**Chapter 1**

**INTRODUCTION**

The number of Smartphone users and mobile application offerings are growing rapidly. A Smartphone is often expected to offer PC-like functionality. , which requires powerful Processors, abundant memory and long-lasting battery life. However, their hardware today is still very limited and these limitations need to take into Consideration. In attempt to alleviate the limitations of Smartphone storages, we are proposing cloud computing environment. It allows to create virtual Smartphone images in the cloud and to remotely run their Mobile applications. They can choose to install their mobile applications either locally or in the cloud. Mainly it frees from the limit of processing power, memory and battery life of a physical Smartphone.

* 1. **PROJECT IDEA**

We are proposing cloud computing environment. Proposed system provides remote mounting of Android file system for diskless/cheap devices. Not a single system is exploring design of remote mounting of Android file system for diskless/cheap devices. This will be advantageous for high battery and high privacy. Our system also has options by which the user can perform the selection for storage. Any user will have its need as it provides high privacy, battery, etc.

**1.2 MOTIVATION OF PROJECT**

Smartphones nowadays have limited internal storage. Devices with more internal storage are costlier. And they are less affordable. Moreover, the OS uses 25% or more of the provided internal storage. Also an android application have less footprints, so we can think of loading the android middleware from cloud. So, the supporting files which an app requires will be fetched from cloud whenever an app runs.

**Chapter 2**

**PROBLEM DEFINITION**

**2.1 LITERATURE SURVEY:**

A. Present Practices

Many smartphone devices (iPhone models, iPhone 5C and iPhone 5S or Google nexus) are available in the market. In upgrading or buying for the first time, one of the decisions a buyer needs to make is about the size of storage memory. This decision is important because these models do not have a slot for an external memory card like many competing phones such as Samsung Galaxy S4. Three trends that are accelerating are storage memory hogs. Expert recommendation is to buy the maximum memory one can afford even though Apple (AAPL) charges an obscene amount for additional storage memory.

B. Memory Hogs

1. First consumers are downloading more and more video. A typical movie download from iTunes may eat 2GB of space. This does not mean that a consumer can download and keep eight movies on a 16GB phone. The reason is that the operating system IOS 7 can eat about 3GB of memory. Therefore on a 16GB phone only about 13GB is available.

2. Second, consumers are shooting more HD videos. HD video using the default camera app may consume about 150MB per minute. In the burst mode, iPhone 5S may eat about 25MB per second that the shutter is kept open.

3. Third, apps and especially games are becoming more graphic intensive and thus consume more memory For example, Infinity Blade 3 eats about 1.4GB. It is pretty easy to fill up 16GB. Even though it is tempting to buy a 16GB phone for cost reasons, most consumers will be sorry later on if they do not select a version with bigger memory from the start.

**2.2 PROBLEM STATEMENT**

Enabling disk-less/cheap Android devices (i.e. with minimal internal disk or flash memory).By minimal disk means, your Android phone would have 2GB disk as opposed to 8/16/32GB internal storage and no external SD card slots.

## 2.3 GOALS AND OBJECTIVE

This project requires to load Android directly from Cloud. For that working with Android Versatile Express emulator is required. Android system runs on top of Linux kernel. As a Linux-based Ubuntu root file system (rootfs) can be mounted over Network File System (NFS), the same has to be done for Android. Many devices already don't have external SD card slot (e.g. Google Nexus devices, Apple iPads etc.) The reason is that it is confusing for users. Apple/Google are trying to move users to cloud.

Dropbox enables Cloud storage for user data/documents. However, it iscurrently an Application that user downloads from the App Store. This project requires you to add support for Cloud Storage in a file system -something similar to NFS, but highly efficient by enabling intelligent caching. This file system will use local disk/SD card as cache for the disk in the cloud. If the data requested is not found in local cache, then a trip to the Cloud (over the network) is made to fetch that data using Wi-Fi/3G/4G LTE, whatever technology is available. The challenge is that each time a Cloud network trip is made over 3G/4G, Cellular Data Service is used and the user gets charged by their Service Provider (e.g. Airtel, Reliance etc.). This also drains battery on the device. So this file system must be highly efficient so as to save network trips by intelligently caching files locally.

**Chapter 3**

**SOFTWARE REQUIREMENTS SPECIFICATION**

* 1. **INTRODUCTION**

The following subsections of the Software Requirements Specifications (SRS) document provide an overview of the entire project. The number of Smartphone users and mobile application offerings are growing rapidly. A Smartphone is often expected to offer PC-like functionality. , which requires powerful Processors, abundant memory and long-lasting battery life. However, their hardware today is still very limited and these limitations need to take into Consideration. In attempt to alleviate the limitations of Smartphone storages, we are proposing cloud computing environment. It allows to create virtual Smartphone images in the cloud and to remotely run their Mobile applications. They can choose to install their mobile applications either locally or in the cloud. Mainly it frees from the limit of processing power, memory and battery life of a physical Smartphone.

**3.1.1 Project Scope**

Not a single system is exploring design of remote mounting of Android file system for diskless/cheap devices. Any user will have its need as it provides high privacy, battery, latency, etc.

* + 1. **Operating Environment**

1. Linux
2. Android OS

## Design and Implementation Constraints

1. Memory Required (Min) : 512 MB
2. Internal Storage : 2 GB
3. Connectivity : Wi-Fi, 3G/4G/LTE,etc

## Assumption and Dependencies

i. The User is expected to have basic android operating knowledge

ii. User is connected to a Network.(Wi-Fi, 3G/4G/LTE,etc)

iii. Android version 4.0 and above.

iv. Kernel with NFS support enabled.

* 1. **SYSTEM FEATURES**

## GUI Integration:

Our system allows applications to fetch supporting files from cloud and link them and run the app on the physical device with graphical-user-interface (GUI) functions between local and remote file-systems. It also features shortcuts to applications in the Smartphone that minimize the steps required for users to launch remote applications.

**3.2.1 Description and Priority**

The feature is of High priority. This is the Main feature of the System. You can say, the whole system is built around it.

**3.2.2 Stimulus/response sequence**

The front end user requests from across the internet and establishes remote sessions to the cloud. Then the change is reflected on users screen.

**3.2.3 Functional requirement**

## Relocation of Applications:

Another innovative feature is the ability to relocate applications and their associated data between Smartphone and local physical storage on cloud. This gives Smartphone users a new degree of freedom in managing their mobile applications.

**3.2.4 Description and Priority**

The feature is of high priority. This is the Supportive feature of the System.

**3.2.5 Stimulus/Response Sequences**

The front end user requests from across the internet and establishes remote sessions to the cloud. Once a remote session is established, the user can install and run mobile applications.

## Performance Boost:

Performance can be boosted by using the vast cloud resources. It free users from the processing-power, memory, and even battery-life limits of their Smartphone.

## Directly upload from your mobile devices:

## Are your mobile devices at maximum capacity? Save valuable space on your tablets and Smartphone with easy photo, video and file uploads direct to your personal cloud.

# 3.3 EXTERNAL INTERFACE REQUIREMENTS

### 3.3.1 User Interface

Android's default user interface is based on direct manipulation. Using touch inputs, that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard.

**3.3.2 Hardware Interfaces**

Android Tablet with Following Features:

* 1GB memory (RAM)
* 8/16GB internal flash memory (disk)
* Dual core processor (ARM or Intel) with Wi-Fi

**3.3.3 Software Interfaces**

Following software should be installed on the machine/smartphone where program will be used:

* Android OS Version 4.0 and above.
* Android emulator(Versatile Express emulator)

**3.3.4 Communication Interface**

* GPS
* Mobile networks: EDGE, GPRs, HSDPA, 4G/LTE..
* Wi-Fi connectivity.

**3.4 NON-FUNCTIONAL REQUIREMENTS**

Non-functional requirements define the needs in terms of performance, safety requirements, design constraints, reliability, availability, security, maintainability, and portability.

**3.4.1 Performance Requirements**

Following are some Basic Performance Requirements that should be followed to get maximum performance.

* Network Speed should be greater.
* Firewall apps should not interfere with the Software working.

**3.4.2 Safety Requirements**

Following are some Precautions that should be taken for safe execution of software

* Android can be configured to verify a user-supplied password prior to providing access to a device.
* Use of a password and/or password complexity rules can be required by a device administrator.
  + 1. **Security requirements**
* As long as server pc is kept safe data is safe & reliable.
* For public Cloud: encryption and decryption.
* For private cloud (Remote Server): security will be maintained automatically (compression or decompression).

**3.4.4 Software Quality Requirements**

* Maintainability: The Software is easy to maintain.
* Portability: The software doesn’t need any installation. It can be used directly.
* Usability: The software is easy to use.

**3.5 ANALYSIS MODEL**

3.5.1 Use Case Diagram

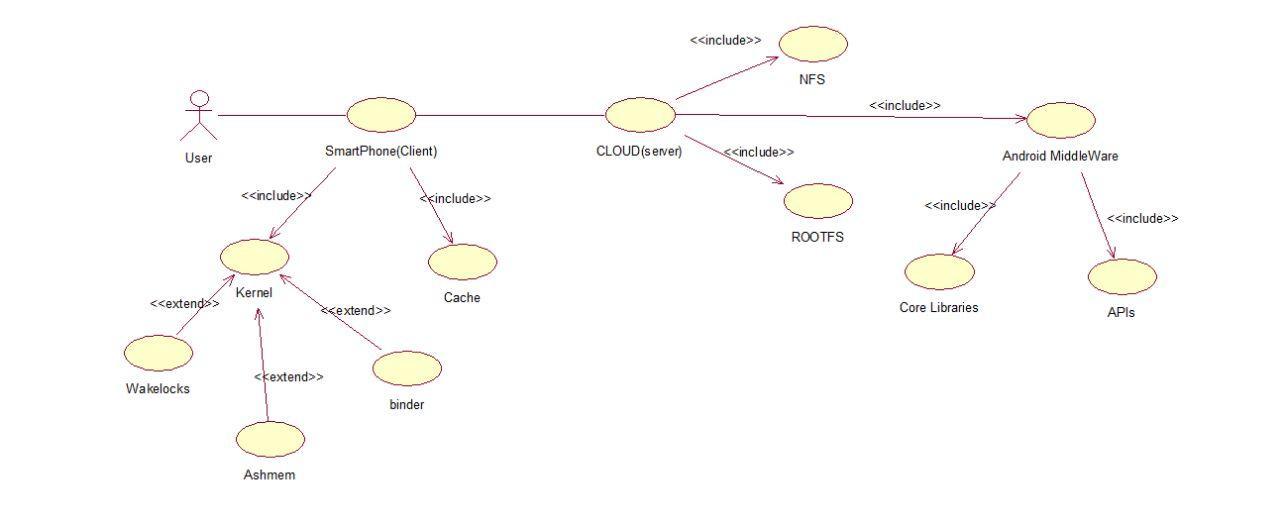


Figure 3.1: Use Case Diagram.

3.5.2 Activity Diagram

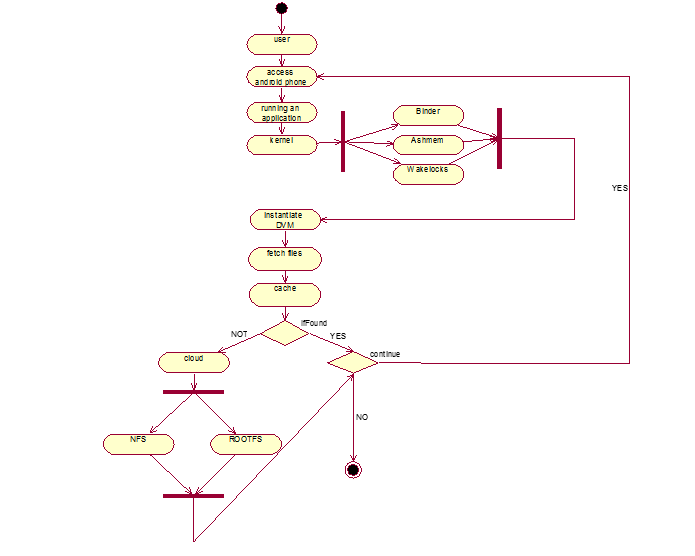


Figure 3.2: Activity Diagram

3.5.3 E-R Diagram

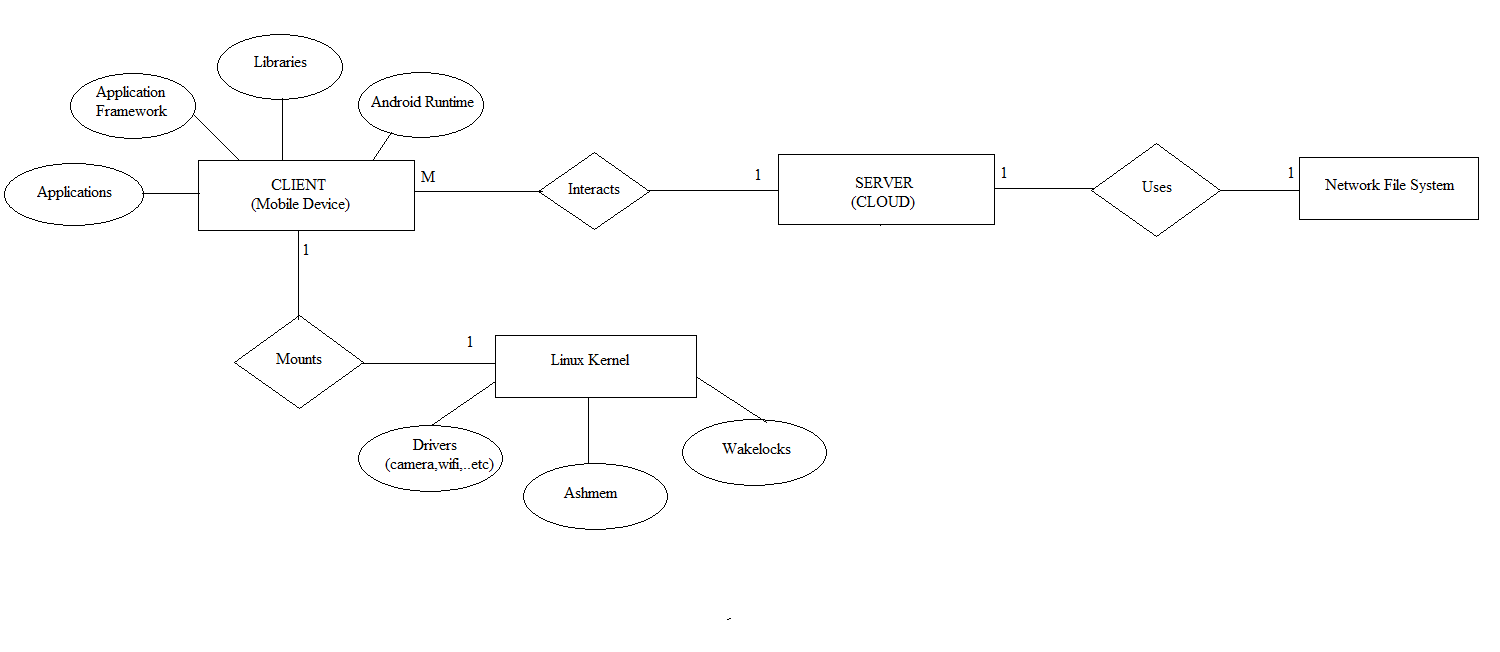


Figure 3.3: E-R Diagram

**3.6 SYSTEM IMPLEMENTATION PLAN**

**3.6.1 SIP**

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Duration** | **Task** | **Description** |
| **Phase 1** | July - August  (21st July – 21st Aug) | Literature survey | * Collect raw data and elaborate on literature surveys. |
| **Phase 2** | August– September  (22nd Aug – 30th Sept) | Analysis | * Analyze the information from a related IEEE paper. * Analysis of Problem statement and its solution. * Statistics of existing system of the organization and proposed system. |
| **Phase 3** | October-December  (1st Oct-21st Oct,  5thDec-15th Dec) | Design | * Designing UML diagrams. * Assign the module and design the process flow control. |
| **Phase 4** | December-February  (16th Dec – 15th Feb) | Coding  &  Testing | * Deciding the Modules for Implementation. * Implementation of proposed Algorithm in Java by using Struts II framework |
| **Phase 5** | February - March  (16th Feb – 10th Mar) | Documentation | * Preparing the necessary Documents after the project Completion * Preparing Test- Cases, its Documents etc * Preparation of Final Project Report. |

Table 3.1 System Implementation Plan

**3.6.2 Gantt chart for Project Scheduling**

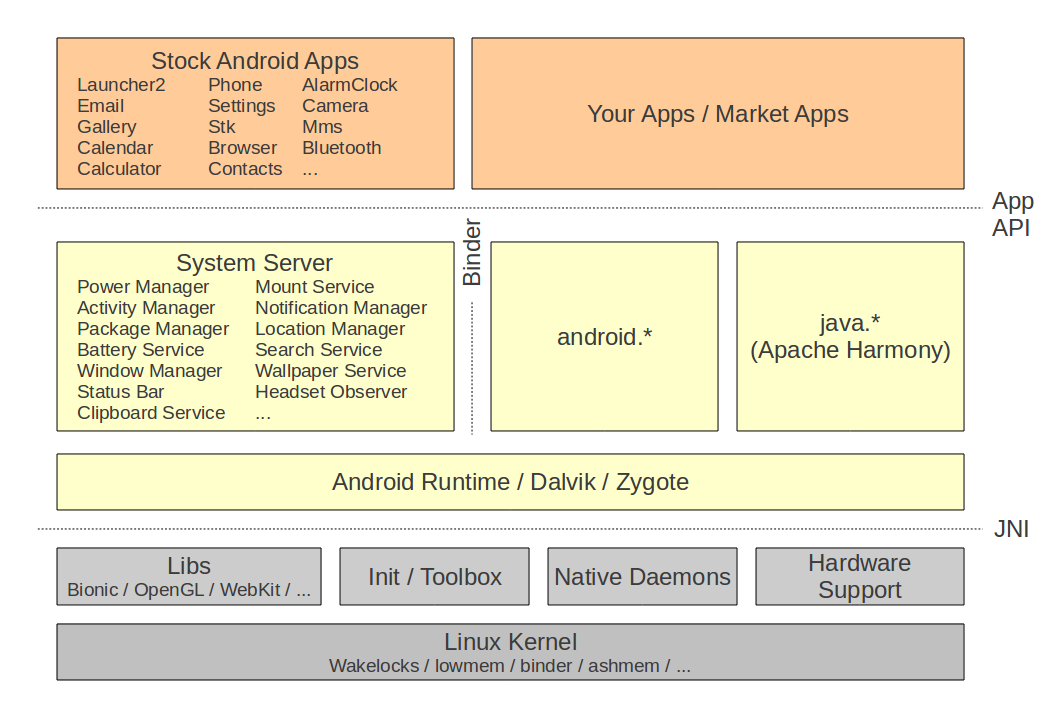
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Months**  **Phases** | **July -2014** | **Aug–2014** | **Sept -2014** | **Oct -2014** | **Dec -2014** | **Jan – 2015** | **Feb - 2015** | **March – 2015** |
| **Phase 1** |  |  |  |  |  |  |  |  |
| **Phase 2** |  |  |  |  |  |  |  |  |
| **Phase 3** |  |  |  |  |  |  |  |  |
| **Phase 4** |  |  |  |  |  |  |  |  |
| **Phase 5** |  |  |  |  |  |  |  |  |

Table 3.2 Gantt chart for Project Scheduling

**Chapter 4**

**SYSTEM DESIGN AND IMPLEMENTATION**

**4.1 SYSTEM ARCHITECTURE**

Figure 4.1 – System Architecture

Architecture defines overall system. Discovery module is responsible for carry out discovery of devices connected in the network. When discovery is completed, the results are stored in discovery database. This is secondary database which is compared with actual database for finding the discrepancies. Reconciliation module then takes those discrepancies & ask user to reconcile. If user performs the reconciliation then changes are updated in main database.

## Linux kernel

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

## Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

## Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

## Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

## Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, and Games etc.

**4.2. IMPLEMENTATION**

The approach that we followed is to NFS-mount our root file system. This means that all files are actually on a host. The host is configured to export the file system, mount the file system at boot time and use it as its own. All files are visible on the host.

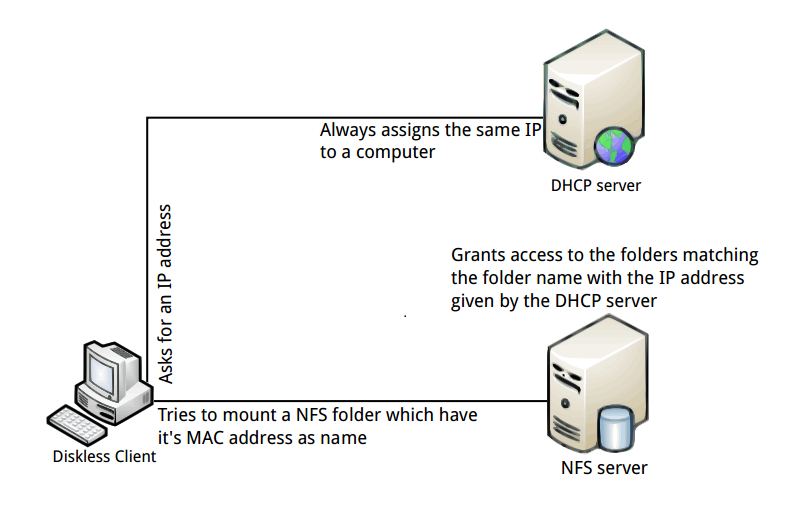


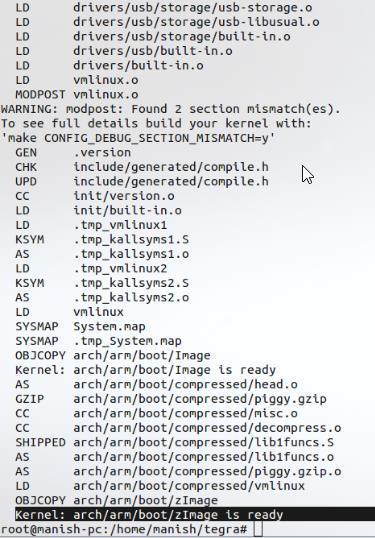
Fig 4.2 External Storage Mount [12].

As targets have increased in storage capacity and speed, the need for NFS-mounting root file systems will decrease. The technique is still useful when bringing up a new port or if you have lots of files that need to change together.

**4.2.1 Building of kernel**

Initially we started to build a custom kernel and gave it NFS and FUSE support. For the same we took the kernel source which is compatible with the device over which flashing of the kernel is to be carried out. And after the Z image is built from the kernel source, it is useful for creating the boot image file that will replace the one which is already there in the ROM of that device in order to flash the NFS and FUSE supported kernel.

NCURSES [13] is a clone of the original System. It is a freely distributable library and compatible with older version of curses interchangeably. It is a library of functions that manages an application's display on character-cell on the command line. In the remainder of the document, the terms curses are used

 **Steps for building the kernel:**

|  |
| --- |
| 1. Installing the libraries ncurses  >**sudo apt-get install libncurses5-dev**  2. For Fetching the kernel source  >**gitclone https://android.googlesource.com/kernel/tegra kernel**  3. For the building of kernel using cross compile tool chain  **>makeARCH=armCross\_Compile=arm-eabi-4.6/bin/arm-eabi**  4. Building the source using “make”  >**make** |

Fig. 4.3 creating the zImage

**4.2.2 Root File system over network file system**

To proceed with this we looked at how EMAC OE systems can be booted using NFS (Network File System) as the root file system. This method can be especially useful during development where the root file system is changing frequently. This can save time as well as wear on the on-board flash device.

**4.2.2.1 Root File system**

A complete root file system for the EMAC OE system [14] to boot from must be stored on the NFS server. The NFS server must be configured to allow clients to access this file system. The root file system does not have to be the directory shared by the NFS file system; it can be in a subdirectory, which means many root file systems can be shared by one NFS server.

**4.2.2.2 Network File system**

To boot an EMAC OE system [15] over NFS, an NFS server must be available on the local network. This is often the same machine that is being used for software development. EMAC recommends using the **nfs-kernel-server** package available on most desktop Linux distributions, if setting up a new NFS server. Once the server has been installed, export a directory to use as the root file system. This is often done using the **/etc/exports** file. This document assumes that the root file system for the board will be located at **/srv/nfs/rootfs** on the NFS server.

**4.2.2.3 Quick emulator**

QEMU [16] is used for CPU emulation for user-level processes, allowing applications are compiled for one architecture to be run on another. Qemu is a virtual machine monitor: It emulates CPUs through dynamic binary translation and provides a set of device models, enable it to run a variety of unmodified guest operating systems. And used together with KVM in order to run virtual machines at near-native speed.

|  |
| --- |
| 1. Installing port map and NFS kernel server  **> sudo apt-get install NFS-kernel-server port map (rebind)**  2. Configure sharing folder in exports  **> sudo gedit /etc/exports**  **>/tmp/nfs 10.0.2.15/255.255.255.0(rw,sync,no\_root\_squash,insecure)**  >**/tmp/nfs 127.0.0.1(rw,sync,no\_root\_squash,insecure)**  3. Kernel command line  >**root=/dev/nfs**  4.Starting the emulator  **emulator-debug init-kernel/home/manish/Versatile Express/arch/arm/boot/zImage -avd nx -qemu-M versatilepb -m 128M -append "root=/dev/nfs nfsroot=192.168.0.102:/srv/nfs/ rw”** |

**Steps for ROOTFS over NFS**

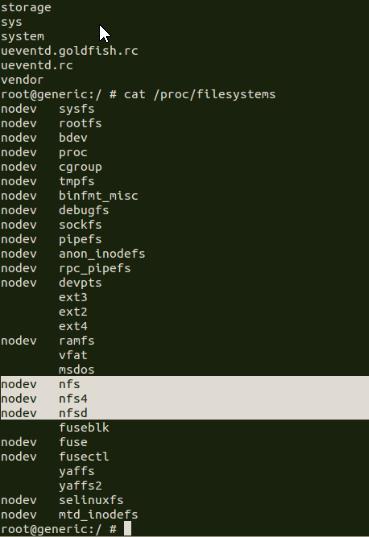
****

Fig. 4.4 Kernel configured to support NFS and FUSE

**4.2.2.4. Advantages of mounting rootfs over NFS:**

* Easy to access and modify the rootfs.
* No limit no size.

**4.2.2.5. Problems with NFS:**

* It takes memory.
* The logs aren’t kept after a reboot.
* Boot time is increased.
* Latency is observed in the system performance.

**4.3 FUSE BASED IMPLEMENTATION.**

As we saw, there are issues with mounting of rootfs over NFS, which lead us to proceed with another approach, i.e. to try to create a file system that emulates NFS, providing ability to read and write to the server using FUSE based implementation. File system in User space (FUSE) [17] is an operating system mechanism for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a "bridge" to the actual kernel interfaces.

FUSE is particularly useful for writing virtual file systems (VFS) [18]. Unlike traditional file systems that essentially save data to and retrieve data from disk, virtual file systems do not actually store data themselves. They act as a view or a translation of an existing file system or storage device. In principle, any resource available to a FUSE implementation can be exported as a file system. File system is a core component of a functional operating system. Traditional File system development has been confined to the kernel space.

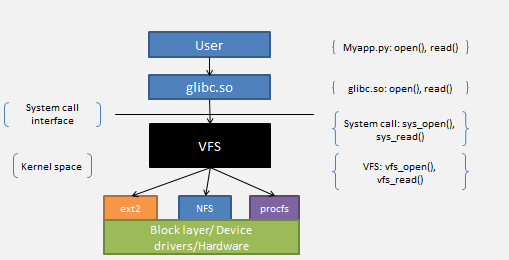


Fig 4.5 Virtual file system [19]

A customized, purpose-built, and user-driven File system development involves extensive knowledge of kernel internals, tools and processes. Alternatively, a user-space File systems are preferred over the kernel space File system, for ease of development, portability and developing prototypes file systems [20], particularly for intuitive abstraction of “non-file” objects. The main intention behind this is to create the file system using fuse that will work as network file system which is read/writes to remote server.

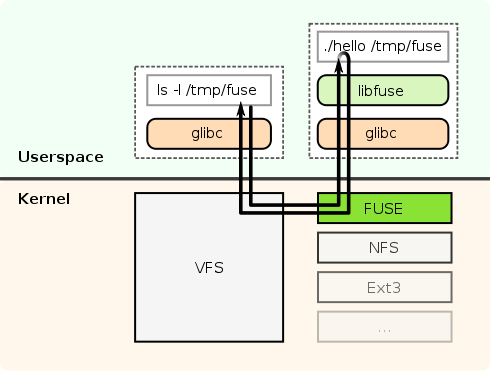


Fig 4.6 Working of File system in User spac

**4.3.1 Working of FUSE**

With the help of FUSE [21] it is possible to implement a fully functional file system in a user space program. The fuse includes

> Library

> Simple installation

> Secure implementation o User space kernel interface.

> Usable by non-privileged users

> Runs on Linux kernels 2.4.X and 2.6.X

> Very stable over time.

The communication between the fuse kernel module and fuse library is done via a special file descriptor which is obtained by opening **/dev/fuse.**  This file can be opened multiple times, and the obtained file descriptor is passed to the mount sys call, to match up the descriptor with the mounted filesystem. In UNIX kernel, a filesystem implementation is abstracted with virtual file system (VFS). VFS is acting as an interface to all available mounted filesystem on the computer file system. Fuse is a kind of a loadable kernel module and act as a file system to VFS. It registers itself with the VFS and opens a special device “/dev/fuse”. The use space library “libfuse.so” polls the device, and read “/dev/fuse” device. FUSE module is an interface between the fuse device and the VFS. It receives file I/O requests from the VFS and writes those requests to “/dev/fuse” device.

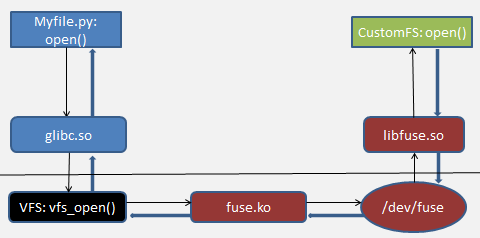


Fig. 4.6 Block diagram of Fuse internal [22]

**4.3.2 Cryptonite using Fuse**

Cryptonite [23] is a framework used for providing mainly industry-standard encryption to not just mobile device's local data, but also the data which is on drop box storage. Cryptonite is an open-source android file encryption and decryption tool based on EncFS and TrueCrypt and used for open EncFS-encrypted files and folders stored on your mobile device as well as browse and export drop-box storage.

If the support of FUSE and NFS is provided to own kernel, then mounting of EncFS and TrueCrypt on our own volume supports cryptonite. Drop box allows us to encrypt and decrypt data present over the drop-box storage. Same we are doing building of filsesystem which will have the access to upload and download (i.e. read and write data) so for this input will be the users’ android data and server will be the drop box. And main intention behind this is we need to replace our own filesystem with EncFS so that our file system will load directly from the cloud storage. Replacing of filesystem with EncFS to read data from remote server so that our own filesystem will talk to remote server to upload and download data from drop-box which very much similar to NFS.

Cryptonite uses fuse to provide encryption for data which is on remote server as well as local to the devices. So that same we are going to replace it with our own filesystem which is built with the help of lib fuse filesystem.

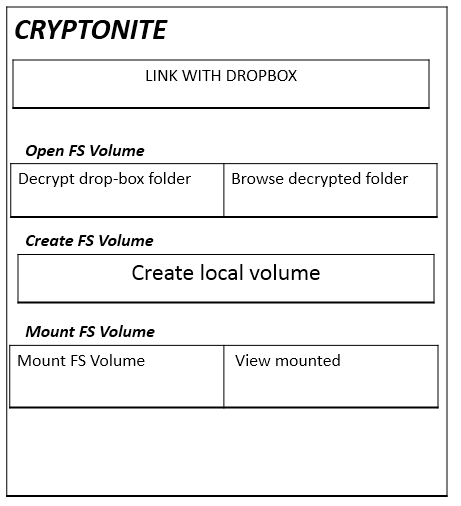


Fig.4.7 working of cryptonite

In cryptonite Files need to download and then re-upload to get them synchronized with with encrypted drop box folder because drop-box doesn't support direct file synchronization on Android. Second, most Android devices don't support FUSE so that EncFS volumes can't be mounted for direct and transparent file encryption and decryption. As a workaround, you can use one of the many apps that provide online storage synchronization (e.g. Folder Sync Lite or Drop sync). Setting up a scheduled sync to a local folder and mounting this local folder with Cryptonite will give you something very close to transparent file encryption and decryption. **4.4 BEHAVIOURAL DESIGN AND DESCRIPTION**

**4.4.1 State Machine Diagram**

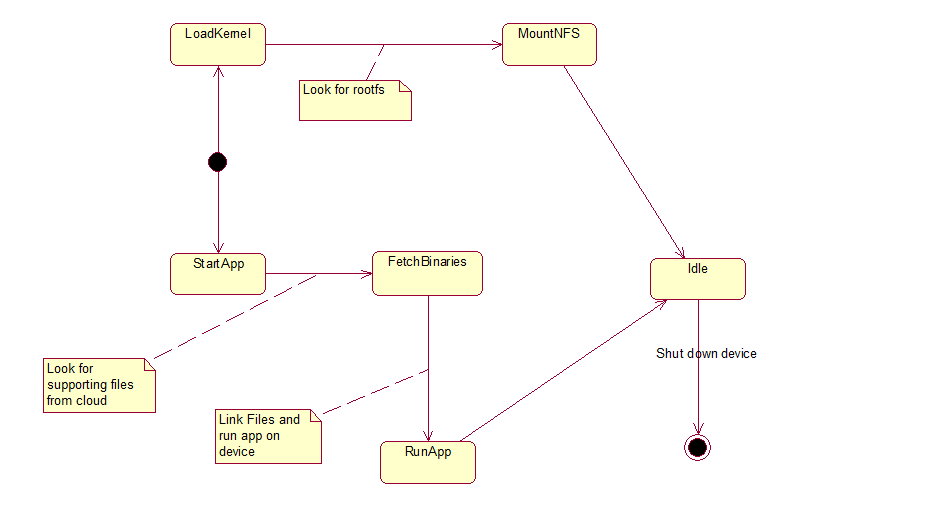


Fig. 4.8: State Machine Diagram

**4.5 USER INTERFACE AND DESIGN**

**4.5.1 Deployment Diagram**

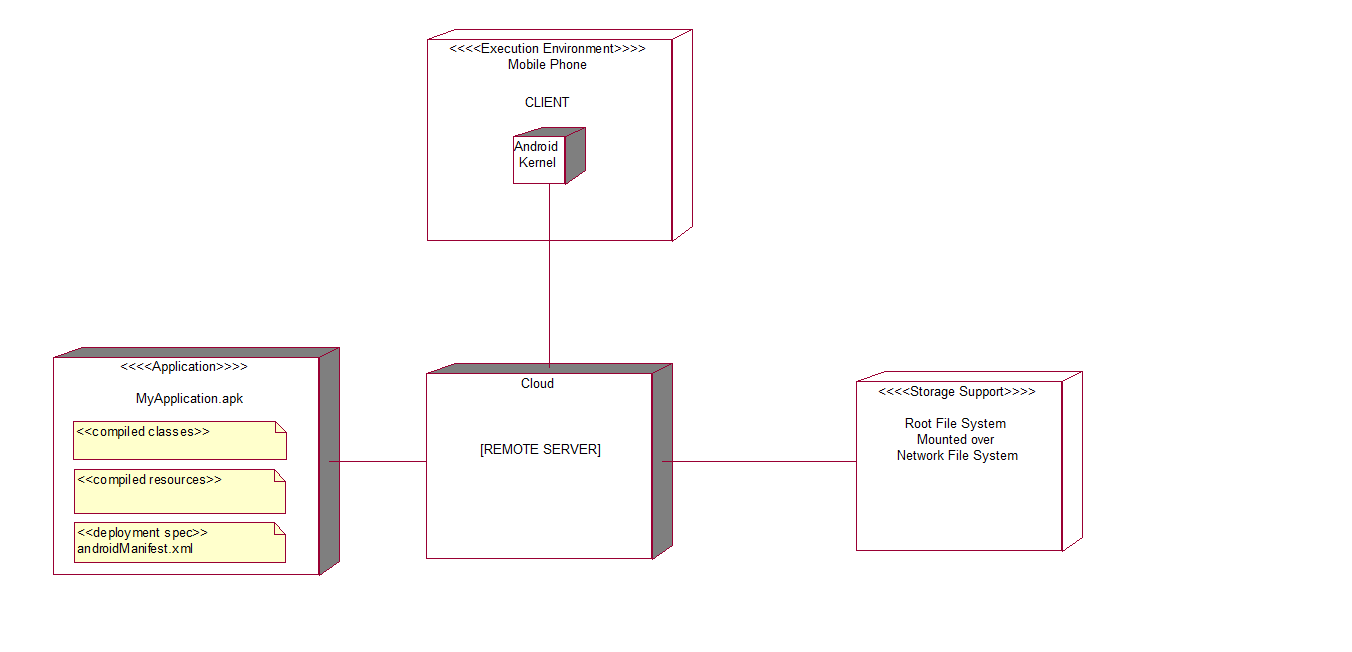


Fig. 4.9 – Deployment Diagram

**Chapter 5**

**TECHNICAL SPECIFICATION**

* 1. **TECHNOLOGY DETAILS**

1. **Java (JDK 7)**

Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer.

1. **Gold Fish Emulator**

The Android emulator runs a virtual CPU that Google calls Versatile Express. Versatile Express executes ARM926T instructions and has hooks for input and output -- such as reading key presses from or displaying video output in the emulator. These interfaces are implemented in files specific to the Versatile Express emulator and will not be compiled into a kernel that runs on real devices.

1. **Android Development Toolkit(A.D.T)**

The Eclipse ADT Bundle provides everything you need to start developing apps, including the Android SDK tools and a version of the Eclipse IDE with built-in ADT (Android Developer Tools) to streamline your Android app development. Applications are written using the Java programming language and run on Dalvik, a custom virtual machine designed for embedded use which runs on top of a Linux kernel.

**iv. Remote Server[cloud or PC]**

NFS mounts work to share a directory between several virtual servers. This has the advantage of saving disk space, as the home directory is only kept on one virtual private server, and others can connect to it over the network. When setting up mounts, NFS is most effective for permanent fixtures that should always be accessible.

Android system runs on top of Linux kernel .Linux-based Ubuntu root file system (rootfs) be mounted over Network File System (NFS). Same procedure is followed for android.

**5.2 ADVANTAGES**

## Reduces Battery Consumption

## Decreases Bandwidth utilisation over Network.

## Expandable storage

## Access from anywhere

**5.3 LIMITATIONS**

Main objective of the project is enable or add support for diskless/cheap android phones/tablets using private/public clouds. We will be addressing issues such as privacy, security. In addition, there will be systems and networking challenges such as:

1. What data to transfer? Should we transfer blocks as needed or should we do prefetching to reduce latency and improve app response time. Should we be caching data? What about user's privacy?

2. Too much data transfer will over 3G cellular network will consume data and cost user. Lot of to-and-fro data traffic between the device and cloud server will also affect the battery life. On-demand data fetching will increase latency and affect user experience. So there's a trade-off here and will only be revealed through experimentation.

3. Should we use Wi-Fi when available and not use cellular 3G to help alleviate data/battery consumption problem.

**5.4 APPLICATION**

In the upcoming future there will be Android applications for all the purposes of a common user. The Android Diskless System can be used in classrooms, libraries, navigation points, kiosks or even in Arcade machines using those apps.

Android diskless system is great for sharing files, but it's particularly good for sensitive information, like passwords etc. File System encrypts everything you upload, and downloads it over a secure HTTPS connection, meaning that file is secure from start to finish. Of course, you can also encrypt in a Zip File.

File System allows users to create a special folder on their computers, which System then synchronizes so that it appears to be the same folder (with the same contents) regardless of which computer is used to view it. Files placed in this folder are accessible via the folder, or through any other Android Device.

**5.5 MATHEMATICAL MODEL**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Description** | **Observation** |
| **1.** | **Problem Description and System** |  |
|  | Let S be Closed System defined as, S ={ Ip, Op, A, Ss, Su, Fi} | System |
|  | S ={ Ip1, Ip2, Ip3, Op1, Op, Op3, A, Ss, Su, Fi}  Where,  Ip1={Boot the device}  Ip2= {Install the application }  Ip3={Open the application}  Op1= {Retrieve main files of app from device}  Op2= { Retrieve supporting files from cloud}  Op3={Retrieve files from Intelligent cache}  Set of actions = A={F1,F2,F3,F4}  Where,  F1= Init  F2= Zygote  Ss- Set of User’s states  Ss={LoadKernel, MountNFS, StartApp, FetchBinaries, RunApp, Idle}  Su- success state  Fi- Failure state | Input at the start  Installation of the app  Run the app  Intermediate outputs obtained to reach final state  Actions are function operated on input data  States of a system  Desired state of the system  Abnormal termination of the system |

|  |  |  |  |
| --- | --- | --- | --- |
| 2. | Mapping Functions  f(x) | X | Y |
|  | F1(Ip1) →Op1  F1- Init  Ip1- Boot the device | Ip1 | Qp1  Qp1 is System becomes aware of the location of files. |
|  | F2(Qp1,Ip2,Ip3) → Qp2  F2-Zygote  Ip2-Istall the app  Ip3-Open the app | Qp1,Ip2,Ip3 | Qp2  Qp2 Retrieve files of corresponding app. |

**Chapter 6**

**PROJECT ESTIMATE, SCHEDULE AND TEAM STRUCTURE**

**6.1 PROJECT ESTIMATE**

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry W. Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current project characteristics.

**COCOMO** is defined in terms of three different models:

* The Basic Model
* The Intermediate Problem
* The Detailed model.

**Basic COCOMO** computes software development effort (and cost) as a function of program size.

Program size is expressed in estimated thousands of source lines of code (SLOC). COCOMO applies to three classes of software projects:

·Organic projects - "small" teams with "good" experience working with "less than rigid"

Requirements.

·Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements.

·Embedded projects - developed within a set of "tight" constraints. It is also combination of organic and semi-detached projects. An embedded project mode will require a great deal of innovation.

The basic COCOMO equations take the form

**-Effort Applied (E)** = ab (KLOC) ^ bb **[man-months]**

**SOFTWARE DEVELPOMENT TIME**

**-Development Time (D)** = cb (Effort Applied) ^db**[months]**

**-People required (P)** = Effort Applied / Development Time **[count]**

Where, KLOC is the estimated number of delivered lines (expressed in thousands) of code for project.

The coefficients *ab*, *bb*, *cb*and*db*are given in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Software Project | Ab | Bb | Cb | Db |
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semi-detached | 3.0 | 1.12 | 2.5 | 0.32 |
| Embedded | 3.6 | 1.2 | 2.5 | 0.35 |

Table Parameters for Project estimation as per Basic Cocomo Model

Basic COCOMO is good for quick estimate of software costs. However it does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.

For our project the estimate is as follows:

Since this is an organic project, the parameters are as follows:

Therefore,

Effort Applied (E) = ab (KLOC) ^ bb [man-months]

= 2.4(1.2) ^ 1.05

= 3.03 man-months\*

Development Time (D) = cb (Effort Applied) ^db [months]

= 2.5(3) ^ 0.38

= 2.15 months\*

People required (P) = Effort Applied / Development Time [count]

=1.40

\*Digits rounded off to two decimal places

**6.2 PROJECT SCHEDULE**

**1) Overview of Project:** We started the studying about the domain android and its related concepts and its different features from month of August to September.

**2) Complete Idea of Project**: Till the end of October we studied different android concepts, its platform, gamming concepts, etc. Our main aim was to learn this new platform and android and develop the game App.

**3) Requirement Phase:** As per the guidance given by our external and internal guides we understood that what were the basic requirements for our aim to be achieved. Approaching end of November we finalized all the requirements and what all was to be actually done.

**4) High level design:** High level design phase was completed by end of December. In that we started with developing models and controllers taking into consideration the architecture of the system.

**5) Detailed Design:** Beginning of January we started with detailed designing of the system. Detailed working of individual modules was finalized. Interaction between different modules and there integration like communication of frontend to back end was finalized in this phase.

**6) Coding phase:** Designing Graphical User Interface (GUI) and actual coding part was started as soon as designing was done. Implementation of design in actual form was carried out till the mid of March.

**7) Testing Phase:** Testing was carried in analogous with the actual coding.

**8) Project Documentation:** The documentation was carried out throughout the SDLC.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | NAME OF MODULE | MODULE Description in brief | Responsible Student Name for module | Duration of completion |
| 1 | Building and Configuration  boot image | The project is based on configuration of kernel image with FUSE and NFS support.  Extracting the Zimage and obtaining the boot image.  With the help of Fast boot boot loader and android debug bridge, flashed the custom rom and flashed the kernel. Which Is having FUSE and NFS support. | Abhijeet Daga | 60 days |
| 2 | Root File-system over Network File-system | 1. Configured the zImage to support NFS and FUSE for qemu.  2. Configured the zImage to support NFS and FUSE for the device.  3. Building of both the kernels.  4. Extracting the root file system.  5. Mounting the root file system over NFS on qemu. | Manish Ghumnani | 65 days |
| 3 | Fuse based File-system | Created fuse based file system.  Replace the same with built in encfs of cryptonite. | Prasanna pawar | 60 days |

**Chapter 7**

**SOFTWARE TESTING**

**7.1 INTRODUCTION**

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the [software](http://en.wikipedia.org/wiki/Software) to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding [software bugs](http://en.wikipedia.org/wiki/Software_bug) (errors or other defects).Testing is important phase in the development of software. Small amount of testing is involved in every phase while developing the software. There are two types of testing, White Box testing & Black Box testing. Black box testing is carried out to check the functionality of software. While box testing is carried out to review the code check it for effectiveness. Testing can be applied to individual modules as well as to all the modules when they are integrated. These two types are called as Unit testing & Integration testing. In testing we try to review each & every part of software & check whether system is behaving as per the need of the end user. Though it is true that not a single software is total bug free, we have to test the software in such a way that it should behave correctly in unexpected situation. Software which is very good providing vast functionality but not according to end user needs, then it is not acceptable. Hence software testing plays an important role in development of any system.

It involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

* meets the requirements that guided its design and development,
* responds correctly to all kinds of inputs,
* is sufficiently usable,
* can be installed and run in its intended environments, and
* Achieves the general result its stakeholder’s desire.

Software testing depending on the testing method employed can be implemented at any time in the development process. However, most of the tests efforts traditionally occur after the requirements have been defined and the coding process has been completed and has been shown that fixing a bug is less expensive when found earlier in the development process.

**7.2 TEST CASES**

A test case, in software engineering, is a set of conditions under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do. The mechanism for determining whether a software program or system has passed or failed such a test is known as a test oracle*.* It may take many test cases to determine that a software program or system is considered sufficiently scrutinized to be released.

**Unit Testing:**

Definition: Contents should be correctly displayed, avoiding any inconsistencies.

Methodology:   
Describe how unit testing will be conducted. Who will write the test scripts for the unit testing, what would be the sequence of events of Unit Testing and how will the testing activity take place?

**System and Integration Testing**

Definition: List what is your understanding of System and Integration Testing for your project.

Methodology:   
Describe how System & Integration testing will be conducted. Who will write the test scripts for the unit testing, what would be sequence of events of System & Integration Testing, and how will the testing activity take place?

**Performance and Stress Testing**

Definition: List what is your understanding of Stress Testing for your project.

Methodology:   
Describe how Performance & Stress testing will be conducted. Who will write the test scripts for the testing, what would be sequence of events of Performance & Stress Testing, and how will the testing activity take place?

**User Acceptance Testing**

Definition:  
The purpose of acceptance test is to confirm that the system is ready for operational use. During acceptance test, end-users (customers) of the system compare the system to its initial requirements.

Methodology:   
Describe how the User Acceptance testing will be conducted. Who will write the test scripts for the testing, what would be sequence of events of User Acceptance Testing, and how will the testing activity take place?

**Automated Regression Testing**

Definition:   
Regression testing is the selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still works as specified in the requirements.

Participants:

Prasanna Pawar

Manish Ghumnani

Abhijeet Daga

Memory Leakage Testing: Memory leakage happens when a computer program or application is unable to manage the memory it is allocated resulting in poor performance of the application and the overall slowdown of the system. As mobile devices have significant constraints of available memory, memory leakage testing is crucial for the proper functioning of an application.

Interrupt Testing: An application while functioning may face several interruptions like incoming calls or network coverage outage and recovery. The different types of interruptions are:

* Incoming and Outgoing [SMS](http://en.wikipedia.org/wiki/SMS) and [MMS](http://en.wikipedia.org/wiki/Multimedia_Messaging_Service)
* Incoming and Outgoing calls
* Incoming Notifications
* Battery Removal
* Cable Insertion and Removal for data transfer
* Network outage and recovery
* Media Player on/off
* Device Power cycle

An application should be able to handle these interruptions by going into a suspended state and resuming afterwards.

Installation testing: This testing is to be performed on qemu emulator for which a certain type of environment is to be setup. Ubuntu OS is to be used for setting up the required environment. Initially sdk and jdk are to be installed also the ncurses library is to be linked so as to run the emulator and configure the kernel through menuconfig.

Certification Testing: To get a certificate of compliance, each mobile device needs to be tested against the guidelines set by different mobile platforms. The Certified Mobile Application Tester popularly known as CMAT certification exam is offered by the Global Association for Quality Management (GAQM) via Pearson Vue Testing Center worldwide to benefit the Mobile Application Testing Community.

HARDWARE REQUIREMENTS:   
Computers  
Modems

Android Device

FEATURES TO BE TESTED:

Identify all software features and combinations of software features that will be tested.

FEATURES NOT TO BE TESTED:

Identify all features and significant combinations of features which will not be tested and the reasons.

Test reports:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | Test case name | Test case objectives | Preconditions | Expected result | Actual result | Pass/ Fail |
| 1 | Installation | Satisfy system requirements | Fuse and nfs support to kernel | Successful mounting of filesystem via nfs | Mounting of rfs over nfs | Pass |
|  |  | Emulator with modified ramdisk image and compiled kennel 3.4 | Start emulator command | Emulator must start with the GUI disabled | Emulator starts with the GUI disabled | Pass |
|  |  | Communication with the emulator | ADB command | Shell for the emulator must appear | Shell for the emulator appears | Pass |
|  |  | Mounting of sd card | Linux mount command | sd card must be mounted on emulator | sd card is mounted on the emulator | Pass |
|  |  | Switch root process | Switch root Linux command | Switch root to Ubuntu file system | Switched root to Ubuntu file system | Pass |
|  |  | Mounting of ROOTFS over NFS in android emulator | Mount the rootfs over NFS by passing parametersto the kernel command line | Mount command should show that rootfs is successfully mounted | Rootfs is successfully mounted over NFS | Pass |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | Feature testing | Correct working of emulator | Run the emulator with correct command line parameter | Emulator start  with mounting of rfs over nfs | Emulator boots with specified file system | Pass |
|  |  | Device startup | Rfs is at the server and kernel is local | Statndard boot time | Increased boot time | Pass |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | UI Testing | System UI | Contents should be correctly displayed, avoiding any inconsistencies. | Good UI response time | Certain delay in UI responsiveness | Pass |
|  |  | Switching between system applications. | Apps must be properly installed | Switching between the app without delay | Slight delay in switching between the apps | Pass |

**TEST CASES**

|  |
| --- |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| USE CASE | OBJECTIVE | ACTOR | PRE-CONDITION | BASIC FLOW | POST-CONDITION |
| Install App | Install application on device | Android user | User should have basic knowledge of installation. | 1. New user for the app is created.  2. App's directory is created in the internal storage.  3. The contents of the .apk file are extracted | Successfully completed installation |
| Booting | System will boot up. | Android user | Kernel should be loaded on device. | 1. System will boot up.  2. Init method will be called.   |  | | --- | |  | | System will boot up successfully. |
| |  | | --- | | Intelligent cache | | |  |  | | --- | --- | | To check whether supporting files are available on cache or not.   |  | | --- | |  | | | |  | | --- | | Android user | | |  | | --- | | Application should be installed on device only. | | |  | | --- | | 1.Application will fetch binaries  2. Check the files inside the cache. | | |  | | --- | | Retrieving the files through intelligent cache only. | |
| |  | | --- | | Cloud storage | | |  | | --- | | To store the supporting File system and middleware of Android System. | | |  | | --- | | Android user | | |  | | --- | | Supporting file system is not found inside the cache. | | |  | | --- | | 1. Checking the intelligent cache.  2. Considering the supporting file system is not found. 3.It will take the trip to cloud in order to retrieve the supporting files | | |  | | --- | | Successfully retrieve the supporting files | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr. No. | Module | Start Date | Expected End Date | Actual End Date |
| 1 | Compilation of Android Source Tree and Android SDK | 20/07/2014 | 30/07/2014 | 15/08/2014 |
| 2 | Versatile Express kernel Customization | 20/08/2014 | 28/08/2014 | 25/08/2014 |
| 3 | RAMDISK modification | 15/09/2014 | 30/09/2014 | 20/09/2014 |
| 4 | sdcard Build | 01/12/2014 | 10/12/2014 | 09/12/2014 |
| 5 | NFS | 05/01/2015 | 30/01/2015 | 30/01/2015 |
| 6 | Working on Emulator | 15/01/2015 | 30/01/2015 | 28/01/2015 |
| 7 | ROOTFS over NFS | 15/01/2015 | 20/02/2015 | 22/02/2015 |
| 8 | file system with FUSE that works as a network file system (read/writes to a remote server similar to NFS) | 12/02/2015 | 22/02/2015 | 24/02/2015 |
| 9 | build fuse on android using ndk | 25/02/2015 | 14/03/2015 | 16/03/2015 |
| 10 | approach similar to cryptonite and replace built in EncFS with our own file system similar to nfs | 20/02/2015 | 27/02/2015 | 01/03/2015 |
| 11 | consider laptop as a server that supports uploading and downloading android data using our own filesystem (similar to nfs) | 02/03/2015 | 20/03/2015 | \*\* |

**Chapter 8**

**Results Snapshots**

When ROOTFS is local

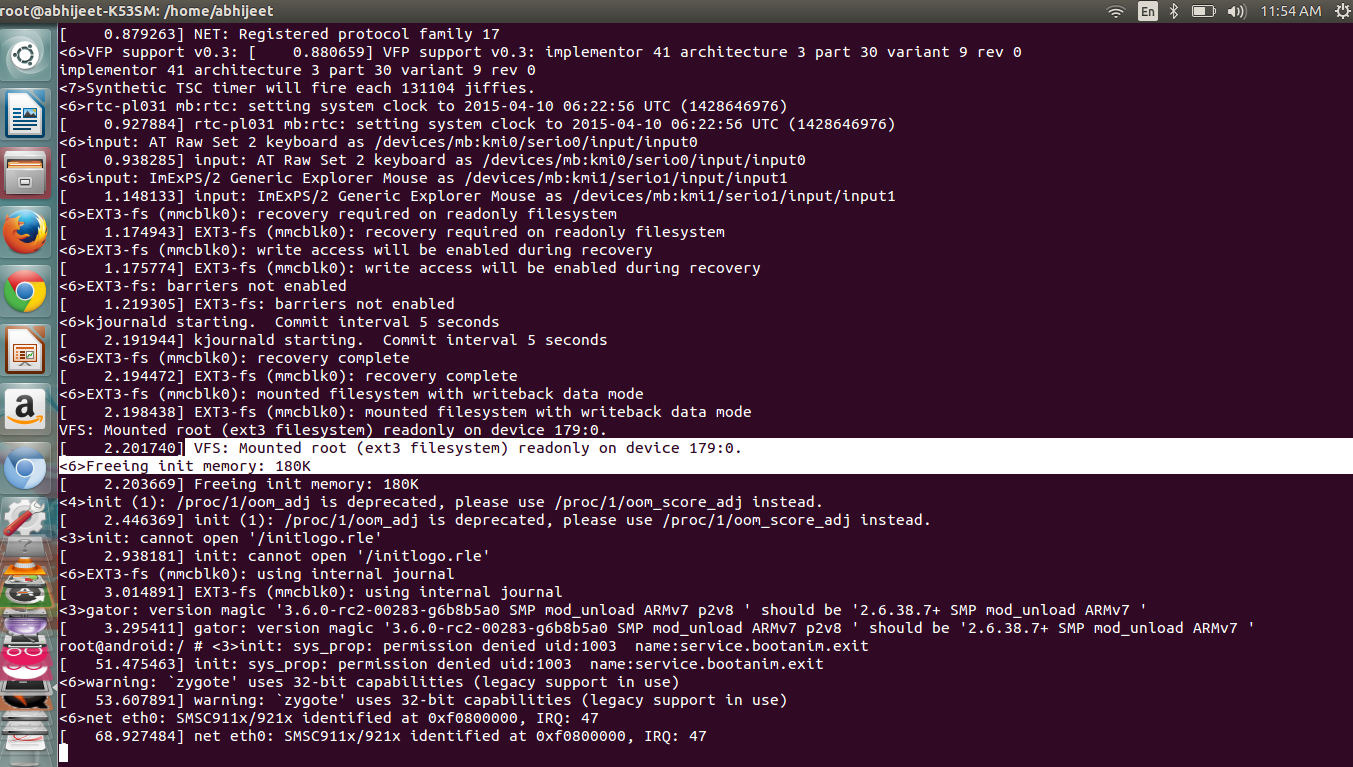


Fig.7.1 Mounting using sdcard (rootfs is local)

Fragmentation in Android OS means more internal memory is consumed

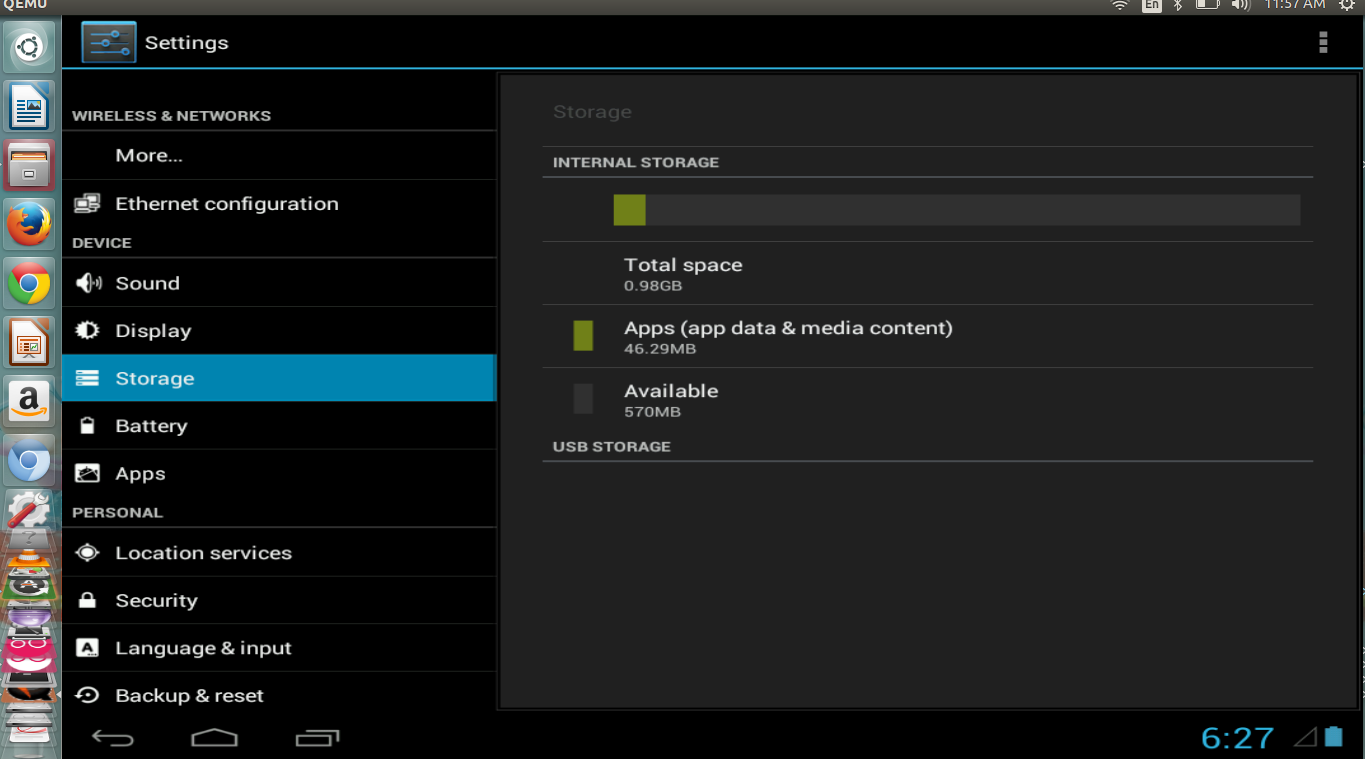


Fig.7.2 Android OS consumes internal storage

To mount ROOTFS over network

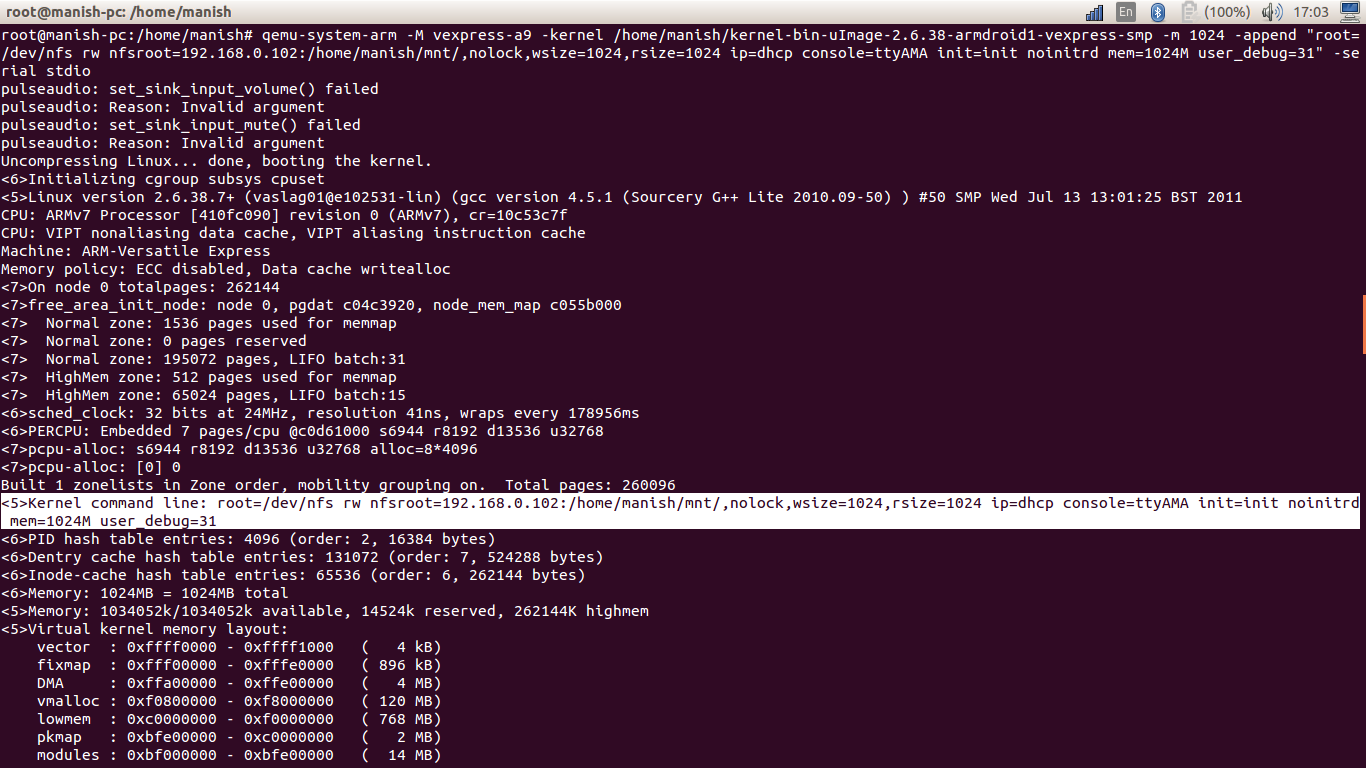
****

Fig.7.3 changing the kernel command line

ROOTFS is not local

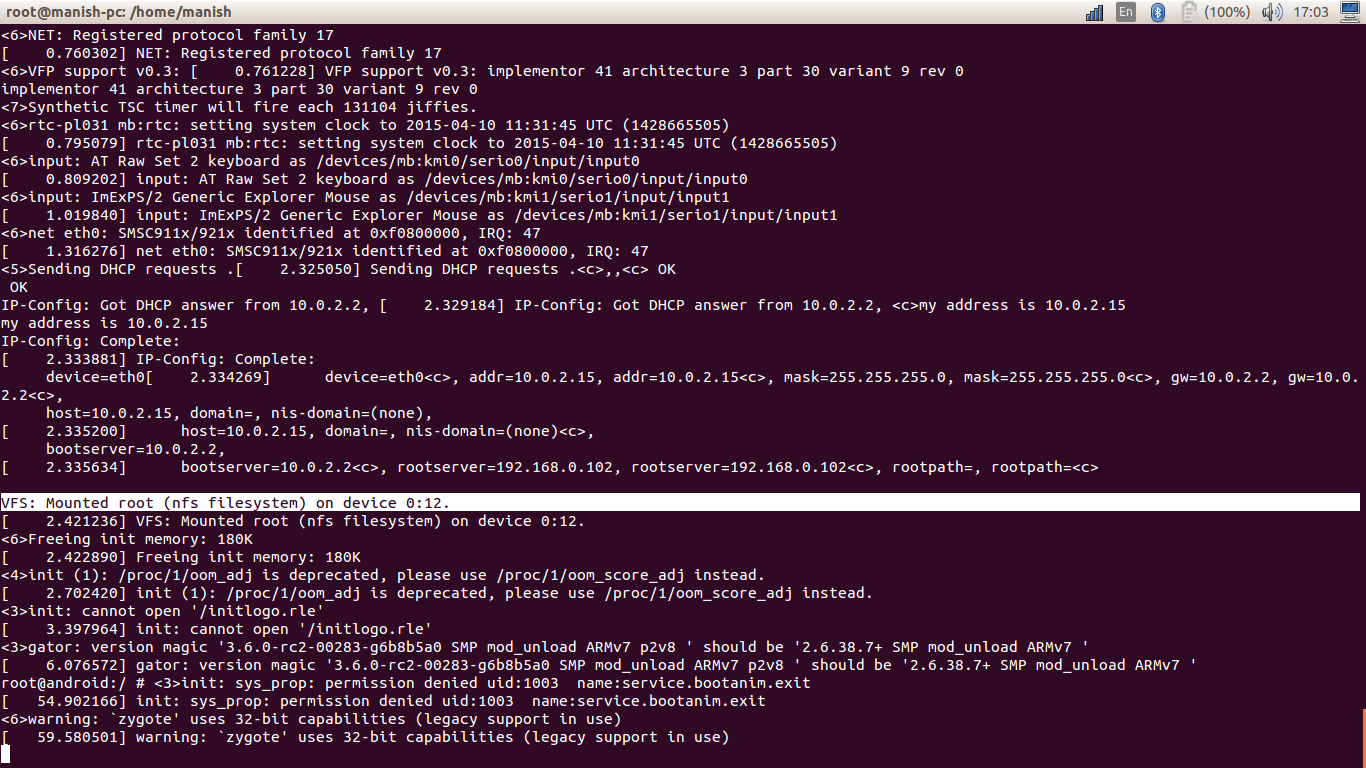


Fig.7.4 ROOTFS is mounted over network

NFS mounting results in more space

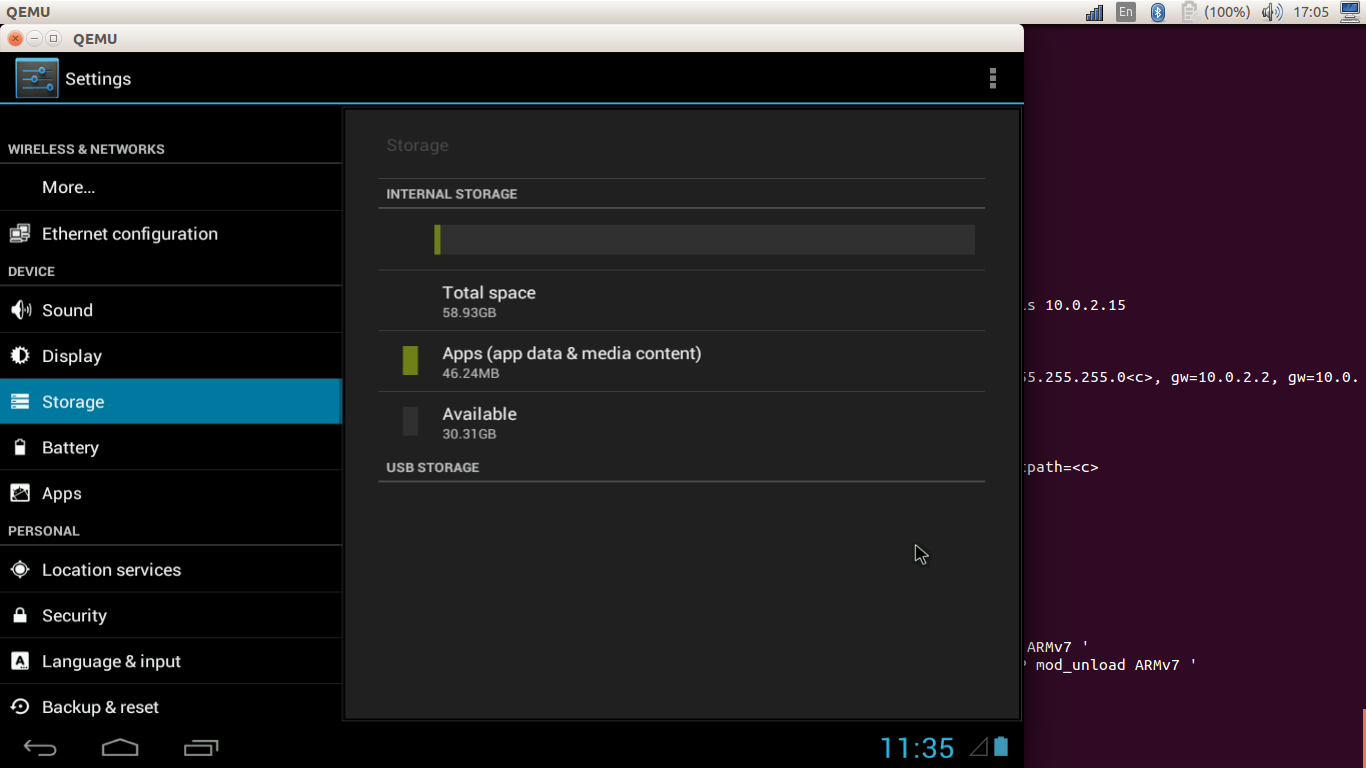
****

Fig.7.5 Entire server storage is available

Changes on the server are secure

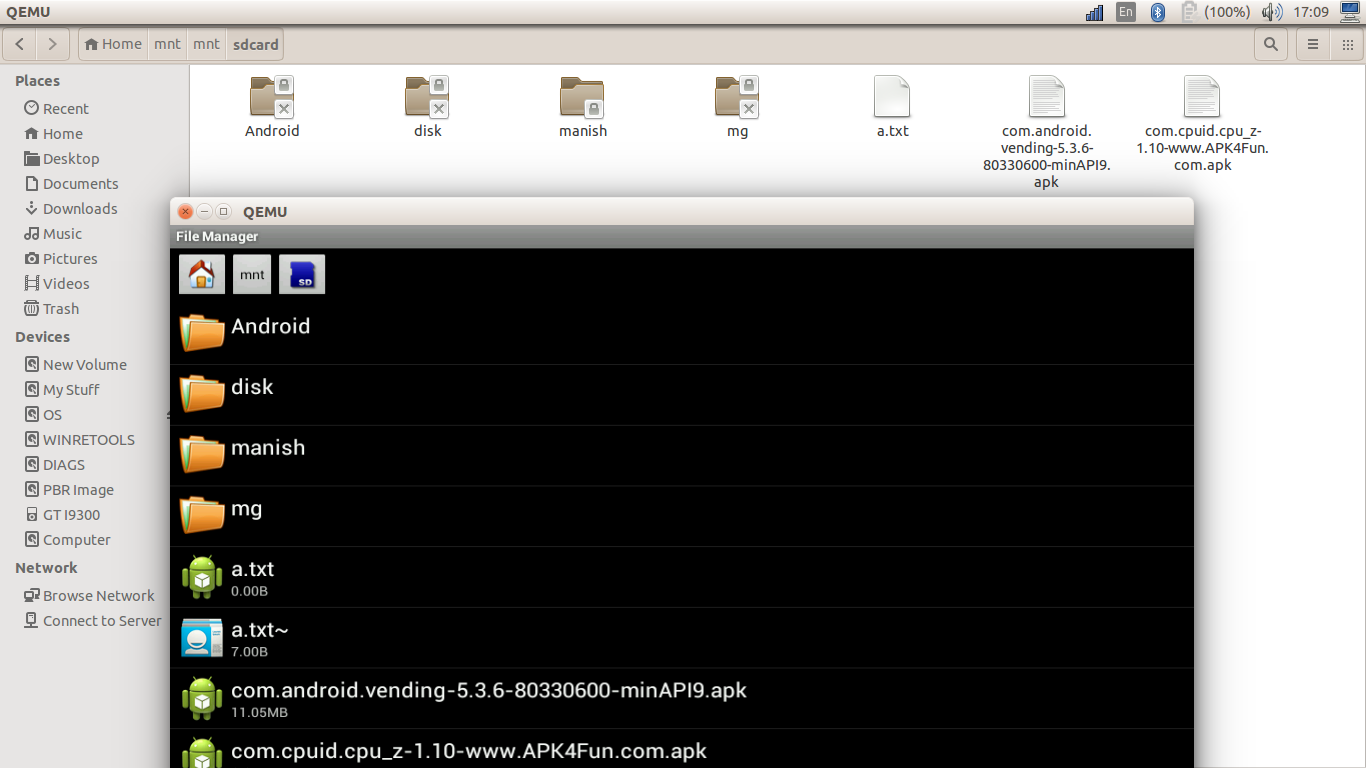


Fig.7.6 Changes are synchronized on both (client/server)

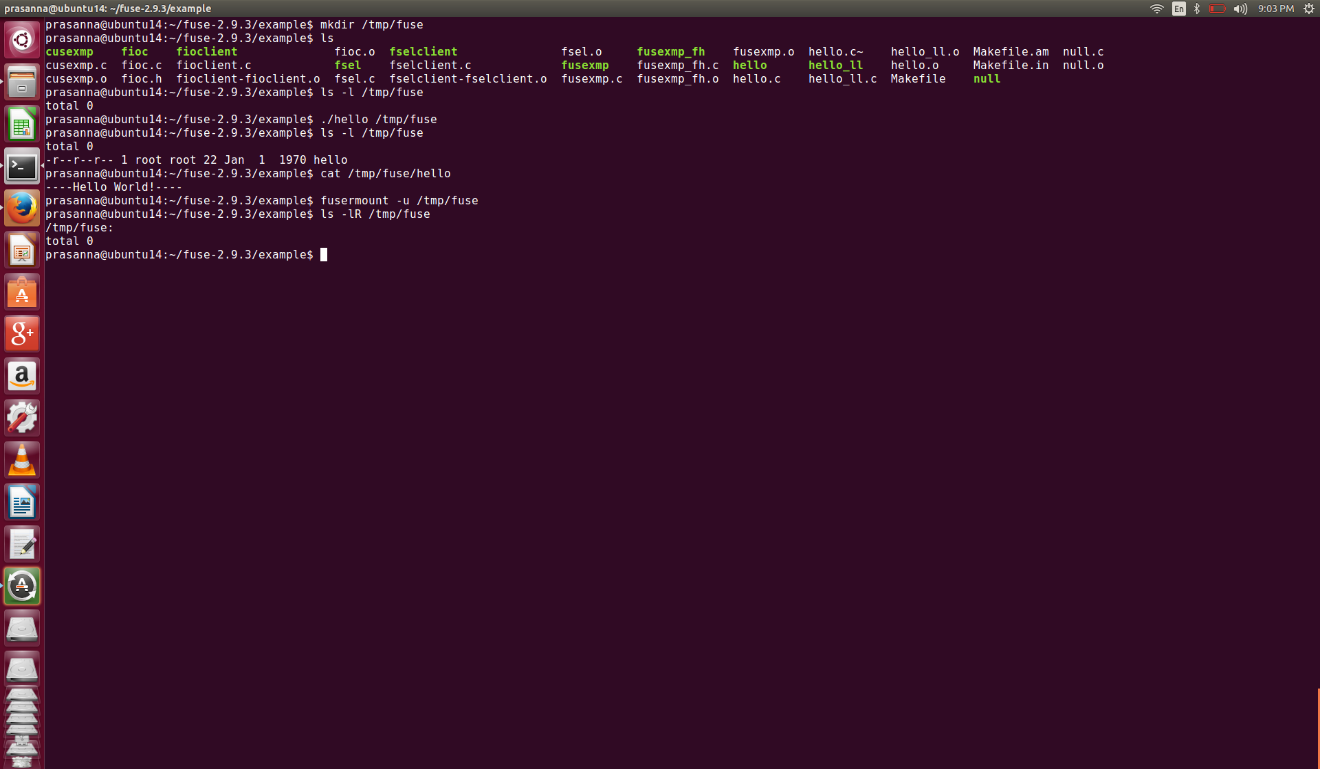
****

Fig.7.7 Sample Implementation of Fuse based FileSystem

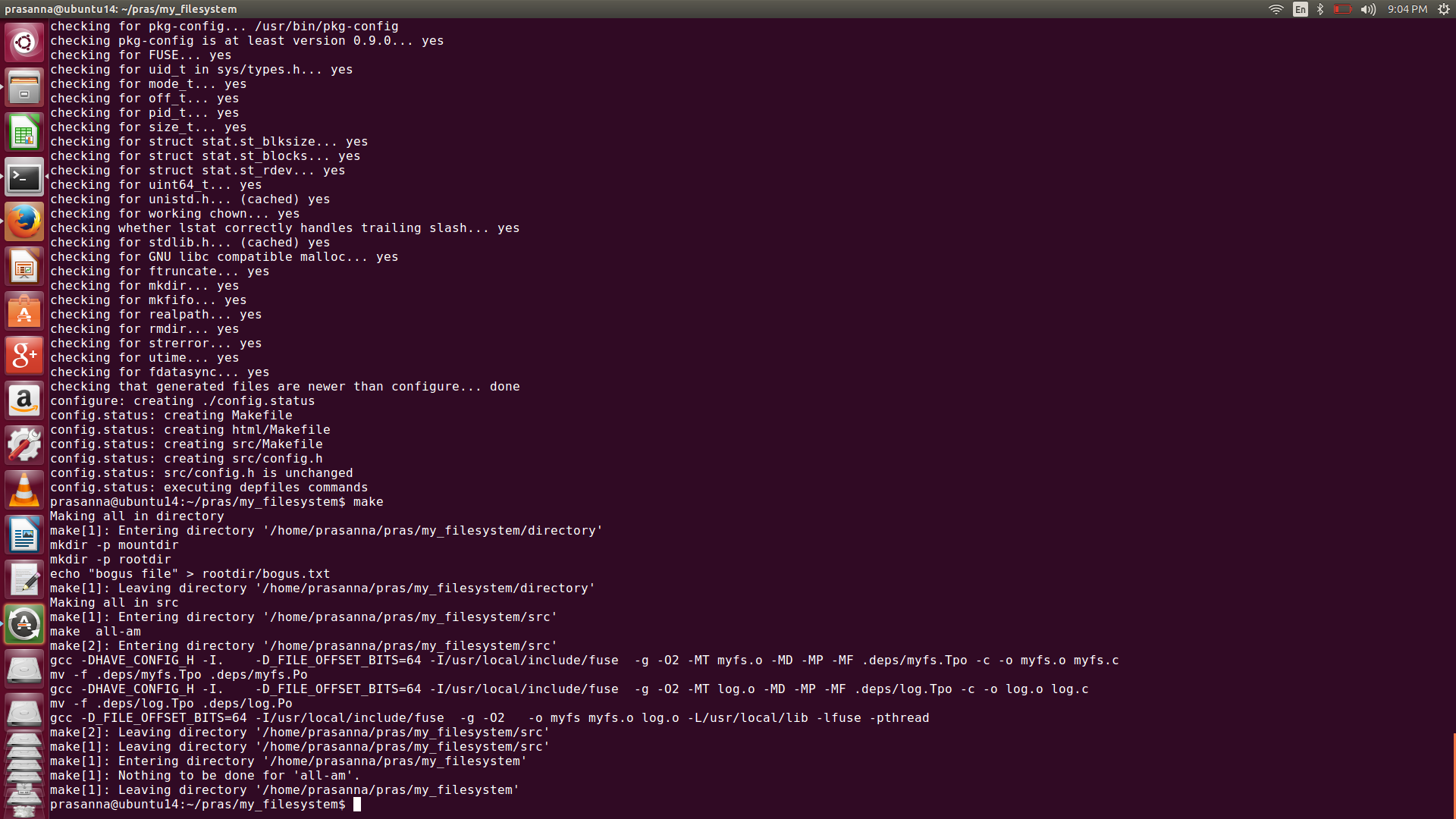
****

Fig.7.8 Configuring and installing of My\_Filesystem

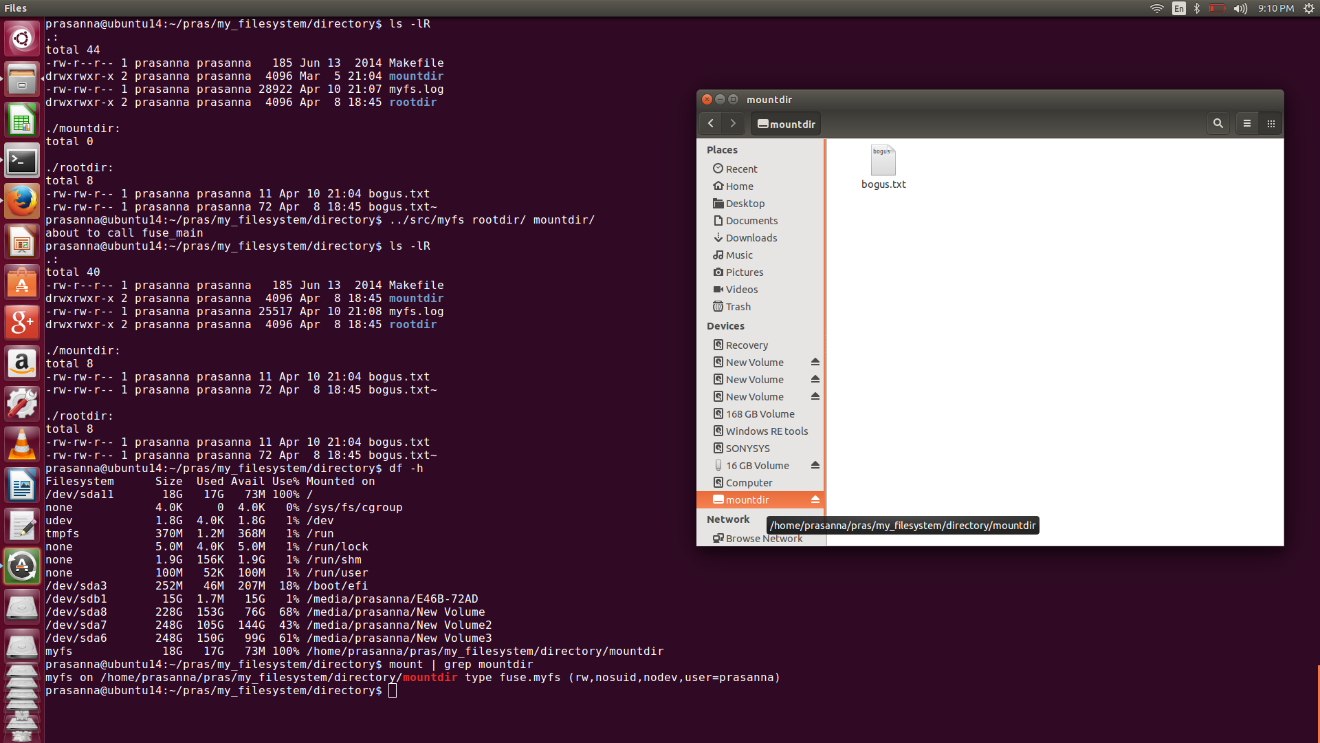
****

Fig.7.9 Implementation of MyFilesystem

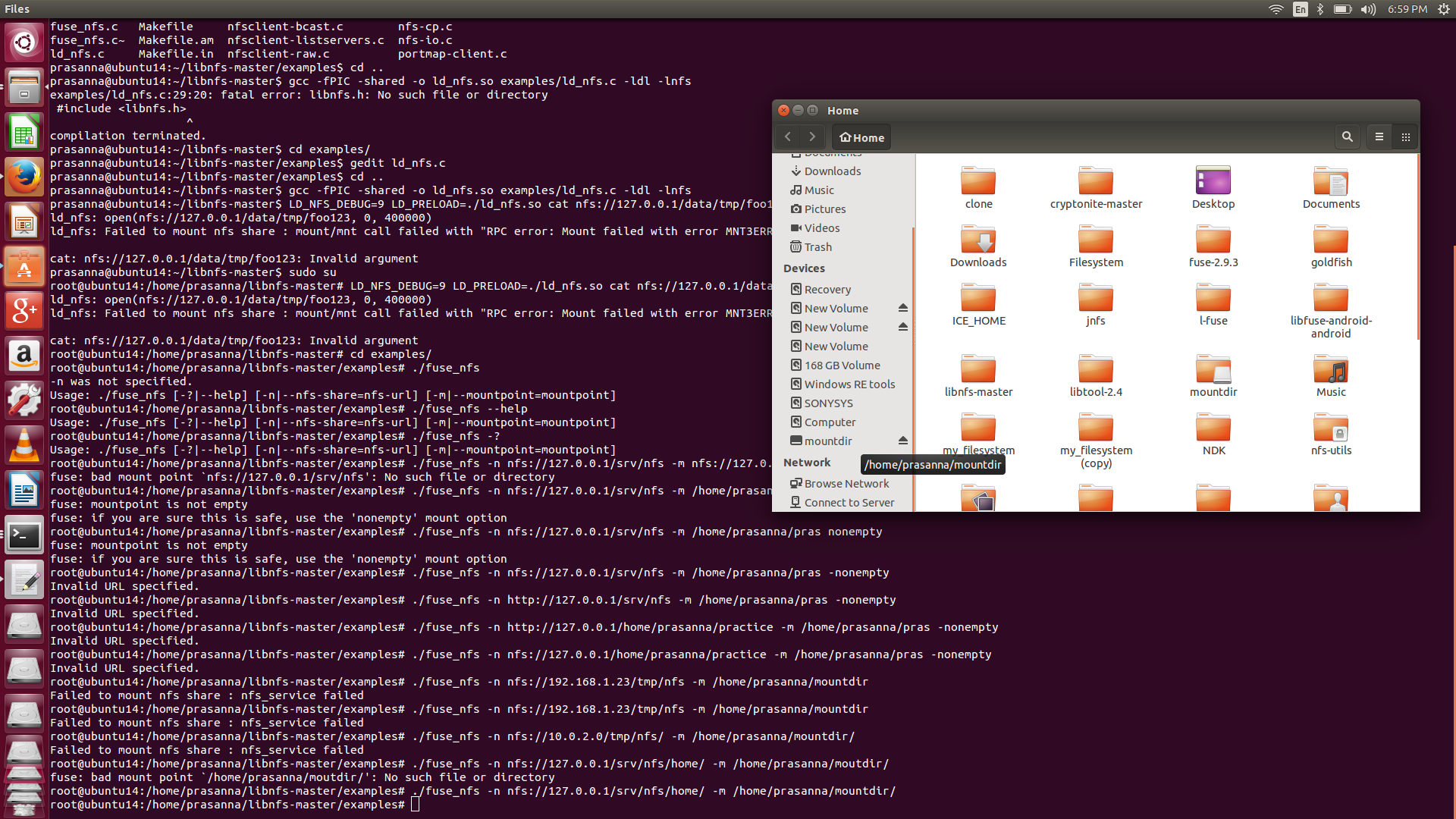
****

Fig.7.10 Implementation of Network File System.

**Chapter 9**

**DEPLOYMENT AND MAINTENANCE**

**9.1 INSTALLATION AND UN-INSTALLATION**

1. Pre requisites

* Any android device with specific version
* Minimum SDK 13 and maximum 21.

1. Installation Steps

1. Download and install jdk and sdk.

2. Download the kernel source

3. Changes made into the bash to setup JAVA environment variable

4. Run install and run adb and emulator from terminal.

5. Link the user level library libGL.so to run emulator and adb commands from the terminal.

6. Building of the kernel.

7. Install NFS and RPCBIND services.

**Chapter 10**

**CONCLUSION AND FUTURE SCOPE**

Using the concept of exploring the design mounting of android file system, i.e. loading the android middle-ware from the cloud. By observing various characteristics such as latency, battery consumption a cost model can be prepared. Once loaded into device memory and cached on the device, it won't consume more data than what apps anyway do by communicating with their in-cloud servers. Also, an alternative can be using FUSE to emulate NFS. Ability to code in User Space without worrying about the complexity of kernel programming stand it in good stead.

**Paper Publication Details:**

**Paper title**: Diskless Android Device

# Name of the conferences:

1. Global Science and Technology Forum journal of Engineering Technology.

2. International Conference on Mathematics, Physics, Chemistry and Engineering Science Research.

3. International Institute of Engineers (IIENG, London).

4. National Conference on Recent Innovations in Science Engineering and Technology (NCRISET-2015).

5. International Journal Of Engineering Research And General Science (IJERGS-2015).

6. Global Journal OF Engineering Science and Researches.

7. Hot Storage 2015 USENIX (Paper Submitted).

**Paper status**: Paper submitted and accepted.

**Chapter 11**

Appendix: Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Team Member | A member of the Team developing a project |
| NFS | Network File System |
| User | Reviewer or Developer |
| JDK | Java Development Kit |
| DVM | Dalvik Virtual Machine |
| ROOTFS | The Android root file system |
| Zygote | Child Process of the init() system call |

**Chapter 12**

**BIBLIOGRAPHY**

[1] “Dropbox - Home - Online backup, file sync and sharing made easy.” http://www.dropbox.com/.

[2] “Android Developers,” http://developer.android.com/index.html.

[3] Sooman Jeong1, Kisung Lee 2, and Youjip Won

I/O Stack Optimization for Smartphone, USENIX annual Technical Conference 2013.

[4] Eric Y. Chen and Mistutaka Itoh,

Virtual smartphone over IP, NTT Information Sharing Platform Laboratories, Tokyo 180-8585, Japan 2011.

[5]http://developer.android.com/tools/help/emulator.html

[6]https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-12-04

[7] http://elinux.org/Android\_Architecture

[8] Byung-Gon Chun and Petros Maniatis,

Cloud based Mobile Augmentation, Intel Research Berkeley 2010.

[9]. Eduardo Cuervo and Aruna Balasubramanian MAUI - offloads stuff to the Cloud to save energy/battery, University of Massachusetts Amherst.

[10]. Mark S. Gordon and D. Anoushe Jamshidi, COMET - migrates apps to Cloud, USA, 2012.

[11]**. S**oumya Simanta and Kiryonh Ha, Cloudlets: Mobile Code Offload in, Hostile Environments, USA, 2007

[12]. Allan Pierra Fuentes, Miguel Albalat Aguila and Cealys Alvarez Trujillo, Android Diskless System

[13].https://tldp.org/HOWTO/NCURSES-Programming- HOWTO/intro.html

[14], [15] http://wiki.emacinc.com/wiki/Booting\_with\_an\_NFS\_Root\_Filesystem.

[16] https://en.wikipedia.org/wiki/QEMU

[17], [18], [21] https://en.wikipedia.org/wiki/Filesystem\_in\_Userspace

[19], [20] Vishal Kanaujia and ChetanGiridhar, FUSE’ing Python for Development of Storage Efficient File System, 244-941-1-PB

[21], [22] https://fuse.sourceforge.net/

[23]https://code.**google**.com/p/cryptonite/