt/rootfs/etc/profile<< "EOF"
# /etc/profile

# Set the initial path
```

```
export PATH=/bin:/usr/bin

if [ `id -u` -eq 0 ] ; then
        PATH=/bin:/sbin:/usr/bin:/usr/sbin
        unset HISTFILE
fi

# Setup some environment variables.
export USER=`id -un`
export LOGNAME=$USER
export HOSTNAME=`/bin/hostname`
export HISTSIZE=1000
export HISTFILESIZE=1000
export PAGER='/bin/more '
export EDITOR='/bin/vi'

# End /etc/profile
EOF
```

9.9.4 /etc/inittab

Now we will create /etc/inittab file for our system, it is file that specifices how to boot and shutdown a system.

```
$cat > /mnt/rootfs/etc/inittab<< "EOF"
# /etc/inittab
::sysinit:/etc/rc.d/startup
tty1::respawn:/sbin/getty 38400 tty1
tty2::respawn:/sbin/getty 38400 tty2
tty3::respawn:/sbin/getty 38400 tty3
tty4::respawn:/sbin/getty 38400 tty4
tty5::respawn:/sbin/getty 38400 tty5
tty6::respawn:/sbin/getty 38400 tty6
# Put a getty on the serial line (for a terminal)
# uncomment this line if your using a serial console
#::respawn:/sbin/getty -L ttyS0 115200 vt100
::shutdown:/etc/rc.d/shutdown
::ctrlaltdel:/sbin/reboot
EOF
```

9.9.5 Configuring the network card

If a network card is to be configured, decide on the IP address, FQDN, and possible aliases for use in the `/etc/hosts` file. We will now create the /etc/hosts file for the appliance: -

```
$cat > /mnt/rootfs/etc/hosts << "EOF"
127.0.0.1 localhost
[192.168.1.1] [<HOSTNAME>.example.org] [HOSTNAME]
EOF
```

The following command creates the `network.conf` file for use by the entire system:

```
$cat > /mnt/rootfs/etc/network.conf << "EOF"
# /etc/network.conf
# Global Networking Configuration
# interface configuration is in /etc/network.d/
```

```
# set to yes to enable networking
NETWORKING=yes

# set to yes to set default route to gateway
USE_GATEWAY=no

# set to gateway IP address
GATEWAY=192.168.0.1
EOF
```

The following command creates a sample `interface.eth0` file for the *eth0* device:

```
$mkdir  /mnt/rootfs/etc/network.d &&
cat >  /mnt/rootfs/etc/network.d/interface.eth0 << "EOF"
# Network Interface Configuration
# network device name
INTERFACE=eth0

# set to yes to use DHCP instead of the settings below
DHCP=no

# IP address
IPADDRESS=192.168.1.2

# netmask
NETMASK=255.255.255.0

# broadcast address
BROADCAST=192.168.1.255
EOF
```

If the system is going to be connected to the Internet, it will need some means of Domain Name Service (DNS) name resolution to resolve Internet domain names to IP addresses, and vice versa. This is best achieved by placing the IP address of the DNS server, available from the ISP or network administrator, into `/etc/resolv.conf`. Create the file by running the following:

```
$cat > /mnt/rootfs/etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf

domain [Your Domain Name]
nameserver [IP address of your primary nameserver]
nameserver [IP address of your secondary nameserver]
# End /etc/resolv.conf
EOF
```

Part of the job of the bootscripts is setting the system's hostname. This needs to be configured in the `/etc/HOSTNAME` file. Create the `HOSTNAME` file and enter a hostname by running:

```
$echo "agropedia" > /mnt/rootfs/etc/HOSTNAME
```

9.9.6 Device File

Following device nodes need to be created for the system to boot:

```
$mknod -m 0666  /mnt/rootfs/dev/null c 1 3
$mknod -m 0600  /mnt/rootfs/dev/console c 5 1
```

9.9.7 Finishing up

After installing the http server, database server, php and having setup all the necessary files we have completed our requirement as required by acquia drupal appliance to run. We now copied the code of the agropedia website into the www folder of the system. The lighttpd.conf file is used to store the configuration of the http server and can be edited to point to the www directory of the appliance. All the database files were also copied to the root file system.

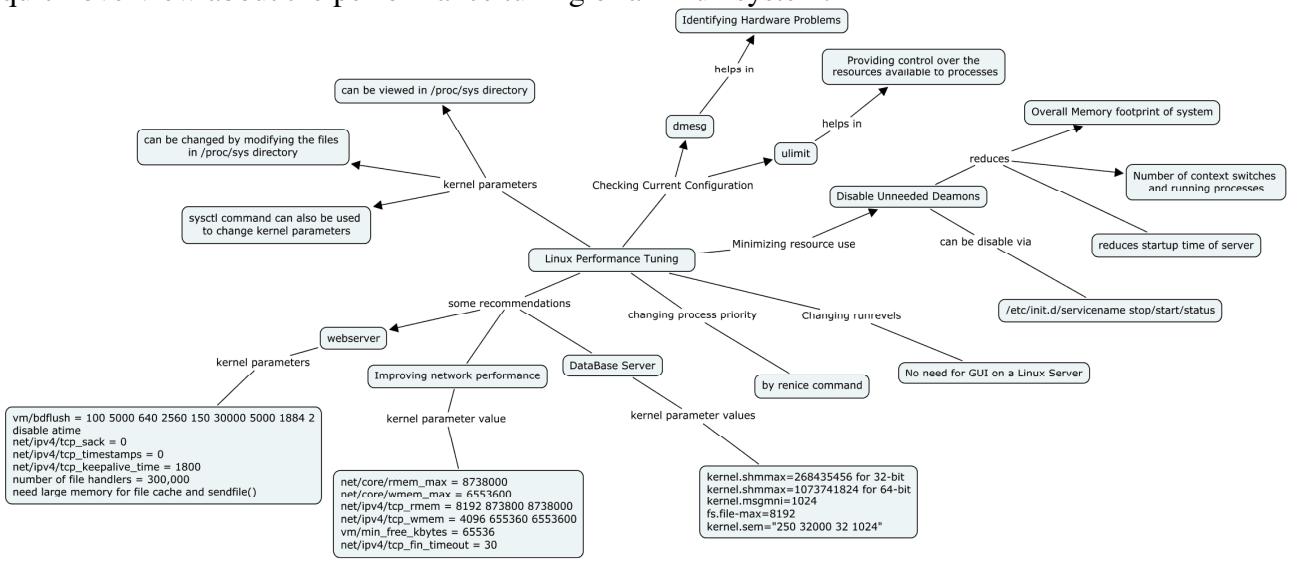
After setting up all the files in the root file system, we now have to copy the whole root filesystem to the target machine that is Asus EeeBox.

9.10 Booting up the System

After the root file system is copied into the appliance we can boot our appliance and connect to the appliance using a browser (from another machine). The agropedia website on the appliance is ready to use.

10. Performance Tuning

Linux kernel offers variety of parameters and settings to let the administrator tweak the system to maximize performance. This section describes the parameters one may want to change in order to maximize the performance of the appliance. One can change the system kernel parameters at run time and revert back to original if the change is found unsuitable. Following concept map gives a quick overview about the performance tuning of a Linux system.



CMAP describing Performance Tuning 1

10.1 Changing Kernel Parameters

The proc file system provides an interface to the running kernel that can be used for monitoring purposes and for changing kernel settings on the fly. To view the current kernel configuration, choose a kernel parameter in the /proc/sys directory and use the **cat** command on the respective file. Reading the files in the /proc directory tree provides a simple way to view configuration parameters that are related to the kernel, processes, memory, network, and other components. The following table lists some of the files that contain kernel information.

File/directory	Purpose
/proc/sys/abi/*	Used to provide support for "foreign" binaries, not native to Linux — those compiled under other UNIX variants such as SCO UnixWare 7, SCO OpenServer, and SUN Solaris™ 2. By default, this support is installed, although it can be removed during installation.
/proc/sys/fs/*	Used to increase the number of open files the OS allows and to handle quota.
/proc/sys/kernel/*	For tuning purposes, one can enable hotplug, manipulate shared memory, and specify the maximum number of PID files and level of debug in syslog.
/proc/sys/net/*	Tuning of network in general, IPV4 and IPV6.
/proc/sys/vm/*	Management of cache memory and buffer.

10.2 Tuning Process Priority

If a process is running too slowly, one can assign more CPU to it by giving it a lower nice level. Linux supports nice levels from 19 (lowest priority) to -20 (highest priority). The default value is 0. To change the nice level of a program to a negative number (which makes it higher priority), it is necessary to log on or **su** to root. To change the nice level of a program already running, issue the command:

```
$renice level pid
```

10.3 Tuning Virtual Memory Subsystem

Tuning the memory subsystem is a challenging task that requires constant monitoring to ensure that changes do not negatively affect other subsystems in the system. If one do choose to modify the virtual memory parameters (in /proc/sys/vm), it is recommend that changes are made to only one parameter at a time monitoring how the server performs. Some parameters that can be changed are:-

- The parameter stored in **`/proc/sys/vm/swappiness`** can be used to define how aggressively memory pages are swapped to disk.
- The value stored in **`/proc/sys/vm/dirty_background_ratio`** defines at what percentage of main memory the pdflush daemon should write data out to the disk.
- The parameter stored in /**`proc/sys/vm/dirty_ratio`** the system administrator can define at what level the actual disk writes will take place. The value stored in dirty_ratio is a percentage of main memory. A value of 10 would mean that data will be written into system memory until the file system cache has a size of 10% of the system's RAM

10.4 Tuning the disk subsystem

Disk plays an important part in the system. A slow disk can be a severe bottleneck in the system causing the system to response very slowly. Mounting file systems with the **noatime** option prevents inode access times from being updated. If file and directory update times are not critical to the implementation, as in a Web-serving environment, one might choose to mount file systems with the noatime flag in the /etc/fstab file. The performance benefit [6] of disabling access time updates to be written to the file system ranges from 0 to 10% with an average of 3% for file server workloads.

Block size is the smallest amount of data that can be read or written to a drive - it can have a direct impact on a server's performance. As a guideline, if the server is handling a lot of small files, then a

smaller block size will be more efficient. If the server is dedicated to handling large files, a larger block size might improve performance.

Other thing to keep in mind to increase the disk subsystem performance is choosing the right i/o elevator while compiling the kernel, choosing the right type of file system, right journaling mode of the file system and correct block size while formatting the file system.

10.5 Tuning the network subsystem

Network is an essential part of the system especially when the appliance is to be used as a network appliance such as web server appliance.

10.5.1 Increasing network buffers

The Linux network stack is careful when it comes to assigning memory resources to network buffers. In high-speed networks that connect server systems, these values should be increased to enable the system to handle more network packets.

- Initial overall TCP memory is calculated automatically based on system memory, one can find the actual values in:
  /proc/sys/net/ipv4/tcp_mem
- Set the default and maximum amount for the receive socket memory to a higher value:
  /proc/sys/net/core/rmem_default
  /proc/sys/net/core/rmem_max
- Set the default and maximum amount for the send socket to a higher value:
  /proc/sys/net/core/wmem_default
  /proc/sys/net/core/wmem_max
- Adjust the maximum amount of option memory buffers to a higher value:
  /proc/sys/net/core/optmem_max

10.5.2 Tuning window sizes

The values in the following file specify the amount of memory that is allocated for each TCP socket when it is created.

/proc/sys/net/core/wmem_max
/proc/sys/net/core/rmem_max

In addition, one should also set the values in the following files for send and receive buffers. The three values specify minimum size, initial size, and maximum size

/proc/sys/net/ipv4/tcp_rmem
/proc/sys/net/ipv4/tcp_wmem

The third value must be the same as or less than the value of wmem_max and rmem_max. Increase the first value on high-speed, high-quality networks so that the TCP windows start out at a sufficiently high value.

10.5.3 Some recommendations [6]

Increase TCP buffer sizes:-

```
/proc/sys/net/core/rmem_max = 16777216
/proc/sys/net/core/wmem_max = 16777216
/proc/sys/net/ipv4/tcp_rmem = 4096 87380 16777216
/proc/sys/net/ipv4/tcp_wmem = 4096 65536 16777216
```

Disabling the following parameters prevents a cracker from using a spoofing attack against the IP address of the server:

```
/proc/sys/net/ipv4/conf/eth0/accept_source_route=0
/proc/sys/net/ipv4/conf/lo/accept_source_route=0
/proc/sys/net/ipv4/conf/default/accept_source_route=0
/proc/sys/net/ipv4/conf/all/accept_source_route=0
```

If this server does not act as a router, it does not have to send redirects, so they can be disabled:

```
/proc/sys/net/ipv4/conf/eth0/send_redirects=0
/proc/sys/net/ipv4/conf/lo/send_redirects=0
/proc/sys/net/ipv4/conf/default/send_redirects=0
/proc/sys/net/ipv4/conf/all/send_redirects=0
```

Configure the server to ignore broadcast pings and smurf attacks:

```
/proc/sys/net/ipv4/icmp_echo_ignore_broadcasts=1
```

Ignore all kinds of icmp packets or pings:

```
/proc/sys/net/ipv4/icmp_echo_ignore_all=1
```

For servers that receive many connections at the same time, the TIME-WAIT sockets for new connections can be reused. This is useful in Web servers, for example:

```
/proc/sys/net/ipv4/tcp_tw_reuse=1
/proc/sys/net/ipv4/tcp_tw_recycle=1
```

By changing the tcp_fin_timeout value, the time from the FIN sequence to when the memory can be freed for new connections can be reduced, thereby improving performance.

```
/proc/sys/net/ipv4/tcp_fin_timeout=30
```

Reduce the TCP keep alive time which is the time after which an unused open connection will be dropped.

```
/proc/sys/net/ipv4/tcp_keepalive_time=1800
```

This can help to protect server from syn flood attack

```
/proc/sys/net/ipv4/tcp_max_syn_backlog=4096
```

11. Available tools

Here we will list some of the tools already available from which we can build a useful Linux system very small in size. The tools listed below are available only for a specific distribution of Linux.

1) **GNAP (Gentoo network appliance)**: GNAP is an easy way to build Gentoo-based network appliance systems, ready for use without the need for a full installation.
2) **MiniBSD**: miniBSD is a project developping a set of scripts that shrinks a running FreeBSD system to a small sized distribution suited for mass storage media, such as USB memory sticks and CF cards. The size of the distribution is generally about 12-15Mb and contains everything one needs to run a FreeBSD system comfortably. The scripts collect the necessary binaries, libraries, configuration files on a running FreeBSD system (4.x, 5.x and 6.x) and creates a disk image that can be saved on a CF card or USB memory stick. This way can be seemed as a top down approach for building an appliance.
3) **NanoBSD**: another FreeBSD tool that can be used to build specialised install images, designed for easy installation and maintenance of systems commonly called “computer appliances”.

12. Summary

This section provides a summary of the process to build an appliance. The two largest components of a standard Linux system are the utilities and the libraries. By replacing these with smaller equivalents a much more compact system can be built. Using Busybox and uClibc allows customizing the appliance by removing un-necessary features, thereby further reducing the final size of the system.

We started by constructing a cross-compilation tool chain which will compile the kernel, system applications and user specific application for the target system which can be a totally different architecture independent of the host machine i.e. our appliance. As a next step cross-compile the kernel and its module for the target system using the tool chain build. We then construct a root file system skeleton following the filesystem hierarchy standards for Linux. Then we copy system libraries, kernel and kernel modules to the filesystem skeleton created. We then start on building the system application BusyBox, which is both small and useful as it provided common utilities one will use on the system being developed, and install it on the root filesytem created above. Lastly we add any other application which a user desires to be in the appliance by cross-compiling that application again the tool chain and installing it on the root file system. This last step differs from application to application. To make the system bootable we need to install grub on the target root filesystem and add some initial system initialization scripts to it like /etc/inittab, /etc/init.d/rcS files.

The agropedia specific code can now be installed to the agropedia appliance, which can run www.agropedia.net website. A step by step procedure to build the agropedia appliance is described in section 9 of this report.

We then looked into tuning the Linux configuration (section 10)

13. Conclusion

In this task we investigated the issues related to building an appliance and brought together sufficient know-how to build one. We demonstrated this by building an appliance for an agriculture knowledge management system, agropedia.

An appliance is a good mechanism to deploy an OKS node. It can be fine-tuned not only to deliver the required functionality, but also loaded with mechanisms for data capture, analysis and reporting. We hope to be able to do this in Phase III.

14. Resources

1. Building Embedded Linux Systems, Second Edition
   by Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, and Philippe Gerum
2. http://free-electrons.com/articles/elfs
3. http://www.ibm.com/developerworks/edu/l-dw-linux-embedded-distro-i.html
4. http://cross-lfs.org/view/clfs-embedded/
5. http://www.tldp.org/HOWTO/Software-Building-HOWTO-1.html
6. http://www.redbooks.ibm.com/abstracts/redp4285.html

7. www.kernel.org
8. www.busybox.net
9. http://www.uclibc.org
10. http://ymorin.is-a-geek.org/dokuwiki/projects/crosstool
11. http://www.gentoo.org/proj/en/base/embedded/gnap.xml
12. http://www.freebsdnews.net/2008/09/02/embedded-freebsd-systems/
13. http://www.agropedi.net