**1. INTRODUCTION**

This project is about how the back end of a group messaging software works. It is a back end program which is dealt with concepts from scratch. This project shows how software in the back end is working and it was an attempt of replicating the real time systems present today.

The project was given to us during the course of my training in Mission R & D as a group project to design.

**1.1 Motivation:**

Using the messaging app/facility in the mobile has always made curious about how it works on the back end side of the system in the software. During my training course in Mission R & D, I was asked to build the back end of a group messaging system which was very similar on what I wanted to work on and was very interested on that side of the system.

This motivated me to proceed towards the project and also kept my tempo throughout the project consistent. The major concept that caught my eye on first sight is that we can design our own structure of message, group and comments and we have to store all of them in a binary file and use the Command User Interface to communicate between the users in a group.

**1.2 Problem Definition:**

In many shared environments, such as homes, hospitals or offices, the way messages are left for absent individuals is that they are written down on scrap paper, notepads or sticky notes and left somewhere for the absent person(s) to notice and read at a later time. Unfortunately, this can lead to communication problems because the messages that are left can be illegible, contain shorthand notations understood only by the message taker, or the message could even be overlooked altogether. Written messages poorly convey tone, and longer messages are cumbersome to compose. They are insecure when left in the open, which could be problematic if the messages contain sensitive information. Paper messages are not persistent unless filled properly by the message receiver. Solutions for the problems described above must be addressed by the message sender, receiver, or both, which adds weight to what should be a simple process.

**1.3 Objective of Project:**

* To be able to send messages to a group successfully.
* To be able to store the data of messages and groups in a file hard core.
* To be able to like and comment a message.
* To able to structure the groups, messages and comments into uniform blocks .
* To be able to comment on comments.
* To be able for any user to enter into any groups.

Finally the objective of the project is to give user friendly interface and give the look, feel and experience as close to the real time as possible.

**1.4 Limitations of Project:**

This project has many limitations when compared to the existing real time systems available. Some of them are listed below

* This project is limited only to the sending of messages and does not handle the deletion of messages.
* It does not provide user authentication i.e. it does not have the facility of checking for password.
* It does not have limit in the count of time each user can like a message or comment.

**2. LITERATURE SURVEY**

**2.1 Introduction:**

Group messaging system is one the most used kind of communication in different fields of the day to day life of the world. We generally call it as group chat now a days.

In group chat all the users can see all the messages of all the active users of that group, and can comment on them or like the message or comment of the message. In this system we can any number of groups and messages.

GMS can be beneficial to many parts of society including providing a more personal communication medium to today’s busy families and in university dormitories. It also provides a more persistent and secure way of leaving messages in a professional environment, such as a hospital, and a reception area in an office. It could be marketed as a solution to the current problems with pen and paper messages as well as a time saving device and a more personal way of communicating. GMS would be sold as a set of peripherals and software in a box in consumer electronics and computer retail stores.

The Group Messaging System (GMS) is a messaging system designed for use in ad-hoc groups. It is a centralized cloud-based service that allows people to communicate using capability based addressing. This document gives an overview of the software architecture of the prototype implementation of GMS.

**2.2 Proposed Solution:**

We propose a solution to the current problems with the pen and paper message process of communication that is beneficial to society and is accomplished by adding value to an existing PC or laptop. By adding an external interface in the form of a customized button pad, web camera, microphone, speakers and a fingerprint reader (optional – for security) and controlling software, we propose a Group Messaging System (GMS) to allow better communication between users of a shared environment.



The Group Messaging System is designed to replace messages left with pen and paper, by providing an easy to use system that allows leaving audio or audio/video messages to other users in the system, which can then be retrieved and reviewed from the recipient’s GMS inbox. In many ways GMS provides a solution for messaging that addresses all the issues previously discussed.

**2.3 Group Messaging System:**

The Group Messaging System (GMS) is a messaging system designed for use in ad-hoc groups. It is a centralized cloud-based service that allows people to communicate using capability based addressing. This document gives an overview of the software architecture of the prototype implementation of GMS (as discussed earlier).

In GMS in the proposed system is completely stored in a 1 GB binary file.

user

user

user

Group

Messenger

File

user

user

All the messages of the users are stored in the group structure and all the comments of the messages are stored in the message structure.

**3. Analysis**

**3.1 Software Requirements Specification:**

The software requirements for this project are discussed below in detailed. The project is mostly written or executed in C programming language. We use the following concepts of computer science

* C Language.
* i-Node (Index Node).
* Bit Vector (for storing the Meta Data).
* File System.

**3.1.1 Introduction To C Language:**

C is a general purpose, imperative computer programming language, supporting  [structured programming](https://en.wikipedia.org/wiki/Structured_programming), [lexical variable scope](https://en.wikipedia.org/wiki/Lexical_variable_scope) and [recursion](https://en.wikipedia.org/wiki/Recursion_(computer_science)), while a [static type system](https://en.wikipedia.org/wiki/Static_type_system) prevents many unintended operations. By design, C provides constructs that map efficiently to typical [machine instructions](https://en.wikipedia.org/wiki/Machine_instruction), and therefore it has found lasting use in applications that had formerly been coded in [assembly language](https://en.wikipedia.org/wiki/Assembly_language), including [operating systems](https://en.wikipedia.org/wiki/Operating_system), as well as various [application software](https://en.wikipedia.org/wiki/Application_software) for computers ranging from [super computers](https://en.wikipedia.org/wiki/Supercomputer) to [embedded systems](https://en.wikipedia.org/wiki/Embedded_system).

C was originally developed by [Dennis Ritchie](https://en.wikipedia.org/wiki/Dennis_Ritchie) between 1969 and 1973 at [AT&T Bell Labs](https://en.wikipedia.org/wiki/AT%26T_Bell_Labs), and used to re-implement the [UNIX](https://en.wikipedia.org/wiki/Unix) operating system. It has since become one of the most widely used programming languages of all time, with C [compilers](https://en.wikipedia.org/wiki/Compiler) from various vendors available for the majority of existing [computer architectures](https://en.wikipedia.org/wiki/Computer_architecture) and operating systems. C has been standardized by the [American National Standards Institute](https://en.wikipedia.org/wiki/American_National_Standards_Institute) (ANSI) since 1989 (see [ANSI C](https://en.wikipedia.org/wiki/ANSI_C)) and subsequently by the [International Organization for Standardization](https://en.wikipedia.org/wiki/International_Organization_for_Standardization) (ISO).

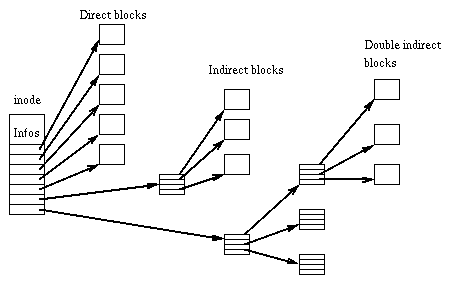
**3.1.2 i-Node (Index Node)**

In a [Unix-style file system](https://en.wikipedia.org/wiki/Unix_filesystem), an index node, informally referred to as an i-node, is a [data structure](https://en.wikipedia.org/wiki/Data_structure) used to represent a file system object, which can be one of various things including a file or a directory. Each i-node stores the attributes and disk block location (of the file system object's data. File system object attributes may include manipulation [metadata](https://en.wikipedia.org/wiki/Metadata) , as well as owner and [permission](https://en.wikipedia.org/wiki/File_system_permissions) data.

**Key Features**

### Fixed logical block size

The structure is partially illustrated in the diagram accompanying this article. The structure allows for i-nodes to describe very large files in file systems with a fixed logical block size. Central to the mechanism is that blocks of addresses (also called indirect blocks) are only allocated as needed. For example, a 12-block file would be described using just the i-node because its blocks fit into the number of direct pointers available. However, a 13-block file needs an indirect block to contain the thirteenth address.



**Ease of data location**

The i-node pointer structure not only allows for files to easily be allocated to non-contiguous blocks, it also allows the data at a particular location inside a file to be easily located. This is possible because the logical block size is fixed. For example, if each block is 8 KB, file data at 120 to 128 KB would be pointed to by the third pointer of the first indirect block (assuming twelve direct pointers in the i-node pointer structure).

**Indirect blocks**

Unlike the i-nodes, which are fixed in number and allocated in a special part of the file system, the indirect blocks may be of any number and are allocated in the same part of the file system as data blocks. The number of pointers in the indirect blocks is dependent on the block size and size of block pointers. Example: with a 512 byte block size, and 4 byte block pointers, each indirect block can consist of 128 (512 / 4) pointers.

The i-node structure used to have 11 or 13 pointers but most modern file systems use 15 pointers stored in the data structure. For the case where there are 12 pointers in the data structure, the pointers are:

* Twelve points that directly point to blocks containing the data for the file. These are called direct pointers.
* One single indirect pointer. This pointer points to a block of pointers that point to blocks containing the data for the file.
* One doubly indirect pointer. This pointer points to a block of pointers that point to other blocks of pointers that point to blocks containing the data for the file.
* One triply indirect pointer. This pointer points to a block of pointers that point other blocks of pointers that point to other blocks of pointers that point to blocks containing the data for the file.

**3.1.3 Bit Vector**

A bit-vector can be written as the sequence of bits contained in the string, preceded by #\*; any delimiter character, such as whitespace, will terminate the bit-vector syntax. For example:

#\*10110 ;A five-bit bit-vector; bit 0 is a 1

#\* ;An empty bit-vector

The bits notated following the #\*, taken from left to right, occupy locations within the bit-vector with increasing indices. The leftmost notated bit is bit-vector element number 0, the next one is element number 1, and so on.

The function prin1 will print any bit-vector (not just a simple one) using this syntax, but the function read will always construct a simple bit-vector when it reads this syntax.

A bit array (also known as bitmap, bit set, bit string, or bit vector) is an array data structure that compactly stores bits. It can be used to implement a simple set data structure. A bit array is effective at exploiting bit-level parallelism in hardware to perform operations quickly.

## Source Code

This bitvector code consists of three files:

[types.h](http://www.csd.uwo.ca/~jamie/BitVectors/types.h) (about 1 Kb):

Definitions of the variable types used in the other files.

[bitarr.c](http://www.csd.uwo.ca/~jamie/BitVectors/bitarr.c) (about 24 Kb):

Functions that encapsulate an implementation of a bitvector abstract data type.

[bitarr.h](http://www.csd.uwo.ca/~jamie/BitVectors/bitarr.h) (about 2 Kb):

Function prototypes for the functions in bitarr.c that have external linkage.

The original code is (c) copyright 1995 by Miller Freeman, Inc.

**3.1.4 File System:**

In [computing](https://en.wikipedia.org/wiki/Computing), a file system is used to control how data is stored and retrieved. Without a file system, information placed in a storage area would be one large body of data with no way to tell where one piece of information stops and the next begins. By separating the data into individual pieces, and giving each piece a name, the information is easily separated and identified. Taking its name from the way paper-based information systems are named, each group of data is called a "[file](https://en.wikipedia.org/wiki/Computer_file)". The structure and logic rules used to manage the groups of information and their names, is called a "file system".

There are many different kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some file systems have been designed to be used for specific applications. For example, the [ISO 9660](https://en.wikipedia.org/wiki/ISO_9660) file system is designed specifically for optical discs.

**Aspects of File system**

**Space Management:**

File systems allocate space in a granular manner, usually multiple physical units on the device. The file system is responsible for organizing [files](https://en.wikipedia.org/wiki/Computer_file) and [directories](https://en.wikipedia.org/wiki/Directory_(file_systems)), and keeping track of which areas of the media belong to which file and which are not being used.

[File system fragmentation](https://en.wikipedia.org/wiki/File_system_fragmentation) occurs when unused space or single files are not contiguous. As a file system is used, files are created, modified and deleted. When a file is created the file system allocates space for the data. Some file systems permit or require specifying an initial space allocation and subsequent incremental allocations as the file grows. As files are deleted the space they were allocated eventually is considered available for use by other files. This creates alternating used and unused areas of various sizes. This is free space fragmentation.

### 

### Filenames:

A filename (or file name) is used to identify a storage location in the file system. Most file systems have restrictions on the length of filenames. In some file systems, filenames are not [case sensitive](https://en.wikipedia.org/wiki/Case_sensitivity) in others, filenames are case sensitive.

Most modern file systems allow filenames to contain a wide range of characters from the [Unicode](https://en.wikipedia.org/wiki/Unicode) character set. However, they may have restrictions on the use of certain special characters, disallowing them within filenames; those characters might be used to indicate a device, device type, and directory prefix, file path separator, or file type.

### Directories:

### File systems typically have directories (also called folders) which allow the user to group files into separate collections. This may be implemented by associating the file name with an index in a [table of contents](https://en.wikipedia.org/wiki/Table_of_contents) or an [i-node](https://en.wikipedia.org/wiki/Inode) in a [Unix-like](https://en.wikipedia.org/wiki/Unix-like) file system. Directory structures may be flat (i.e. linear), or allow hierarchies where directories may contain subdirectories. The first file system to support arbitrary hierarchies of directories was used in the [Multics](https://en.wikipedia.org/wiki/Multics" \o "Multics) operating system.

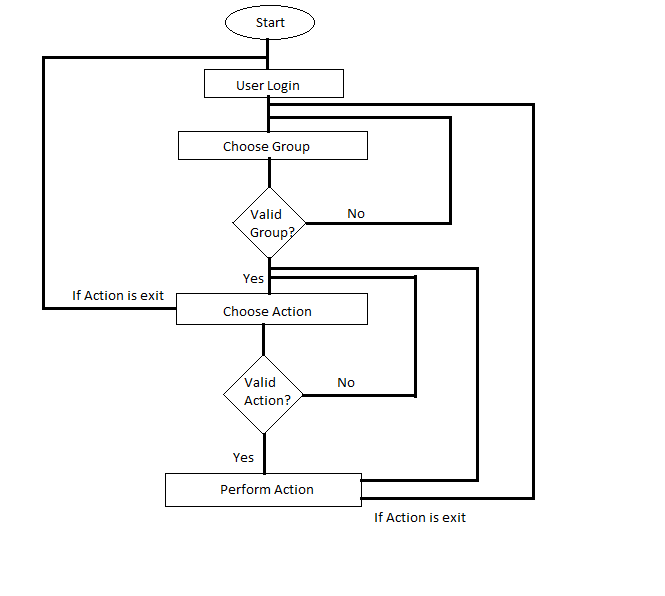
### Metadata:

### The [length](https://en.wikipedia.org/wiki/File_size) of the data contained in a file may be stored as the number of blocks allocated for the file or as a [byte](https://en.wikipedia.org/wiki/Byte) count. The [time](https://en.wikipedia.org/wiki/System_time) that the file was last modified may be stored as the file's timestamp. File systems might store the file creation time, the time it was last accessed, the time the file's [metadata](https://en.wikipedia.org/wiki/Metadata) was changed, or the time the file was last backed up. Other information can include the file's [device type](https://en.wikipedia.org/wiki/Device_file) , its owner [user ID](https://en.wikipedia.org/wiki/User_ID) and [group ID](https://en.wikipedia.org/wiki/Group_ID), its [access permissions](https://en.wikipedia.org/wiki/File_system_permissions) and other [file attributes](https://en.wikipedia.org/wiki/File_attribute).

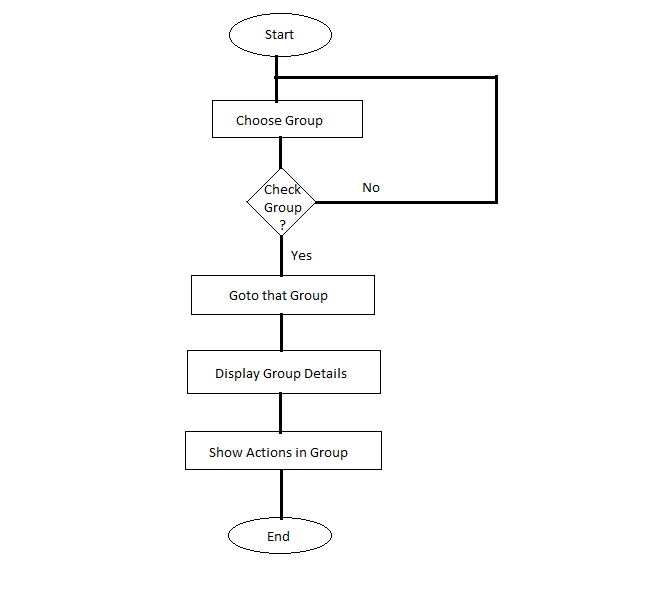
**3.2 Hardware Components Required**

* 1GB External Memory.
* C language compiler.
* 512 MB RAM (Random Access Memory).
* Multiprocessor.
* Greater than 2 GB ROM(Read Only Memory)/Secondary Memory.

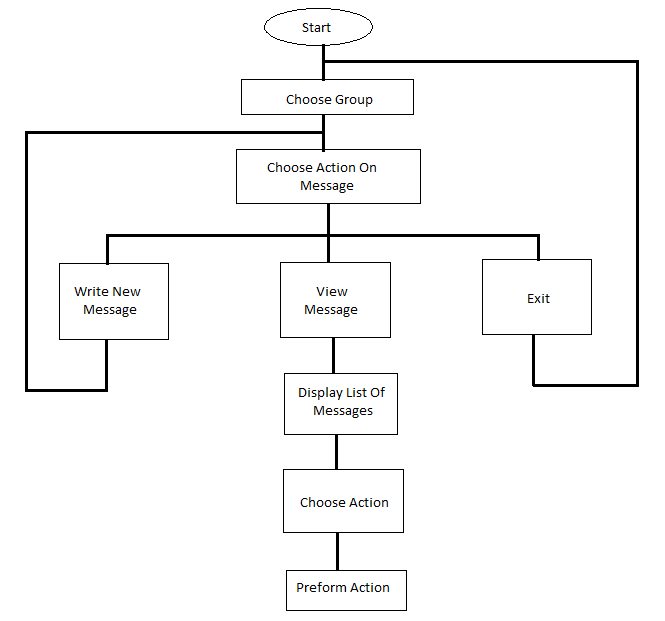
**3.3 Flowcharts**



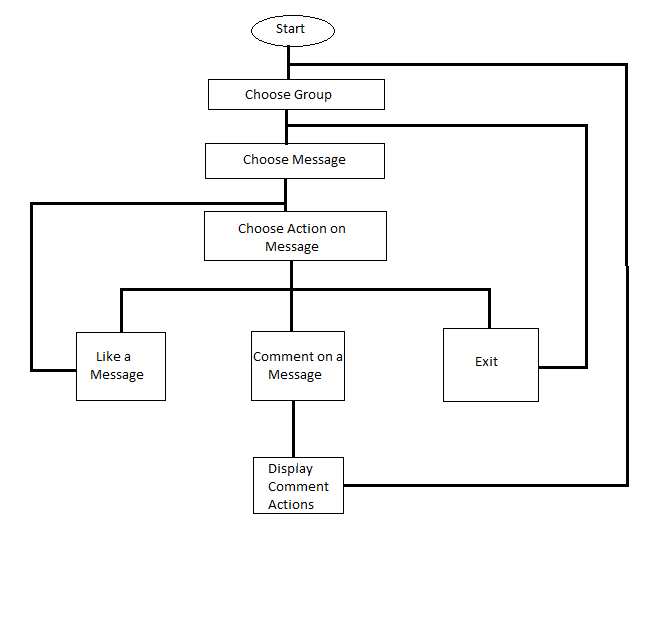
The above flow chart is the complete main flow of the program. It shows the general and overall view of the flow. The detail and elaborate flow of the functions of the program are given late



The above flow chart represents the flow of the user when they enter the group and try to perform any action. A group is where any user can login and message in the group and can also like and comment on any message.



The above flow chart represents how the flow of the message structure in the program. This project has actions like viewing messages of the group, commenting the message and liking the message. The flow of comments is discussed later.



The above flow chart shows flow of the comments structure in the system. We can like a comment in comment actions and also comment on a comment.

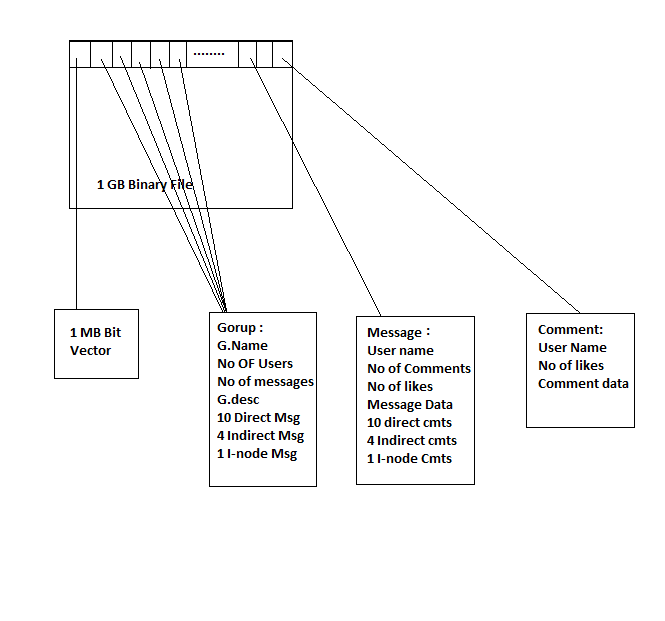
**4. DESIGN**

**4.1 Design Model**

This project is designed structure is designed according the file size given and structured in an way that all the block of memory are uniform to avoid fragmentation in the memory and for efficient and optimal use of memory.

In this project we use 1GB file to store the data of the system and we use a bit vector to for space tracking. The structure of the design is give below:

**File storage structure of the System**



The designing of the structure of the system is divided into three modules. These modules are designed such that every structure is able to fit in the file and the structure in the block size of 128 Bytes of each. They are listed below:

* Group Structure
* Message Structure
* Comment Structure

**Group Structure:**

The group in this project is 128 Bytes the size and structure of the Group is given below:

|  |  |
| --- | --- |
| **Attributes** | **Size of the attribute** |
| Group Name | 10 Bytes |
| No of Users | 4 Bytes |
| No of Messages | 4 Bytes |
| Group Description | 50 Bytes |
| Direct Message Address | 10 x 4 Bytes |
| Indirect Message Address | 4 x 4 Bytes |
| I-node Message Address | 4 Bytes |
| **Total** | **128 Bytes** |

**Table-1**

This structure is specifically designed to fit and function only in this system as the file structure is designed personally it cannot be reused in other pre-defined system.

The structure of the group in the program is given as follows:

**struct groups**

**{**

**char gname[10];**

**char desc[50];**

**int num\_user, num\_mgs;**

**int direct\_mgs[10];**

**int indirect\_mgs[4];**

**int d\_indirect\_mgs;**

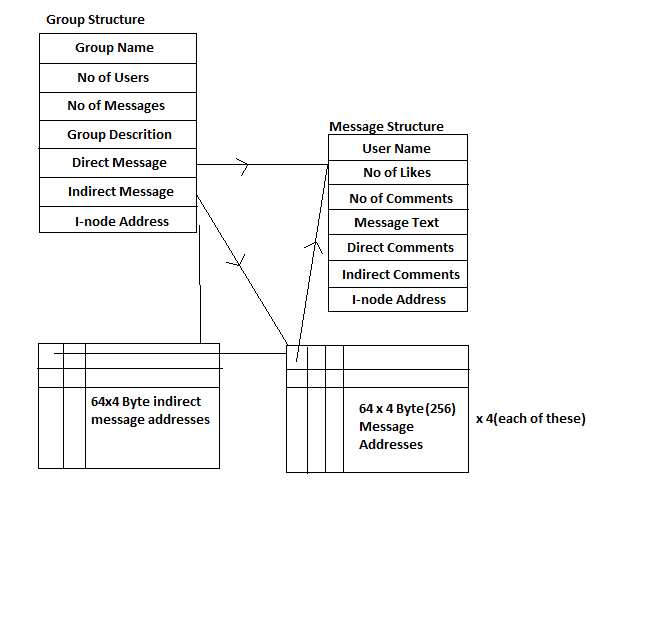
**}**

Where size of char is 1 Byte and the size of int is 4 Bytes. There are 10 direct message addresses which are int, there are 4 indirect addresses and one I node/indirect indirect address for messages.

The groups in this system are static i.e. we cannot add or delete groups and cannot edit the name or description of the group. The user has to message in these given groups only they cannot add/create new group as of their own.

The actions that can be performed on group and in a group by the user are given below:

* Login into a group.
* Read the group description.
* Write a new message.
* View the last 10 messages in the group.



**Message Structure:**

The Message in this project is 256 Bytes the size and structure of the Message is given below:

|  |  |
| --- | --- |
| **Attributes** | **Size of the attribute** |
| User Name | 8 Bytes |
| Message Text | 180 Bytes |
| No of comments | 4 Bytes |
| No of Likes | 4 Bytes |
| Direct Comment Address | 10 x 4 Bytes |
| Indirect Comment Address | 4 x 4 Bytes |
| I-node Comment Address | 4 Bytes |
| **Total** | **256 Bytes** |

**Table-2**

This structure is specifically designed to fit and function only in this system as the file structure is designed personally it cannot be reused in other pre-defined system. The message structure occupies two 128 Bytes blocks in the 1 GB binary file.

The structure of the message in the program is given as follows:

**struct messages**

**{**

**char user[8];**

**char message[180];**

**int likes, no\_of\_comments;**

**int direct\_comment[10];**

**int indirect\_comment[4];**

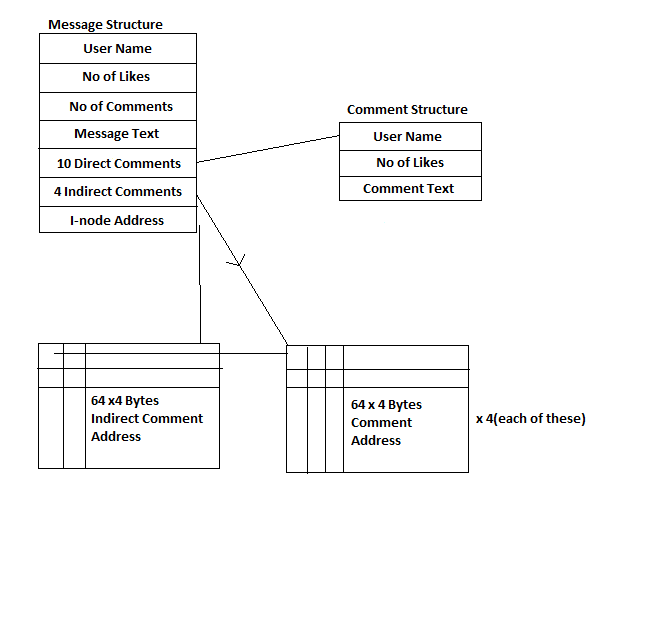
**int d\_indirect\_comment;**

**}**

Where size of char is 1 Byte and the size of int is 4 Bytes. There are 10 direct comment addresses which are int, there are 4 indirect addresses and one I node/indirect indirect address for comments.

The actions that can be performed on message and in a message structure by the user are given below:

* Read a message.
* Can see which user has written what message.
* Can comment on any message.
* Can like a message.
* Can see how many likes and comments are there for each message.



**Comment Structure:**

The Comment in this project is 128 Bytes the size and structure of the Comment is given below:

|  |  |
| --- | --- |
| **Attributes** | **Size of the Attribute** |
| User Name | 8 Bytes |
| Comment Text | 116 Bytes |
| No of Likes | 4 Bytes |
| **Total** | **128 Bytes** |

**Table-3**

The structure of the message in the program is given as follows:

**struct comments**

**{**

**char user[8];**

**int likes;**

**char comment[116];**

**}**

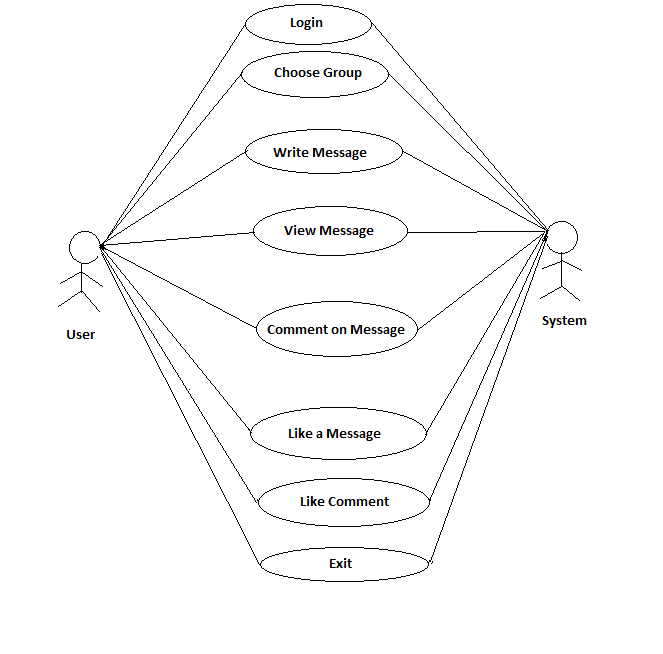
Where size of char is 1 Byte and the size of int is 4 Bytes.

The actions that can be performed on comment and in a comment structure by the user are given below:

* Read a comment.
* Can see which user has written what comment.
* Can comment on any comment.
* Can like a comment.
* Can see how many likes are there for each comment.

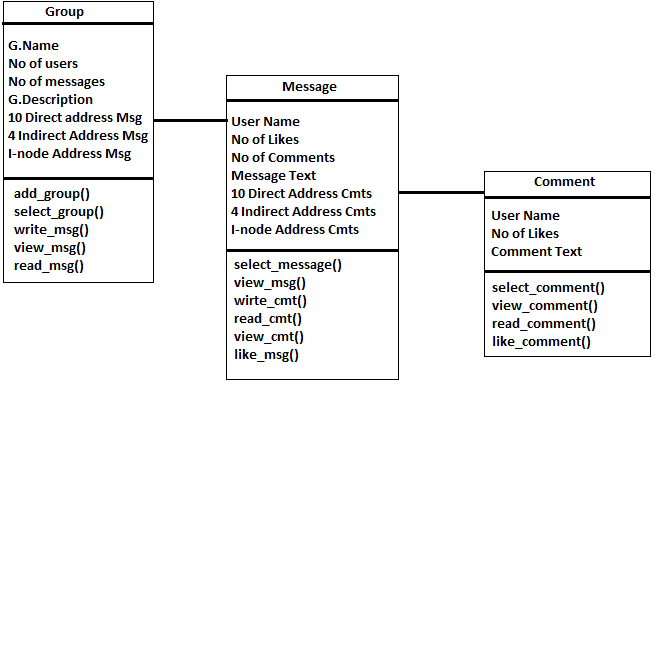
**4.2** **UML diagrams:**

**Use Case Diagram:**

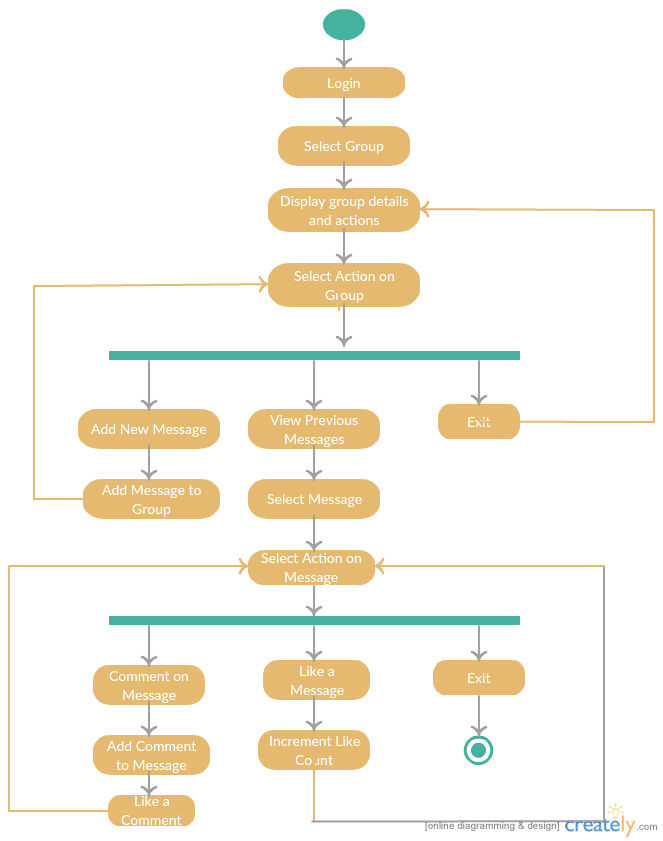


The above describes the use case diagram of the group messaging system, where the user can access the system with different actions.

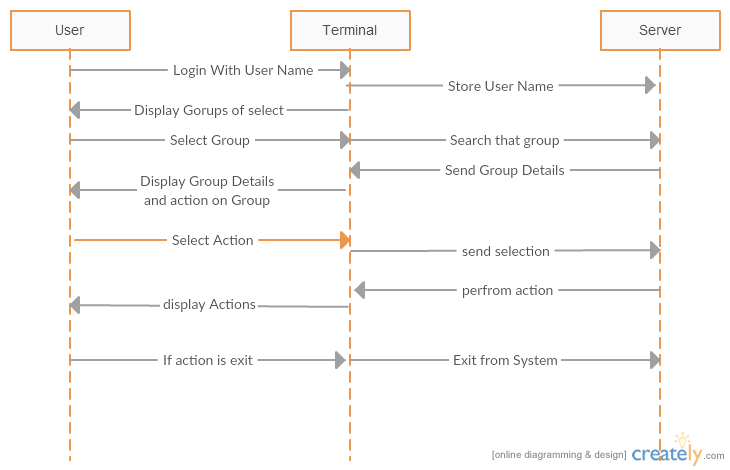
**Class Diagram:**



**Activity Diagram:**



**Sequence Diagram:**



**5. IMPLEMENTATION & RESULTS**

**5.1 Explanation of Key functions:**

The key functions of the program are given below:

* Byte vector
* Create file
* Write message
* View message
* Write comment
* View comment

**Byte Vector:**

This function is used to find which block in the file is empty and to find the corresponding block required to retrieve for actions to be performed. The function is coded as follows:

int byte\_vector\_write(FILE \*fp, int flag)

{ char b = "0", \*ch, \*h = "1"; int i, k;

rewind(fp);

for (i = 0; i < (1024 \* 1024 \* 8); i++)

{ ch = getc(fp);

if (ch == '0') break;

}i -= 2;

fseek(fp, -1, 1);

if (flag == 1)

for (k = 0; k < 2; k++)

fwrite((char\*)h, strlen(h), 1, fp);

else

fwrite((char\*)h, strlen(h), 1, fp);

return i;

}

**Create File:**

This function is used to create the 1 gb file from the program itself using the file operating pre-defined functions. The function for this is given as follow:

void start()

{

int i;

char \*user;

FILE \*fp;

if (fopen("myfile.bin", "r") == NULL)

{

fp = fopen("myfile.bin", "wb");

fseek(fp, 1024 \* 1024 \* 1024, SEEK\_SET);

fputc('\n', fp);

fclose(fp);

bit\_vector\_init();

create\_groups();

}

}

**Write Message:**

This function deals with the adding of the message text, increment the no of messages count, giving the user name to the message, initializing the no of comments, no of like count in the message and group structure.

**View Message:**

This function deals with the reading of message it involves the retrieving of the message address from the group structure, retrieving the message structure from the address attained from the group and displaying the message with no of likes and no of comments.

**Write Comment:**

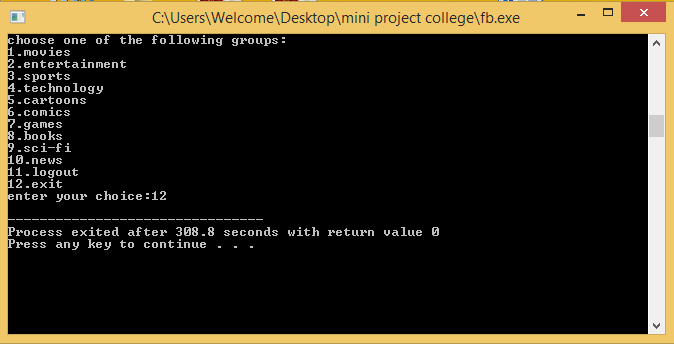
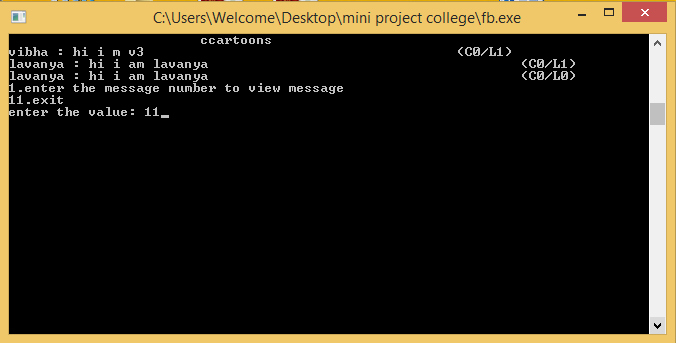
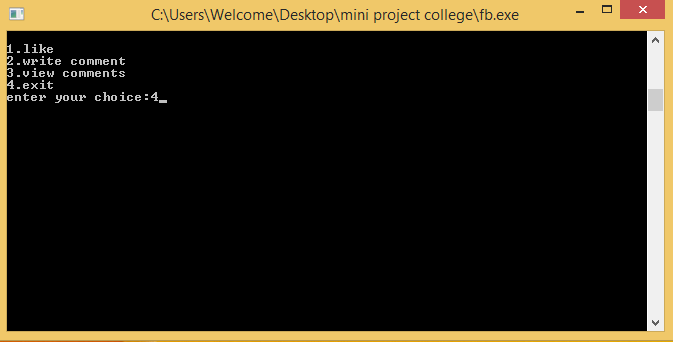
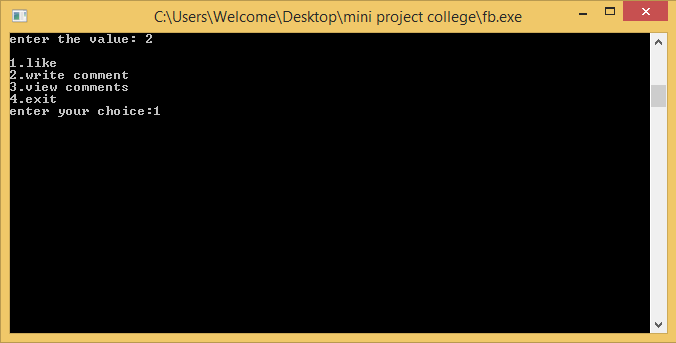
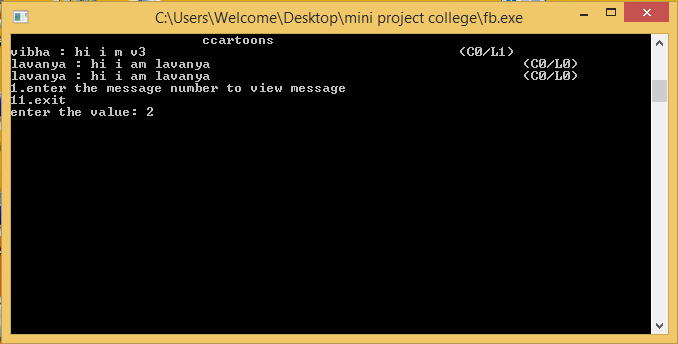
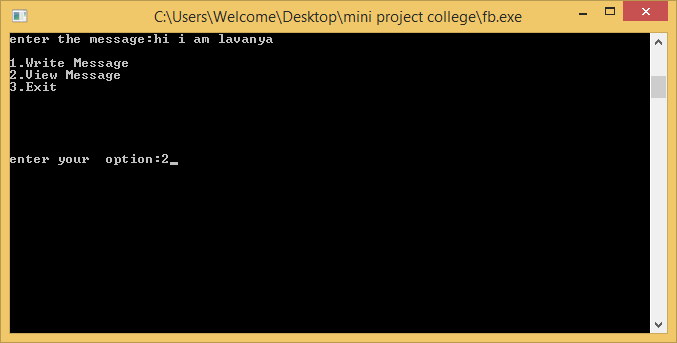
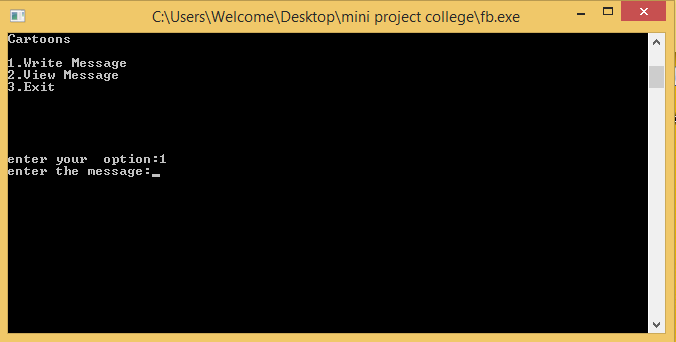
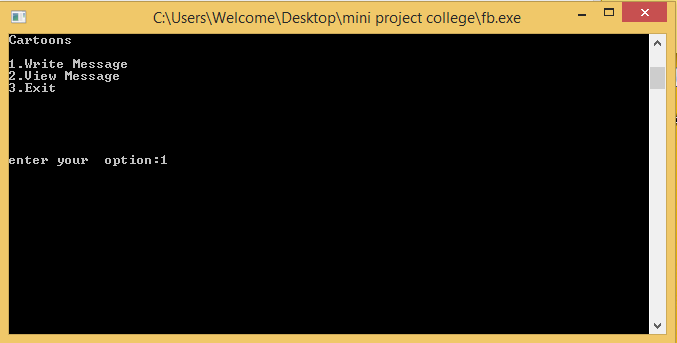
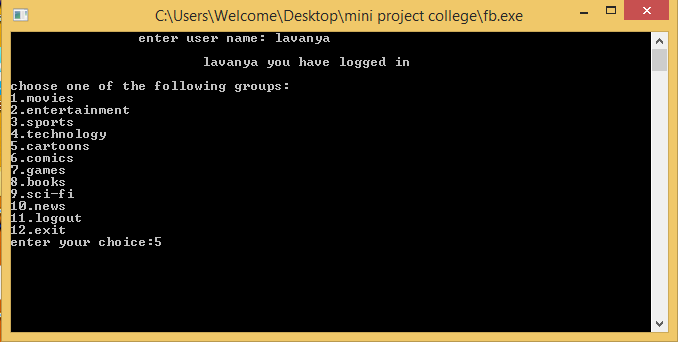
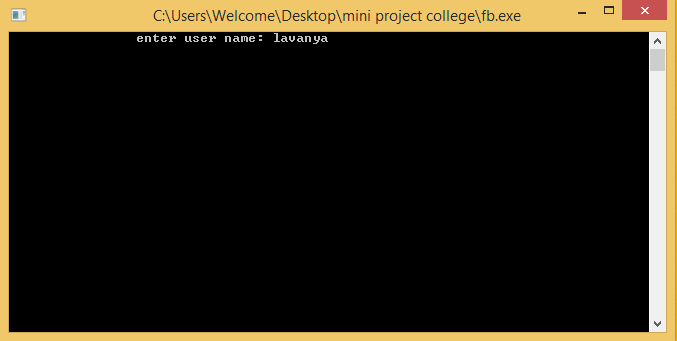
This function deals with the adding of the comment text, increment the no of comment count, giving the user name to the comment, initializing the no of comments, no of like count in the comment and message structure.

**View Comment:**

This function deals with the reading of comment it involves the retrieving of the comment address from the message structure, retrieving the comment structure from the address attained from the message and displaying the comment with no of likes and User name.

**5.2 Method of Implementation**

**5.2.1 Output Screens:**



**6. CONCLUSION**

The Group Messaging System is now a fully functioning, feature complete prototype as per the intended design. After arriving at the idea of the GMS, we developed the requirements analysis document (RAD), followed by the system design document.

The proposed system can be improvised and be updated with the current technologies. Including the delete option into the program makes it more user feasible, giving the additional option of user creating the groups would be a good development in the project, which would be included later.

This system can be customized and given to the application builders as a blue print to build their own application with their own settings. It is an attempt to replicate the real time application.

**REFERENCES**

* Wikipedia Website.
* Multiple different sources from the web.
* http.www.creately.com(for UML Diagrams).