LNMIIT
The LNM Institute of Information Technology

FILE SYSTEM FOR IMAGES IN ANDROID

Instructor: **Prof. Gaurav Somani**

Names of the Team members:

Shubham Bansal **y10uc315**

Dipesh Palod **y10uc107**

Shivam Agarwal **y10uc304**

ABSTRACT:

The propose of the report is to give a brief idea of what we are working on, what the problems we faced during the project, how we solved it and what can be the future implementations we plan for. Managing a file system requires different processes like reading, writing, storage, deletion, retrieval, update and addressing.

The main problems we faced during the project were how to utilise the memory in best fashion thereby not reducing the fastest possible access and retrieval.

We tried different methods and algorithms each time with a better version than before.

For now the project works for the best possible conditions determined by us.

INTRODUCTION:

As the name suggests, we started the project with an aim to implement a file system for images in Android which manages the storage, allocation, retrieval and updating of image files.

*A **file system** is a means to organize data expected to be retained after a program terminates by providing procedures to store, retrieve and update data, as well as manage the available space on the device(s) which contain it. A file system organizes data in an*

efficient manner and is tuned to the specific characteristics of the device. A tight coupling usually exists between the operating system and the file system.

(Ref. http://en.wikipedia.org/wiki/File_system)

This file system would provide the user with easy handling of image files which includes the best possible allocations, fast searching and access to the files with all the important details.

So basically our motive is to manage the memory/ storage as best as possible and fastest access to the files.

ALGORITHM/METHOD OF IMPLEMENTATION:

Phase I :

During the project we found out different methods of building file system which include block-wise allocations and continuous allocations.

During the initial phase of the project we developed a file system using a block allocation method (each block of size 4kb).

In this method, an image was read and it was stored in blocks of size 4KB each. In each block the initial 6 bytes would contain the next block address in which the remaining data of the file was stored and so on. So an image would be scattered in the memory in thee form of blocks each pointing to the next.

This results in linked allocations of the blocks.

DRAWBACKS:

Each block uses 6 bytes for the address of the next block, thus resulting in memory wastage. (**3MB wastage in 2GB storage**)

SOLUTION:

To get rid of the drawbacks of the traditional approach or at least minimise them we tried a new algorithm which would use continuous allocation as well as block allocation method. During the process of overcoming the memory wastage problem, we realised that if we take the size of the image into account then there can be chances of improving the code.

We know that in a phone camera, the size of the image depends upon its resolution, colour etc and it has a minimum fixed size too.

So the next idea was to assume the images to be of size greater than equal to 300kb.

Phase II :

ASSUMPTION: Minimum Image Size: 300KB

In this method we allocated the first 300KB block in continuous and then break the rest of the part in divisions of 4KB block that would point to the next block using block-wise linked allocation. This would save space that we were first using for addresses in the previous case as now the initial 300KB block would contain only the next 4KB block's address. Now with every image, we are saving 444 bytes (296/4*6 bytes) that we were wasting with each 4KB block.

DRAWBACKS:

Before turning the idea into the code, we listed down the possible shortcomings of this method. It was then we realised that the algorithm for this may seem to be quite accurate for now but after a long time it would create trouble and result into a large amount of memory wastage.

For example,

1. Let's assume that the storage is almost full and containing a free space of only 20KB in a blocks of 4KB.

2. Then I delete a file of size 590KB from the storage. This would empty a full block of 300KB and the other 290KB in the form of 4KB blocks.

3. Now I store a file of size exactly 300KB which would occupy the recently freed 300KB block.

4. Now I am left with blocks of 4KB having a free space of (290+20) KB=310KB.

5. But now if I want to store a file of 305KB, then I am not allowed although there is enough free space. I don't have a 300KB block to allocate the image firstly.

Phase III :

To overcome the above problems we got on to a new idea which is as following:

Now we have all the blocks of 4KB each. Whenever an image file is read I search for the first free block in the storage. This first block would contain initial 6 bytes which will tell us the number of next continuous blocks and after these continuous blocks are allocated with the data then we have another 6 bytes which would contain the address of next block allocated which is not continuous.

SOFTWARES:

1. Eclipse (Java Integrated development toolkit)
2. ADT plugins for Eclipse and Android Virtual Device
3. Android SDK
4. Android NDK (For providing c/c++ interface with android apps)
5. gcc Compiler

CONCLUSION:

At the end of the project we understand that there is still a possibility that we can improve the code in the future. For now we came through different problems related to memory management and the access speed of the files and it was the best possible algorithm we could determine for now.

THANK YOU