

Application for distributing jobs among computers connected in LAN in Linux/GNU Environment

A BACHELOR'S THESIS

*Submitted in partial fulfilment
of the requirements for the award of the degree*

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

(B.Tech. in IT)

Submitted by

Deepak Bhorla (IIT2009112)

Under the Guidance of:

Dr. T. Pant

Asst. Professor
IIIT-Allahabad



**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
ALLAHABAD – 211 012 (INDIA)**

May, 2013

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this thesis entitled “**Application for distributing jobs among computers connected in LAN in Linux/GNU Environment**”, submitted in the partial fulfilment of the degree of Bachelor of Technology (B.Tech.), in Information Technology at Indian Institute of Information Technology, Allahabad, is an authentic record of my original work carried out under the guidance of **Dr. T. Pant**. Due acknowledgements have been made in the text of the thesis to all other material used. This thesis work was done in full compliance with the requirements and constraints of the prescribed curriculum.

Place: Allahabad
Date: 6/5/2013

Deepak Bhoria
IIT2009112

CERTIFICATE FROM SUPERVISOR

I do hereby recommend that the thesis work prepared under my supervision by Deepak Bhoria titled “Application for distributing jobs among computers connected in LAN in Linux/GNU Environment” be accepted in the partial fulfillment of the requirements of the degree of Bachelor of Technology in Information Technology for Examination.

Date: 6/5/2013
Place: Allahabad
Committee for Evaluation of the Thesis

Dr. T. Pant
Asst. Professor., IIITA

ACKNOWLEDGEMENTS

As understanding of the study like this is never the outcome of the efforts of a single person, rather it bears the imprint of a number of persons who directly or indirectly help us in completing the present study. I would be failing in our duty if I don't say a word of thanks to all those whose sincere advice made this documentation of topic a real educative, effective and pleasurable one.

It is my privilege to study at Indian Institute of Information Technology, Allahabad where students and professors are always eager to learn new things and to make continuous improvements by providing innovative solutions. I am highly grateful to the honourable **Dr. M. D. Tiwari, Director IIIT-Allahabad**, for his ever helping attitude and encouraging all of us to excel in studies.

Regarding this thesis work, first and foremost, I would like to heartily thank my supervisor **Dr. T. Pant** for his able guidance. His fruitful suggestions, valuable comments and support were an immense help for me. In spite of his hectic schedule he took pains, with smile, in various discussions which enriched me with new enthusiasm and vigor. I am especially indebted to my parents and friends for their love and support.

Place: Allahabad

Date: 5 /5 /2013

Deepak Bhoria
(IIT2009112)

B Tech Final Year, IIITA

ABSTRACT

In this project an application to distribute jobs among computers connected in LAN is created. Firstly it is implemented for Linux/GNU systems but it can further be extended for Windows and UNIX machines in future. Generally when we create program; we create it keeping single machine in mind. But today we are surrounded by lots of computing devices which remain idle most of the time. This project provides a mechanism by which a single machine can harness the computing power of other neighbouring machines.

This project assumes that neighbouring machines are connected with each other through LAN. So we can use POSIX API for socket programming to communicate over network. First we seek for neighbour machines that have CPU and Memory usage lower than defined by the owner of the machine. Then we create a list of all the neighbouring machines which are ready to provide their resources for our job. After that we send our instructions (along with data) which we want to be executed by neighbouring machine. The neighbouring machine upon completion returns back the result of our task.

Table of Contents

1. Introduction.....	6
1.1 Currently existing technologies.....	6
1.2 Analysis of previous research in this area.....	8
1.3 Problem definition and scope.....	9
1.4 Formulation of the present problem.....	10
1.5 Organization of the thesis.....	10
2. Description of Hardware and Software Used.....	11
2.1 Hardware.....	11
2.2 Software.....	11
3. Theoretical Tools – Analysis and Development.....	16
3.1 Types of communication available.....	16
3.2 Socket Programming usage.....	16
3.3 Multiple connection handling using threads.....	17
3.4 System call & functions from libraries.....	18
4. Implementation.....	19
4.1 Important Messages used for communication.....	19
4.2 Work flow diagram.....	22
4.3 Important header files and their functions.....	22
4.4 GUI and handling user input.....	25
5. Testing and Analysis.....	27
5.1 Testing.....	27
5.2 Analysis.....	27
5.3 Results.....	27
6. Conclusions.....	32
7. Recommendations and Future Work.....	33
Appendix	34
References.....	51

1. INTRODUCTION

Today we are surrounded by lots of computing machines like laptops, desktops etc. which are connected to each other through network like LAN. If we have a heavy task to do; so why not divide the task among idle neighbouring machines. This will increase the efficiency of system. But we cannot go to each machine and see how much CPU, Memory or other resources (like hard-disk space, network bandwidth) are being currently utilised by machine. Because firstly we may not have a user account on neighbouring machines (say your friend's system in lab); secondly it will be a tedious task to check each machine for its resource consumption.

We try to solve this issue by running a demon on each neighbouring machine which replies for any request about its system's current status of resources.

After analysing reply from neighbours we can send job to neighbouring machines which will execute the job and return result to us. First we shall define some terms like:

Neighbour: is a machine connected to network.

Needy: is the machine which wants help from neighbouring machines.

Volunteer: is the machine which replies that it is ready to help.

We have to create an application which provides us a list of neighbouring machines which are willing to help us. We can then send jobs to neighbouring machine and let it complete. As a programmer you don't have to worry about on which machine your program will be executed. If your program is not present on other machine you can compile the program on destination machine and executable will be generated according to destined machine architecture and now you can send instructions to other machine.

Instructions to other machine will be sent using shell script. Nothing is better than using pre-existing utility and executable programs for tasks.

1.1 Currently existing methods and technologies:

SSH:

Secure Shell (SSH)[1], sometimes known as Secure Socket Shell, is a UNIX-based command interface and protocol for securely getting access to a remote computer. It is widely used by network administrators for secure data communication, remote shell services, command execution or to control Web and other kinds of servers remotely. It helps to communicate between two networked computers via a secure channel over an insecure network. SSH is actually a suite of three utilities - *slogin*, *ssh*, and *scp* - that are secure versions of the earlier UNIX utilities *rlogin*, *rsh*, and *rcp*. It was designed as replacement for *Telnet* and other insecure remote shell protocols such as *rsh* which sends information like passwords in plaintext which can be intercepted and disclosed via packet analysis. Both ends of the client/server connection are authenticated using a digital certificate, and passwords are protected by being encrypted. Encryption algorithms include *Blowfish*, *DES* and *IDEA* and *IDEA* is the default algorithm for encryption. The encryption used by ssh is to provide confidentiality and integrity over an unsecured network such as Internet.

SSH uses RSA public key cryptography for both connection and authentication.

SSH client and server communication takes place as follow:

During SSH client and server configuration, two configuration files are installed: a client configuration file, which is read by an SSH client process when the SSH command is invoked; and the server configuration file, which is read by an SSH server process when a connection request arrives from an SSH client.

When TCP/IP Service is started on an SSH server host, the auxiliary server creates a listening socket for SSH. The SSH server is now ready to accept a remote connection request. When you execute an SSH command on a remote client host, the SSH client is initiated. The client reads the configuration file and initiates a TCP connection to a server host using the specified destination port. On an SSH server host, the auxiliary server creates a copy of the server process, which reads the server's configuration file.

To establish a secure connection[1]:

1. The SSH client and server exchange information about supported protocol versions. This enables different implementations to overcome incompatibilities.
2. The SSH server initiates a host public key exchange with the client to prove its identity. Each server host has a public and private key pair, which is created during the SSH server configuration. This pair uniquely identifies the server host. The first time an SSH client connects to a server, SSH prompts the user to accept a copy of the server's public host key.
3. If the user response is yes, SSH copies the server's public host key to the SSH client host. The client host uses this public host key to authenticate the SSH server on subsequent connections.
4. If, during subsequent connection attempts, the SSH client detects that the SSH server's host key differs from the one stored on the client and warning message is displayed.
5. The SSH client and server negotiate session parameters by exchanging information about supported parameters, including authentication methods, hash functions, and data compression methods.
6. The SSH client and server develop a shared (symmetric) session key to encrypt data using a key exchange method. When both the client and server know the secret data encryption key, a secure connection is established and the client and server can exchange data securely. The session key expires either when the session ends or when the session timer expires and a rekey operation is done.
7. After the SSH client and server authenticate each other, the session is ready to authenticate the user by applying one or more of the authentication methods.
8. The SSH server checks the user's identity. The user must have a valid user account and home directory on the server. If the server fails to authenticate the user, the server refuses the connection.
9. After SSH authenticates the user's identity, the actual secure data transfer between client and server occurs.
10. The SSH server runs in a loop, accepting messages from the client, performing required actions, and returning reply messages to the client.
11. When the user closes the connection, the server process terminates. The auxiliary server continues to listen for new SSH connection requests.

Remote Desktop Login:

Remote Desktop login[2] enables you or another person to view and interact with your desktop environment from another computer system either on the same network or over the internet. This is useful if you need to work on your computer when you are away from your desk while traveling or sitting in a coffee shop. Windows allows this using RDP(remote desktop protocol) while Linux does this using VNC(virtual network computing). There are both secure and insecure methods for remote access. It is advisable to use secure method over an insecure network.

In computing, Virtual Network Computing (VNC)[3] is a graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer. It transmits the keyboard and mouse events from one computer to another, relaying the graphical screen updates back in the other direction, over a network.

In Linux we can use VNC for remote display system which allows the user to view the desktop of a remote machine anywhere on the internet. It can also be directed through SSH for security.

Basically you install VNC server on the server and install client on your local PC. Setup is extremely easy and server is very stable. On client side, you can set the resolution and connect to IP of VNC server. It can be a bit slow compared to Windows remote desktop and also has the tendency to take more time refreshing over low-bandwidth links.

VNC is platform-independent – a VNC viewer on one operating system may connect to a VNC server on the same or any other operating system. There are clients and servers for many GUI-based operating systems and for Java. Multiple clients may connect to a VNC server at the same time. Popular uses for this technology include remote technical support and accessing files on one's work computer from one's home computer, or vice versa.

1.2 Analysis of previous research in this area

SSH cons:

1. Requires username and password of remote computers for login.
2. Many people will not give their login credentials to you.
3. Difficult to remember usernames and passwords for all neighbouring users.
4. If you need to distribute n jobs you need to open n terminals and check each one which is very hectic.
5. It gives you complete shell of other person you can do anything harmful.
6. Doesn't provide GUI, it's a command line application.
7. Sending and Receiving data is through commands which are not fully transparent to user.

VNC cons:

1. Requires username and password of remote computer for login.
2. A lot of people will not give their account credentials to you.
3. It's difficult to remember username and passwords of all neighbouring machines.
4. If you want to distribute jobs to n computers; you have to open n windows for remote login.
5. Gives you complete control of other machine.

6. It's very resource consuming. Opening n connections to remote computer will make your computer too slow.
7. More than two users can't remote login to same computer that means more than two users cannot give job to same computer even if it's resources are not utilised fully.
8. The VNC protocol[3] is pixel based. Although this leads to great flexibility (i.e.- any type of desktop can be displayed), it is often less efficient than solutions that have a better understanding of the underlying graphic layout like X11 or Windows Remote Desktop Protocol. Those protocols send graphic primitives or high level commands in a simpler form (e.g., "open window"), whereas RFB just sends the raw pixel data.

1.3 Problem definition and scope

In this project an application for distributing jobs to neighbouring machines is created. Today we are surrounded by lots of computing devices which remain idle most of the time. This project provides a mechanism by which a single machine can harness the computing power of other neighbouring machines.

This project assumes that neighbouring machines are connected with each other through LAN. So we can use POSIX API for socket programming to communicate over network. First we seek for neighbour machines that have CPU and Memory usage lower than defined by the owner of the machine. Then we create a list for all the neighbouring machines which are ready to provide their resources for our job. After that we send our instructions (along with data) which we want to be accomplished by neighbouring machine. The neighbouring machine upon completion returns back the result of our task.

First option for giving job to each neighbour machine is to go to each machine and login with username and password of that machine. It is very exhausting and tedious task to go to each machine, check its cpu or memory usage and then give it job. After giving regularly checking weather job submitted by us is complete or not is even more frustrating. When we have machines connected with each other why go to each one?

Jobs can be distributed using ssh or remote desktop login. It's good alternative to previous approach. Why we need another application? But in this case also we need username/password of other machine. Not all of our neighbours may be willing to provide us their username/passwords. In case of remote desktop login a lot of cpu and network resources are used up by the process itself, so we should not distribute cpu or networking intensive jobs using remote desktop login. But without login at neighbour machine how can we give our job to its shell. So we need a process that runs in background and provides us shell to do that. We run watchman & multicastwatchman process on neighbour machines which can answer our requests.

Scope:

This application distributes your job to shell on other machines that are idle. So without much effort you can harness their resources for your job and when job is complete; they give you result back. It can ease a lot of works for you. You can compile your program on another architecture and try running it on other machine. You can download any data on other machine and then collect it etc.

1.4 Formulation of present problem

Presently existing technology either don't provide GUI for example ssh or provide GUI which consume too much network, cpu and memory resources making system slow for example remote desktop login. We shall create a simple GUI on needy machine which will list all volunteers willing to help. We can easily send our job directory to volunteer. Volunteer after job completion will return the output directory back. Our application will be divided in three processes.

Distapp process will provide GUI for sending job to other machines. Needy can run it any time; navigate for volunteers by pressing navigate toolbar button. On pressing refresh button it will refresh the list of volunteers.

Watchman process will run in background and handles all the communication i.e. sending/receiving messages, files, sending directories etc.

Multicast Watchman process will also run in background and handle requests from needy about help. It will consult policy file and reply back to needy whether it is willing to help or not.

1.5 Organization of the thesis

The outline of the thesis is as follow –

Section 2 describes the hardware and software requirements for application.

- Section 2.1 deals with hardware required
- Section 2.2 deals with software support required.

Section 3 describes work done during analysis phase. This includes design of messages used for communication between needy and volunteer. Necessary functions required to create this application.

Section 4 describes how the software was developed. Necessary header files and work done by functions is described here.

Section 5 describes testing and analysis methods used in the work .

- Sub-section 5.1 deal with testing part
- Sub-section 5.2 deals with analysis part

Section 6 includes conclusion derived from the work.

Section 7 describes some recommendations and future work that can be done to extend this project's functionalities.

In appendix we have described the source code of the project.

At last references are listed.

2. DESCRIPTION OF HARDWARE AND SOFTWARE USED

2.1 Hardware used:

This application doesn't have any special hardware requirements. Any normal computer with Linux/GNU installed on it can run this application. Because it is network based application so more the bandwidth better will be communication between machines and data will be easily and fastly transferred. I have tested it on my laptop with

CPU: intel core 2 duo

RAM: 4GB

And on desktops with various permutation of following configurations in lab

CPU: intel i5, i3

RAM: 4GB, 2GB

Computers were connected using wired LAN with 100 MB/s bandwidth.

2.2 Software Requirements:

This application currently runs on Linux/GNU environment only. I have tried it on i386, i686 and x86_64 kernel architectures in Linux Ubuntu 11.10, Ubuntu 12.04, Ubuntu 12.10, Mint 13. Most of the information we needed about project was collected from the web. Also Linux man pages were quite helpful for clearing doubts. Some important details of software used for project development are described below.

There are many IDE (integrated development environments) available for creating project in Linux. I used *Anjuta*[4] for development purposes because it is free, open-source, customizable and easy to use. Few points about *Anjuta* are described as follow:

ANJUTA: is a versatile software development studio featuring a number of advanced programming facilities like:

- 1) Project management
- 2) Application wizard
- 3) Interactive debugger
- 4) Source editor
- 5) Version control
- 6) Plugin for GUI designer
- 7) Code completion and syntax highlighting
- 8) Profiler etc.

It focuses on providing simple and usable user interface, yet powerful for efficient development. *Anjuta* cannot create GUI required for our project but it can load xml file for GUI. So we will be using *GLADE*[5] interface designer for creating GUI for our project. Few points about *GLADE* are described as follow:

GLADE: Glade Interface Designer is a graphical user interface builder for GTK+, with additional options like:

- 1) Undo and redo support in all operations
- 2) Support for multiple open projects
- 3) Removal of code generation
- 4) Contextual help system with Devhelp
- 5) Programming language-independent, and does not produce code for events, but rather an XML file that is then used with an appropriate binding

So this GTKBuilder XML file created by *Glade* is used by *Anjuta* to create GUI for applications. *Glade* uses GTK+ for creating GUI. We will use GTK+ for creating GUI because:

GTK+ : GTK+[20] is written in C but has been designed from the ground up to support a wide range of languages, not only C/C++ but also languages such as Perl and Python and has additional options like:

- 1) GTK+ is cross-platform
- 2) GTK+ is open-source

Except all these tools we will be using auto tools for configuring, making and installing project. Before creating a project on Linux it's necessary to have good knowledge of GNU Build System (or Auto tools).

GNU Build System (or Autotools) :

Autotools[6][8] is a suite of programming tools designed to assist in making source-code packages portable to many Unix-like systems. It can be difficult to make a software program portable because:

- The C compiler may differ from system to system
- Certain library functions are missing on some systems
- Header files may have different names

One way to handle this is to write conditional code, with code blocks selected by means of pre-processor directives (`#ifdef`); but because of the wide variety of build environments this approach quickly becomes unmanageable.

Autotools is designed to address this problem more manageably. *Autotools* is part of the GNU toolchain and is widely used in many free-software and open-source packages. Its component tools are free-software-licensed under the GNU General Public License.

Autotools consists of the GNU utility programs:

- 1) Autoconf
- 2) Automake
- 3) Libtool.

Other related tools frequently used alongside it include:

- 4) GNU's make program
- 5) GNU gettext, pkg-config
- 6) GNU Compiler Collection (also called GCC)

We will discuss each of the components in brief.

GNU Autoconf:

Autoconf[6][8] is a tool for producing configure scripts for building, installing and packaging software on computer systems where a *Bourne shell* is available. A configure script configures a software package for installation on a particular target system. After running a series of tests on the target system the configure script generates header files and a *makefile* from templates, thus customizing the software package for the target system.

Autoconf generates a configure script based on the contents of a *configure.ac* file which characterizes a particular body of source code. The configure script, when run, scans the build environment and generates a subordinate *config.status* script which, in turn, converts other input files and most commonly *Makefile.in* into output files (*Makefile*) which are appropriate for that build environment. Finally the *make* program uses *Makefile* to generate executable programs from source code.

To process files, *autoconf* uses the GNU implementation of the m4 macro system.

Autoconf comes with several auxiliary programs such as *Autoheader*, which is used to help manage C header files; *Autoscan*, which can create an initial input file for *Autoconf*; and *ifnames*, which can list C pre-processor identifiers used in the program.

GNU Automake[6][8]:

GNU *Automake* is a programming tool that produces portable *makefiles* for use by the *make* program, used in compiling software. It is made by the Free Software Foundation as one of the GNU programs, and is part of the GNU build system. The *makefiles* produced follow the GNU Coding Standards.

It is written in the Perl programming language and must be used with GNU *autoconf*. *Automake* contains the following commands:

- I. aclocal
- II. automake

Like *Autoconf*, *Automake* used to be not entirely backwards compatible. For example, a project created with *automake* 1.4 will not necessarily work with *automake* 1.9. The situation however has improved noticeably with more recent releases.

GNU Libtool[6]:

Libtool helps to manage the creation of static and dynamic libraries on various Unix-like operating systems. *Libtool* accomplishes this by abstracting the library-creation process, hiding differences between various systems.

GNU Make[7]:

GNU *Make* is a utility that automatically builds executable programs and libraries from source code by reading files called *makefiles* which specify how to derive the target program. It also provides functionalities like pattern-matching in dependency graphs and build targets, functions which may be invoked allowing functionality like listing the files in the current directory etc.

GNU pkg-config[6]:

pkg-config is computer software that provides a unified interface for querying installed libraries for the purpose of compiling software from its source code. When a library is installed a *.pc* file should be included and placed into a directory with other *.pc* files. This *.pc* file contains a list of dependent libraries that programs using the package also need to compile. Entries also typically include the location of header files, version information and a description.

It is very helpful because if you are using a library, you don't need to remember on which other libraries this library is dependent or where the corresponding header files are.

GCC (GNU Compiler Collection):

The GNU Compiler Collection (GCC)[21] is a compiler system produced by the GNU Project supporting various programming languages. It is key component of GNU Build system. It is often chosen for developing software that is required to execute on a wide variety of hardware and/or operating systems.

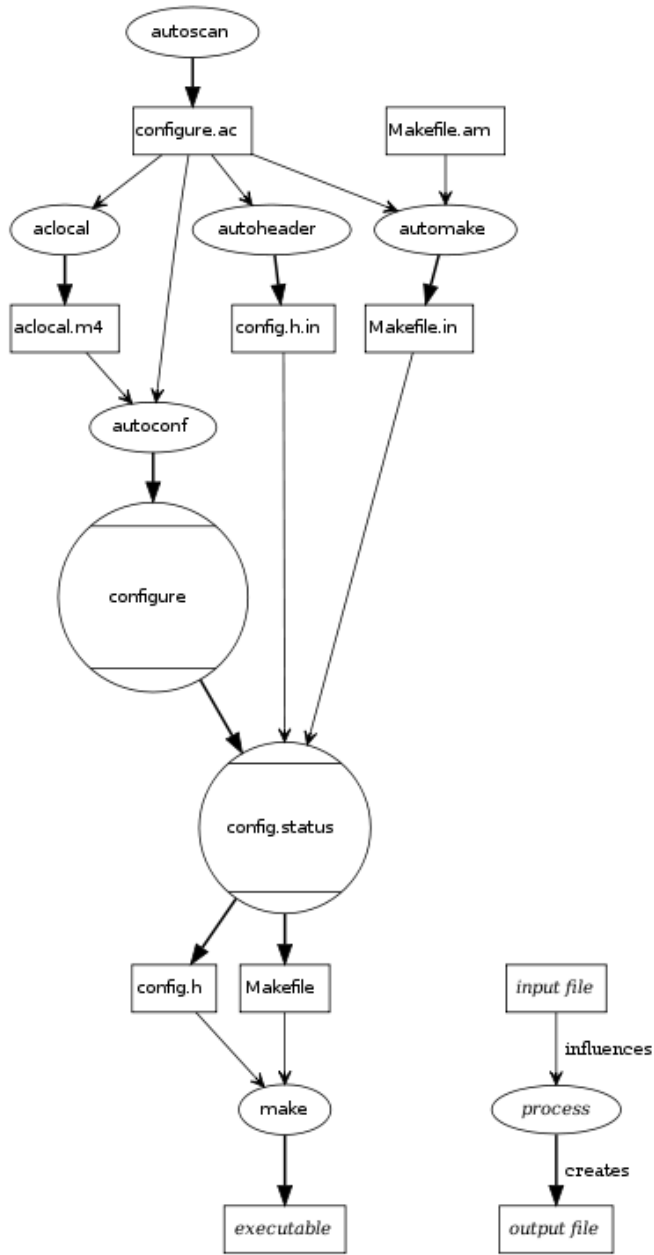


Figure 1: Flow Diagram of *Autotools* [22]

3. THEORETICAL TOOLS: ANALYSIS AND DEVELOPMENT

3.1 Types of communication available

Application running on needy machine wants to communicate with application running on volunteer machine and vice-versa. So we have to understand how this communication takes place in network. We have processes running on each end and process to process delivery needs two identifiers – IP address & port number at each end to make a connection. This combination of an IP address and port number is called a socket address. Client socket address defines the client process uniquely just as server socket address defines server process uniquely. If application layer process needs reliability we use a reliable transport layer protocol like TCP[10] and if process doesn't require reliability we use unreliable transport layer protocol like UDP[10]. TCP or transport layer protocol[10] is a process-to-process protocol. It is connection oriented i.e. it creates a virtual connection between two machines and provides flow and control at transport layer. There are some well-known applications that use TCP ports like FTP(20, 21), HTTP(80), DNS(53). We can't use these ports which are already used by other applications. We shall be using port 44445 for TCP based communication. TCP is stream oriented protocol i.e. it allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. Because the sending and the receiving processes may not write or read data at the same speed, TCP uses buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction. TCP reads bytes from these buffers and sends it to recipient but it keeps these bytes in the buffer until it receives an acknowledgment for sent bytes. If it doesn't receive acknowledgment it resends the byte after particular time period. Although TCP is stream oriented but IP sends data in packets. So, TCP groups a number of bytes together into a packet called a segment and send it to IP which takes care of rest.

UDP or User Datagram Protocol[10] is a connectionless, unreliable transport protocol. It helps in achieving process to process communication with minimum of overhead. There are some application using UDP ports like RPC(111), SNMP(161,162) etc. We shall be using port 44444 for UDP communication.

To achieve this communication we shall be using socket programming.

3.2 socket programming for achieving communication

Typically two processes communicate with each other on a single system through one of the following inter process communication techniques.

- a. Pipes
- b. Message queues
- c. Shared memory

But two processes on network interact with each other through sockets. A socket is a combination of IP address and port on one system. So, on each system a socket exists for a process interacting with the socket on other system over the network.

We shall be using socket programming[11] for interaction b/w our machine and neighbouring machines. Simple steps involved in programming sockets are:

- 1) Create the socket
- 2) Identify the socket
- 3) On the server, wait for an incoming connection
- 4) On the client, connect to the server's socket
- 5) Send and receive messages
- 6) Close the socket

3.3 Handling multiple connections using threads

It was required to handle multiple connections on both needy and volunteer side because one needy may send jobs to multiple volunteers; similarly one volunteer may send its services to other needy. So we must understand concept of multithreading and its usage. A thread of execution is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler. The implementation of threads and processes differs from one operating system to another. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources. Linux uses POSIX pthread API[12] for threads. A thread is sometimes referred to as a lightweight process. A thread will share all global variables and file descriptors of the parent process which allows the programmer to separate multiple tasks easily within a process. We could spawn a thread for each incoming connection request instead of forking a new process for each connection. This would make the network code inside the thread relatively simple. Using multiple threads[13] will also use fewer system resources compared to forking a child process to handle the connection request. Another advantage of using threads is that they will automatically take advantage of machines with multiple processors. It shares everything, except each thread will have its own program counter, stack and registers. Since each thread has its own stack, local variables will not be shared between threads.

Since all threads of a process share the same global variables, a problem arises with synchronization of access to global variables. For example, let's assume you have a global variable X and two threads A and B. Let's say threads A and B will merely increment the value of X. When thread A begins execution, it copies the value of X into the registers and increments it. Before it gets a chance to write the value back to memory, this thread is suspended. The next thread starts, reads the same value of X that the first thread read, increments it and writes it back to memory. Then, the first thread finishes execution and writes its value from the register back to memory. After these two threads finish, the value of X is incremented by 1 instead of 2 as you would expect. The workaround for this problem is to use a mutex (mutual exclusion) to make sure only one thread is accessing a particular section of your code. When one thread locks the mutex, it has exclusive access to that section of code until it unlocks the mutex. If a second thread tries to lock the mutex while another thread has it locked, the second thread will block until the mutex is unlocked and is once more available.

3.4 System calls and functions available from libraries

Along with standard C language we require system calls to perform actions like create a directory, delete a directory, change to directory, move along a directory, checking whether given path is file or directory etc. Computer softwares are developed to either automate some tasks or solve some problems. A problem can be broken into small problems each of which may require some services like computing the length of a string, opening a file etc. We call these functions as standard functions as any application can use them. These standard functions can be classified into two major categories:

- I. Library function calls
- II. System function calls

The functions which are a part of standard C library are known as Library functions. For example the standard string manipulation functions like `strcmp()`, `strlen()` etc. The functions which change the execution mode of the program from user mode to kernel mode are known as system calls. These calls are required in case some services are required by the program from kernel. For example, if we want to change the date and time of the system or if we want to create a network socket then these services can only be provided by kernel and hence these cases require system calls. System calls[14][15] acts as entry point to OS kernel. There are certain tasks that can only be done if a process is running in kernel mode. Examples of these tasks can be interacting with hardware etc. So if a process wants to do such kind of task then it would require itself to be running in kernel mode which is made possible by system calls. Thus a system call is how a program requests a service from an operating system's kernel. This may include hardware related services (e.g. accessing the hard disk), creating and executing new processes, and communicating with integral kernel services (like scheduling). Library functions may or may not call a system call. For example library function `strlen()` function used for calculating length of given string doesn't use any system call. But function like `fopen()` uses system call `open()` internally.

We required some data structure which could remain alive till our application executes. Best choice was to use Glib data structures. Glib[16] is a cross-platform software utility library that began as part of the GTK+ project. But after GTK+ version 2, GLib was released as a separate library so other developers, those who did not make use of the GUI-related portions of GTK+[20], could make use of the non-GUI portions of the library without the overhead of depending on the entire GUI library. GLib is a cross-platform library, applications using it to interface with the operating system are usually portable across different operating systems. GLib provides advanced data structures, such as memory chunks, doubly and singly linked lists, hash tables, dynamic strings and string utilities, such as a lexical scanner, string chunks (groups of strings), dynamic arrays, balanced binary trees, N-ary trees, quarks (a two-way association of a string and a unique integer identifier), keyed data lists, relations and tuples etc. But I used GHashTable and String utility functions of Glib[20] for storing <ipaddress, current working directory> entry and <ipaddress, open file descriptor> entry.

For creating GUI I have used GTK+ library. GTK+ is an object-oriented widget toolkit written in the C programming language; it uses the GLib object system for the object orientation. On the X11 display server, GTK+ uses Xlib to draw widgets. Using Xlib provides flexibility and allows GTK+ to be used on platforms where the X Window System is unavailable.

4. IMPLEMENTATION OF PROPOSED METHODOLOGY

Our application is group of three processes running on each machine whether it is needy or volunteer. First process is distapp which is GUI application for providing interface between user and application. Second is watchman process which listens at port 44445. This process runs in an infinite loop and waits for any incoming connection. Third is multicast_watchman process which also runs in an infinite loop and listens at port 44444. Needy and volunteer communicate with each other passing messages and data.

4.1 List of Important messages passed for communication:

1. __HELP__

This message is sent by needy machine when user clicks navigate toolbar button. This message is multicast to group HELP_GROUP (225.0.0.37) which is a broadcast address.

2. __READY_TO_HELP__

After getting __HELP__ message on multicast_watchman which listens on HELP_PORT(44444). Multicast_watchman calculates the average cpu usage for last 1, 5, 15 minutes and compares it with policy file readings. If it is able to help the needy it sends __READY_TO_HELP__ message back to needy.

3. __NOT_READY_TO_HELP__

If volunteer is not able to help needy it replies back with __NOT_READY_TO_HELP__ message.

4. __INIT_DELETE_DIR__

If needy decides to send job to volunteer; it sends this signal for initialisation to volunteer. Volunteer on receiving this signal initialises directory structure on machine. In /tmp directory if needy directory is not present; it creates needy directory there and chdir() to this directory. Then it deletes directory with name equal to needy's ip address (if that directory already exists). After that it creates fresh directory named as needy's ip address and chdir() to it.

So at volunteer side directory structure like /tmp/needy/needy_ip_addr exists after this step. Volunteer also inserts an entry in HashTable which stores <ip_addr, current_working_directory>.

5. __INIT_OVERWRITE_DIR__

It is similar to __INIT_DELETE_DIR__. But on receiving this message volunteer doesn't delete existing needy_ip_addr directory in /tmp/needy; but it is ready to overwrite the contents of /tmp/needy/needy_ip_addr.

6. __DIR__

On receiving this message volunteer creates a directory with name appended after this message and chdir() to this directory. It also puts/updates entry of <ip_addr, cwd> in hash table containing these entries.

7. __DIR_UP__

On receiving this message volunteer chdir() to one directory upper to its current working directory. It gets its current working directory from hash table which stores entries <ip_addr, current_working_directory>.

8. __FILE__

On receiving this message volunteer creates file with name appended to __FILE__ message. It then opens this file and stores <ip_addr, file_descriptor> in hash table which stores these entries. This file can be written in future.

9. __FILE_CLOSE__

On receiving this message volunteer closes the file descriptor present in hash table storing <ip_addr, file_descriptor> and removes entry from this hash table.

10. __DONE_SENDING_DATA__

This message is sent when needy has sent all required data for jobs to volunteer. Upon getting this message volunteer checks for presence of scripts directory in /tmp/needy/needy_ip_addr/start_dir. If scripts directory is not found it replies back with __NO_SCRIPTS_DIR_FOUND__, otherwise it feeds the scripts one by one to /bin/bash and executes them.

11. __INIT_DELETE_DIR_VOL__

This message is sent by volunteer to needy when it has completed the job given to it. Needy on receiving this message creates the /tmp/vol directory (if it doesn't already exist) and then chdir() to it. It then creates directory named vol_ip_addr and chdir() to it; puts entry <ip_addr, cwd> in corresponding hash table.

There are some special messages which are sent in case of error or some exception which are as follow:

12. `__NO_SCRIPTS_DIR_FOUND__`

This message is sent by volunteer when it doesn't find scripts directory asking the needy to abort the job.

13. `__ABORT__`

Volunteer upon receiving this message aborts the job running on behalf of needy, removes the directory structure in `/tmp/needy/needy_ip_addr`, closes file descriptor opened corresponding to needy in hash table & remove entry `<ip_addr, file_descriptor>` from hash table, remove `<ip_addr, cwd>` from hash table.

14. `__FINISH__`

This message is sent by volunteer to needy after receiving `__ABORT__` message. This signal cleans up entries `<ip_addr, cwd>` & `<ip_addr, fd>` in corresponding hash tables on needy side. So that needy can one again send job to volunteer.

15. `__PREV_ENTRY_FOUND_IPCWD__`

This message can be sent by any one (needy or volunteer) due to presence of `<ip_addr, cwd>`. It may occur if needy has sent another job to volunteer before volunteer could finish the earlier job sent by same needy.

16. `__PREV_ENTRY_FOUND_IPFD__`

This message can be sent by any one (needy or volunteer) due to presence of `<ip_addr, fd>`. It may occur if file we want to open is already opened on either side. It may occur if needy has sent another job to volunteer before volunteer could finish the earlier job sent by same needy.

Making software is a problem, given to us; we solve this problem by dividing it into small sub-problems and finding solutions for each sub-problem. At last we have solution for our large problem. Our application consists of various tasks and these tasks are accomplished by functions. We have written function for each small task. These functions when combined make our application software. According to C ideology functions doing similar task are grouped into header file. We shall discuss some of header files and what tasks the functions contained in them do.

3. Dialog.h

This is the header file corresponding to dialog.c. It contains functions for showing error to stderr.

- i. show_error_dialog() function is called to show error dialog like path entered is incorrect
- ii. show_info_dialog() showing info dialog like job you have sent is finished.

4. dir_op.h

This header file corresponds to dir_op.c. It contains functions like:

- i. init_path() which can be used to initialise a path. For example a call to initialise path /tmp/needy/ip_addr will create corresponding directory heirarchy and chdir() to it. If path already exists it will only chdir() to it.
- ii. is_dir_exist() to find out whether path given exists or not.
- iii. remove_dir() to remove directory recursively. If path is given; all directories below that tree are removed.
- iv. iterate_dir() to iterate the provided path. It is useful for sending files present below tree given by path.

5. dir_sender.h

This header file corresponds to dir_sender.c. It contains functions like:

- i. send_dir() to send whole directory tree below provided path. It is used by needy to send data and scripts for job to volunteer machine.
- ii. send_result() used by volunteer to send result back to needy after job completion.

6. file_sender.h

This header file corresponds to file_sender.c. It contains functions like send_file() used to send provided file to given ip. It is used by both needy and volunteer.

7. Sender.h

This header file corresponds to Sender.c. It contains functions like send_msg() used to send given message string to provided ip. It is used by both needy and volunteer.

8. Multicaster.h

This header file corresponds to multicaster.c. It contains function like:

- i. multicasts() it simply multicast __HELP__ message to HELP_GROUP on HELP_PORT.

9. get_pid.h

This header file corresponds to get_pid.c. It contains functions like

- i. get_pid() which find the process id if provided with name of process.

10. ip.h

This header file corresponds to ip.c. It contains function get_my_ip_addr() which return the ip address assigned to eth0 interface. It is useful for watchman process.

11. Scripts.h

This header file corresponds to scripts.c. It contains functions like:

- i. feed_scripts() which feeds script files present in provided directory one by one to run_script() function
- ii. run_script() it feeds script file to bash shell

12. calc_load.h

This header file corresponds to calc_load.c. It contains functions like

- i. get_number_cores() it calculates number of core present in CPU
- ii. get_avg_load() it calculates avg cpu load for last 1, 5, 15 minutes

13. hash_tab.h

This header file corresponds to hash_tab.c. We use two hash tables in our application; one to store <ip,cwd> and other to store <ip, fd>. It contains functions like:

- i. create_hash_table() to create hash table
- ii. does_ip_exist() is there any entry corresponding to ip provided
- iii. ipfd_insert_pair() insert an <ip, fd> entry in provided hash table
- iv. ipcwd_insert_pair() insert an <ip, cwd> entry in provided hash table
- v. ipfd_remove_pair() remove an <ip, fd> entry in provided hash table
- vi. ipcwd_remove_pair() remove an <ip, cwd> entry in provided hash table
- vii. find_fd() return the fd corresponding to ip in hash table
- viii. find_cwd() return the cwd corresponding to ip in hash table
- ix. ipfd_show_all_pair() printf all <ip,fd> pairs present in hash table
- x. ipcwd_show_all_pair() printf all <ip,cwd> pairs present in hash table

14. watchman.h

This header file corresponds to watchman.c. It contains functions like:

- i. est_con() to establish create a socket and listen on it for requests
- ii. receive_msg() to receive msg from client and do corresponding jobs

15. multicast_watchman.h

This header file corresponds to multicast_watchman.c. It contains function for sending back reply whether volunteer is ready to help or not.

16. gui.h

It contains necessary pointers for accessing GUI window in application.

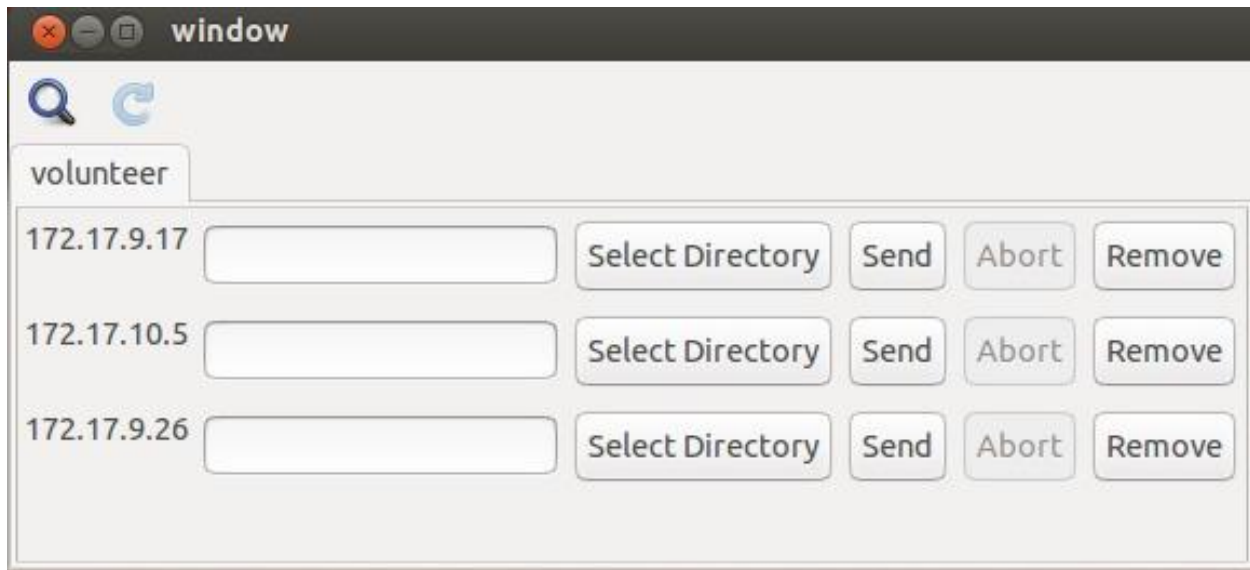
17. distapp.h

This header file corresponds to distapp.c. It contains pointers for different widgets in application and signal handler for various signals generated during user interaction with application GUI. It contains function like:

- i. make_hbox() to dynamically create a HBOX and add entries like label containing ip address of volunteer, an entry widget, buttons for selecting path, send, abort and remove the entry
- ii. add_to_iplist() to add an entry corresponding to volunteer

4.4 Creating GUI and handling user input:

For providing graphical interface to user we are using tabbed scrolled window as number of volunteers may grow large. We require some options to show to user so we shall create simple GUI like below using Glade UI Designer. It is very simple just drag and drop the widgets you want on working area and it generates corresponding distapp.ui file containing information in xml.



As we know GTK+ library is event driven system. All GUI applications are event driven. The application starts a main loop, which continuously checks for newly generated events. If there is no event, the application waits and does nothing. In GTK+ an event is a message from the X server. When the event reaches a widget, it may react to this event by emitting a signal. We have to connect a specific callback to this signal. This callback is just a handler function, which reacts to a signal.

We have added `on_navigate_clicked()` handler to navigate toolbar button. When user presses navigate toolbar button a `__HELP__` message is multicasted to `HELP_GROUP`.

`on_refresh_clicked()` handler is added to refresh button in toolbar. When this button is clicked application reads `/tmp/volunteer.txt` and reads the volunteer ip addresses from it. If volunteer is blacklisted in `vol.deny` file; it is not loaded in the list otherwise its ip is sent to `add_to_iplist()` function which dynamically creates horizontal box and inserts ip address of volunteer, buttons for other functionalities and show it to user.

User can select path to job directory by pressing button labeled "Select Directory". It gets the path for selected directory and puts into GTK entry widget. User can manually enter path or even drag and drop the job directory to GTK entry.

- a. `on_send_button_clicked()` handler is associated with send button. When user presses this button it checks weather path to job is correct or not. If path provided doesn't exit it show an error dialog informing user to enter correct path. If path is correct it sends the directory tree below path.
- b. `on_abort_button_clicked()` handler is associated with abort button. When abort button is clicked it sends `__ABORT__` message to volunteer.
- c. `on_rem_button_clicked()` is handler for remove button. When user presses this button entry corresponding to volunteer ip address is removed from list destroying the `GTK_HBOX`.

There is a process named watchman which doesn't interact with user in GUI. But it outputs what it is currently doing to stdout or stderr. User doesn't provides options to it. It just runs continuously and processes the incoming messages.

Another process named multicast watchman also doesn't interact with user. It replies back with message `__READY_TO_HELP__` or `__NOT_READY_TO_HELP__` according to cpu usages and policy file readings.

5. TESTING AND ANALYSIS

5.1 Testing

I have tested this project on various computers in lab. Some of them had Ubuntu 11.10 (i686), Ubuntu 11.10(x86_64), Ubuntu 12.04 (i686), Ubuntu 12.04(x86_64), Ubuntu 12.10 (i686), Ubuntu 12.10 (x86_64), Linux Mint 13 (x86_64), Linux Mint (i686) connected together through LAN and it works perfectly well with them.

5.2 Analysis

Watchman process crashed when I used to send more than 4 jobs to same volunteer machine because `MAX_ALLOWED_CLIENTS` in `defs.h` file was defined 7 in that case. I found that `listen(my_fd, MAX_ALLOWED_CLIENTS)` in `watchman.c` was able to keep only `MAX_ALLOWED_CLIENTS` in the queue only. You can increase its value; I increased its value to 30 and tested by sending 6 jobs simultaneously to same volunteer and it worked well. So we can set the value of `MAX_ALLOWED_CLIENTS` according to our needs.

Please don't send more than one requests from same needy to same volunteer simultaneously because at volunteer side hash table stores entry `<ip, cwd>` and `<ip, fd>` corresponding to previous request from needy and on getting second request it will reply back with `__PREV_ENTRY_FOUND_IPCWD__` or with `__PREV_ENTRY_FOUND_IPFD__`.

If we close application and rerun it; it's better to rerun watchman before distapp process. If we rerun distapp only, watchman would still be having previous value in hash table thus distapp GUI may not be able to show some entries in volunteer list thinking that it had previously shown these entries and no need to refresh them.

5.3 Results

Needy side:



Figure 3: needy GUI before navigate clicked

```

pc@pc-HCL-Desktop: ~/Documents/distapp_test_sV14/src
File Edit View Search Terminal Tabs Help
pc@pc-HCL-Desktop: ~/Docu...  pc@pc-HCL-Desktop: ~/Docu...  pc@pc-HCL-Desktop: ~/Docu...
pc@pc-HCL-Desktop:~/Documents/distapp_test_sV14/src$ ./watchman
Interface : <eth0>
Address : <172.17.9.17>
Waiting for sender!

```

Figure 4: needy watchman before navigate button clicked

```

pc@pc-HCL-Desktop: ~/Documents/distapp_test_sV14/src
File Edit View Search Terminal Tabs Help
pc@pc-HCL-Desktop: ~/Do...  pc@pc-HCL-Desktop: ~/Do...  pc@pc-HCL-Desktop: ~/Do...
pc@pc-HCL-Desktop:~/Documents/distapp_test_sV14/src$ ./multicast_watchman
multicast_watchman is running

```

Figure 5: needy multicast watchman before navigate button clicked

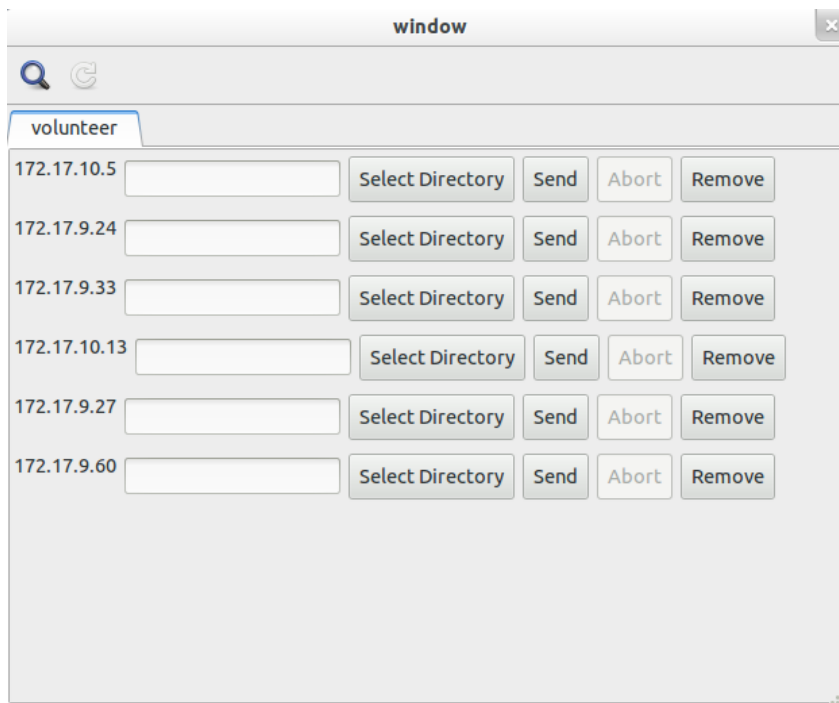


Figure 6: needy GUI after navigator and list refresher clicked

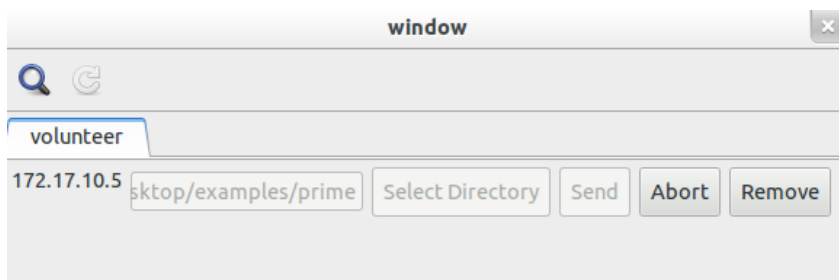


Figure 7: needy GUI after sending job directory to volunteer

```

pc@pc-HCL-Desktop: ~/Documents/distapp_test_sV14/src
File Edit View Search Terminal Tabs Help
pc@pc-HCL-Desktop: ~/Do...  pc@pc-HCL-Desktop: ~/Do...  pc@pc-HCL-Desktop: ~/Do...
pc@pc-HCL-Desktop:~/Documents/distapp_test_sV14/src$ ./watchman
  Interface : <eth0>
  Address : <172.17.9.17>
Waiting for sender!
received order:  __READY TO HELP__  from: 172.17.9.17
received order:  __READY TO HELP__  from: 172.17.10.5
received order:  __INIT DELETE DIR VOL__  from: 172.17.10.5
removed dir /tmp/vol/172.17.10.5/output
received order:  __DIR__  from: 172.17.10.5
order to create dir with path /tmp/vol/172.17.10.5/output
now cwd: /tmp/vol/172.17.10.5/output
received order:  __FILE__  from: 172.17.10.5
order to create file /tmp/vol/172.17.10.5/output/prime
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __FILE__  from: 172.17.10.5
order to create file /tmp/vol/172.17.10.5/output/primes.txt
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __DIR UP__  from: 172.17.10.5
received order:  __DIR UP__  from: 172.17.10.5
received order:  __FINISH__  from: 172.17.10.5
Ending communication with volunteer
printing <ip,fd> entries
printing <ip,cwd> entries
key: 172.17.9.17 value: GARBAGE

```

Figure 8: needy watchman after send button clicked

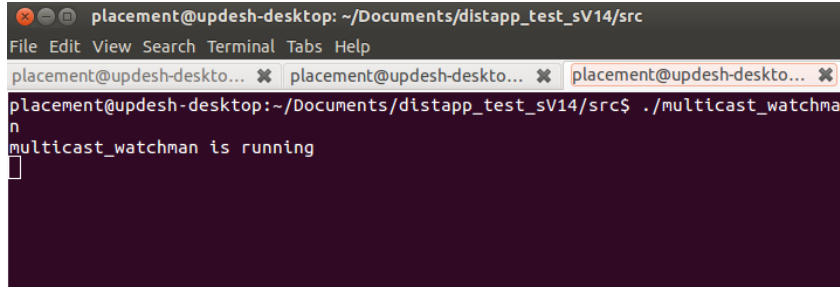
```

pc@pc-HCL-Desktop: ~/Documents/distapp_test_sV14/src
File Edit View Search Terminal Tabs Help
pc@pc-HCL-Desktop: ~/Do...  pc@pc-HCL-Desktop: ~/Do...  pc@pc-HCL-Desktop: ~/Do...
pc@pc-HCL-Desktop:~/Documents/distapp_test_sV14/src$ ./watchman
  Interface : <eth0>
  Address : <172.17.9.17>
Waiting for sender!
received order:  __READY TO HELP__  from: 172.17.9.17
received order:  __READY TO HELP__  from: 172.17.10.5
received order:  __INIT DELETE DIR VOL__  from: 172.17.10.5
removed dir /tmp/vol/172.17.10.5/output
received order:  __DIR__  from: 172.17.10.5
order to create dir with path /tmp/vol/172.17.10.5/output
now cwd: /tmp/vol/172.17.10.5/output
received order:  __FILE__  from: 172.17.10.5
order to create file /tmp/vol/172.17.10.5/output/prime
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __FILE__  from: 172.17.10.5
order to create file /tmp/vol/172.17.10.5/output/primes.txt
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __FILE CLOSE__  from: 172.17.10.5
received order:  __DIR UP__  from: 172.17.10.5
received order:  __DIR UP__  from: 172.17.10.5
received order:  __FINISH__  from: 172.17.10.5
Ending communication with volunteer
printing <ip,fd> entries
printing <ip,cwd> entries
key: 172.17.9.17 value: GARBAGE
received order:  __FINISH__  from: 172.17.10.5
Ending communication with volunteer
printing <ip,fd> entries
printing <ip,cwd> entries
key: 172.17.9.17 value: GARBAGE

```

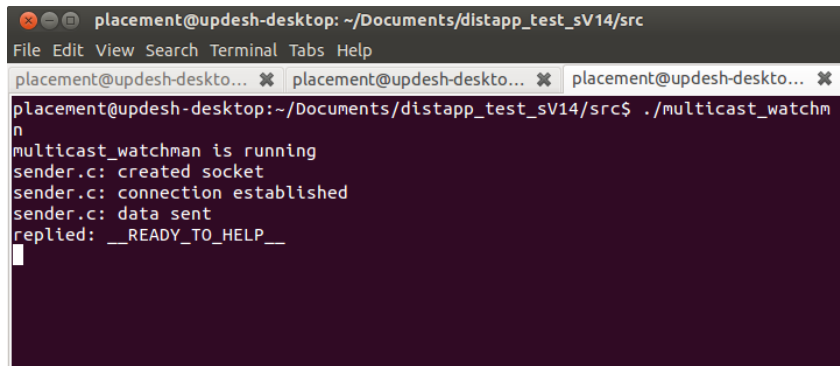
Figure 9: needy watchman after abort button click

Volunteer side:



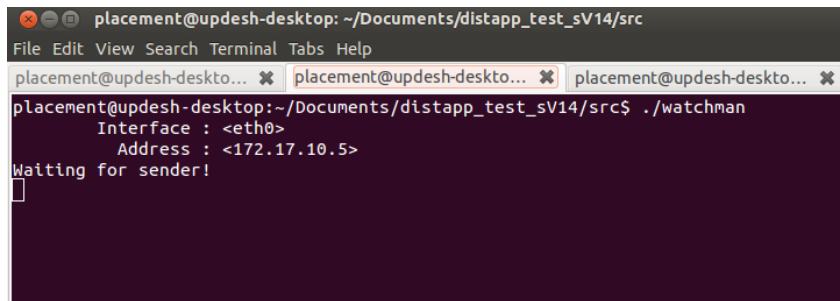
```
placement@updeshtop: ~/Documents/distapp_test_sv14/src
File Edit View Search Terminal Tabs Help
placement@updeshtop: ~/Documents/distapp_test_sv14/src$ ./multicast_watchman
multicast_watchman is running
```

Figure 10: volunteer multicast watchman before navigator clicked by needy



```
placement@updeshtop: ~/Documents/distapp_test_sv14/src
File Edit View Search Terminal Tabs Help
placement@updeshtop: ~/Documents/distapp_test_sv14/src$ ./multicast_watchman
multicast_watchman is running
sender.c: created socket
sender.c: connection established
sender.c: data sent
replied: __READY_TO_HELP__
```

Figure 11: volunteer multicast watchman after navigator clicked by needy



```
placement@updeshtop: ~/Documents/distapp_test_sv14/src
File Edit View Search Terminal Tabs Help
placement@updeshtop: ~/Documents/distapp_test_sv14/src$ ./watchman
Interface : <eth0>
Address : <172.17.10.5>
Waiting for sender!
```

Figure 12: Volunteer watchman before any job sent by needy

```

placement@updesht-desktop: ~/Documents/distapp_test_sv14/src
File Edit View Search Terminal Tabs Help
placement@updesht-deskto... placement@updesht-deskto... placement@updesht-deskto...
now cwd: /tmp/needy/172.17.9.17/prime/input
received order: __FILE__ from: 172.17.9.17
order to create file /tmp/needy/172.17.9.17/prime/input/prime.c
received order: __FILE_CLOSE__ from: 172.17.9.17
received order: __FILE_CLOSE__ from: 172.17.9.17
received order: __DIR_UP__ from: 172.17.9.17
received order: __DIR_UP__ from: 172.17.9.17
received order: __DIR_UP__ from: 172.17.9.17
received order: __DONE_SENDING_DATA__ from: 172.17.9.17
start dir: prime
temp_arr2 /tmp/needy/172.17.9.17/prime/scripts
trying to feed script /tmp/needy/172.17.9.17/prime/scripts
path: /tmp/needy/172.17.9.17/prime/scripts
running script file /tmp/needy/172.17.9.17/prime/scripts/prime.sh
sender.c: created socket
sender.c: connection established
sender.c: data sent
sender.c: created socket
sender.c: connection established
sender.c: data sent
base_dir: __DIR__ output
FILE FOUND: prime
sender.c: created socket
sender.c: connection established
sender.c: data sent
file_sender.c: created socket
file_sender.c: connection established
file_sender.c: file sent
sender.c: created socket
sender.c: connection established
sender.c: data sent
FILE FOUND: primes.txt
sender.c: created socket
sender.c: connection established
sender.c: data sent
file_sender.c: created socket
file_sender.c: connection established
file_sender.c: file sent
sender.c: created socket
sender.c: connection established
sender.c: data sent
sender.c: created socket
sender.c: connection established

```

Figure 13: Volunteer after job sent by needy

```

received order: __ABORT__ from: 172.17.9.17
got fd: -1removing dir /tmp/needy/172.17.9.17
removed file /tmp/needy/172.17.9.17/prime/output/prime
removed file /tmp/needy/172.17.9.17/prime/output/primes.txt
removed dir /tmp/needy/172.17.9.17/prime/output
removed file /tmp/needy/172.17.9.17/prime/input/prime.c
removed dir /tmp/needy/172.17.9.17/prime/input
removed file /tmp/needy/172.17.9.17/prime/scripts/prime.sh
removed file /tmp/needy/172.17.9.17/prime/scripts/prime.sh
removed dir /tmp/needy/172.17.9.17/prime/scripts
removed dir /tmp/needy/172.17.9.17/prime
sender.c: created socket
sender.c: connection established
sender.c: data sent
printing <ip,fd> entries
printing <ip,cwd> entries

```

Figure 14: Volunteer after abort by needy

6. CONCLUSIONS

Thus application helps user to distribute its jobs in the form of shell script to neighbouring machines which are not very busy in an easy and interactive way with GUI. Neighbouring machines on completion return back the result to Needy machine. Thus it helps in harnessing the computational power of idle neighbouring machines distributing the jobs among them. User can manually give the job to neighbour he wants. For example if we know that volunteer A has good internet connection we can send site grabbing job to it while volunteer B having high RAM & cpu power can be given job for searching and indexing pages grabbed by volunteer A. volunteer C can be given a C program to compile and run it there and send results back to us. All these jobs can be managed from your own computer. If we don't want to help some neighbour we can put his IP address in needy.deny file while if we don't want to get help from some neighbour we can put his IP in volunteer.deny file. Thus we have more control on to whom we can send job and from whom we can get help.

7. RECOMMENDATION AND FUTURE WORK

Although we can send our jobs to neighbour machines and get results back easily with this application. But this application doesn't provide security features. For example, on volunteer machine needy can move out of BASE_NEEEDY directory which should not be allowed. Needy now has permissions and access rights of the volunteer. If root user has installed this application root user's permission and access rights will be given to process sent by needy. So this watchman process should not be run as root user of volunteer.

We can remove this scenario altogether if we provide needy with less powerful shell than bash. Means we have to jail root the needy jobs. By doing this needy process will not have access outside BASE_NEEEDY directory. Thus it will be jailed and can only access binaries allowed by volunteer. We have to write less powerful shell like bash configurable by user at volunteer side. Volunteer can add binaries like ls, chdir, mkdir, touch etc. but deny accessing binaries like passwd, sudo, adduser and other administrative binaries to needy; thus increasing security.

Second thing is that over an insecure network messages and data communication by our application can be tampered by adversary as it is not encrypted. We can exchange keys and send encrypted data over network.

APPENDIX

A.1 EXPLANATION OF SOURCE CODE

We shall discuss important three processes in our source code. Explanation of code is within the comments in source code. Note that file here may not be complete source file some important functions are listed instead. Sometimes only instructions are there to state what is happening here instead of code.

Filename : defs.h

```
#ifndef DEFS_H
#define DEFS_H

/*although watchman.c gets its host computer's ip address by running
 * get_my_ip_addr() in ip.c but is safer if you put your ip address here in
 * MY_ADDR
 */
#define MY_ADDR "172.0.0.1"

// port where watchman process listens
#define MY_PORT 44445

/*
 * set MAX_ALLOWED_CLIENTS to value so that it can handle as defined number of
 * requests in queue at a time
 */
#define MAX_ALLOWED_CLIENTS 127

/* defines the size of array used to store sender's message*/
#define MAX_SENDER_MSG_SIZE 1024

/* defines the size of array used to store name of path */
#define MAX_DIR_SIZE 256

/* defines the size of array used to store name of directory */
#define MAX_DIR_NAME_SIZE 64

//set directory permissions which is created when __DIR__ message is got
#define DIR_PERM 0777

//set file permissions which is created when __FILE__ message is got
#define FILE_PERM 0777

/*
 * define the name of file which is read when user clicks on
 * reload toolbar button, mode used when writing this file and
 * location of this file
 */
#define VOLUNTEER_TXT "volunteer.txt"
#define VOLUNTEER_TXT_MODE "a"
#define VOLUNTEER_TXT_PATH "/tmp/volunteer.txt"
```

```

/*
 * this file is of no particular use define its name and path
 */
#define NEEDY_TXT "needy.txt"
#define NEEDY_TXT_PATH "/tmp/needy.txt"

/*
 * on volunteer side
 * define the path where all needy script, input, output directories
 * will be kept
 */
#define BASE_NEEDY "/tmp/needy"

/*
 * on needy side
 * define the path where all volunteer output directory will be kept
 */
#define BASE_VOL "/tmp/vol"

//shells used for executing shell scripts on volunteer side
#define BASH_SHELL "/bin/bash"
#define SH_SHELL "/bin/sh"

//for multicast_watchman
/*
 * set the port on which multicast_watchman will listen
 */
#define HELP_PORT 44444
#define HELP_GROUP "225.0.0.37"

//arr size for receiving msg which in this case is strlen(__HELP__) + 1
#define MSGBUFSIZE 9
#define NEEDY_TXT_MODE "a"

/*
 * file containing threshold value of cpu usage for 1, 5, 15 minutes
 * you can set the threshold values yourself default it is 50, 50, 50
 */
#define POLICY_TXT "policy.txt"
#define POLICY_TXT_MODE "r"

/*
 * name of file containing ip addresses of needy which are denied for help by
 * replying back __NOT_READY_TO_HELP__
 */
#define NEEDY_DENY_LIST "needy.deny"

/*
 * max number of needy denied for help
 * it is size of array if value is more you can increase it
 */
#define MAX_DENY_LIST_SIZE 30

//for distapp.c
// ip addresses of volunteers whom we don't want to send our job

```

```
#define VOL_DENY_LIST "vol.deny"
```

```
//names are misleading they just define the size of ip-addr xxx.xxx.xxx.xxx
```

```
#define SHMSIZE 16
```

```
#define SHMKEY 1234
```

```
#endif
```

Filename : distapp.c

```
/*#include necessary files
```

```
/* For testing propose use the local (not installed) ui file */
```

```
/* #define UI_FILE PACKAGE_DATA_DIR"/ui/distapp.ui" */
```

```
#define UI_FILE TERMINAL_SRC"distapp.ui"
```

```
#define TOP_WINDOW "window"
```

```
// initialise the deny list
```

```
char *deny_list[MAX_DENY_LIST_SIZE];
```

```
int dl_size = 0;
```

```
// function below returns 1 if ip passed as parameter is denied for help
```

```
int is_denied( char *ip ){
```

```
int i = 0, ret = 0;
```

```
for( i = 0; i < dl_size; i++ ){
```

```
if( strcmp( deny_list[i], ip ) == 0 ){
```

```
ret = 1;
```

```
break;
```

```
}
```

```
}
```

```
return ret;
```

```
}
```

```
//
```

```
/* Create a new window loading a file */
```

```
static void
```

```
distapp_new_window (GApplication *app,
```

```
GFile *file)
```

```
{
```

```
GtkWidget *window;
```

```
GtkBuilder *builder;
```

```
GError* error = NULL;
```

```
DistappPrivate *priv = DISTAPP_GET_PRIVATE(app);
```

```
/* Load UI from file */
```

```
builder = gtk_builder_new ();
```

```
//load the interface distapp.ui created after drag and drop process
```

```
if (!gtk_builder_add_from_file (builder, UI_FILE, &error))
```

```
{
```

```
g_critical ("Couldn't load builder file: %s", error->message);
```

```
g_error_free (error);
```

```

    }

/* Auto-connect signal handlers */
gtk_builder_connect_signals (builder, app);

/* Get the window object from the ui file */
window = GTK_WIDGET (gtk_builder_get_object (builder, TOP_WINDOW));
if (!window)
{
    g_critical ("Widget \"%s\" is missing in file %s.",
                TOP_WINDOW,
                UI_FILE);
}

mydata.vol_viewport = GTK_VIEWPORT( gtk_builder_get_object (builder, VOL_VIEWPORT) );
mydata.main_window = window;

mydata.vbox = gtk_vbox_new ( FALSE, 0 );
gtk_container_add ( GTK_CONTAINER( mydata.vol_viewport ), mydata.vbox );

/* ANJUTA: Widgets initialization for distapp.ui - DO NOT REMOVE */

g_object_unref (builder);

gtk_window_set_application (GTK_WINDOW (window), GTK_APPLICATION (app));
if (file != NULL)
{
    /* TODO: Add code here to open the file in the new window */
}

gtk_widget_show_all (GTK_WIDGET (window));

//load the deny list
char temp[SHMSIZE];
FILE *ifp = fopen(VOL_DENY_LIST,"r");
if( ifp == NULL ){
//function below shows error on stdout
    show_err( "ifp is NULL\n");
}
else{
//put the ip address into VOL_DENY_LIST file
    while( fgets( temp, SHMSIZE, ifp ) != NULL ){
        char *ptr = (char *)malloc( strlen( temp ) );
        strcpy( ptr, temp );
        deny_list[dl_size] = ptr;
        dl_size++;
    }
}
//
}

void on_send_button_clicked( GtkWidget *widget, gpointer data ){

    Info *info = (Info *)data;
    //g_print("send button clicked for %s\n", info -> ip );

```

```

        //function below enables or disables a widget if send TRUE or FALSE value //disable send button
        and path entry and enable abort button
        gtk_widget_set_sensitive ( info -> send_button, FALSE );

```

```

        .
        .
        .
//function below gets string from gtk entry box
        char *path = gtk_entry_get_text ( info -> path_entry );
        char *ip = info -> ip;

        //if path exists
        if( is_dir_exist ( path ) > 0 ){
            //send the dir
            send_dir( path, ip );
        }
        //otherwise path given is NULL
        //show message to give a valid path
        else{
            show_error_dialog ("Please enter a valid path" );
//unable or disable widgets as done previously like
            gtk_widget_set_sensitive ( info -> send_button, TRUE );
            .
            .
            .
        }
}

```

```

void on_abort_button_clicked( GtkWidget *widget, gpointer data ){
    Info *info = (Info *)data;
//send abort signal to volunteer

        if( send_msg ( info -> ip, "__ABORT__" ) == 0 ){
            //disable abort button and enable remove button
        }
}

```

```

void on_rem_button_clicked( GtkWidget *widget, gpointer data ){
//just remove the hbox containg necessary widgets corresponding to ip address
        gtk_widget_destroy ( info -> hbox );
}

```

```

void on_path_button_clicked( GtkWidget *widget, gpointer data ){
    Info *info = (Info *)data;
// create a dialog using function below
        dialog = gtk_dialog_new( ... );
//get path to folder which is selected by user

        if (gtk_dialog_run (GTK_DIALOG (dialog)) == GTK_RESPONSE_ACCEPT){

            gchar *path;
            path = gtk_file_chooser_get_uri (GTK_FILE_CHOOSER (dialog));
            gtk_entry_set_text ( info -> path_entry, path + 7 );
            g_free ( path );

```

```

    }
    gtk_widget_destroy (dialog);
}

GtkWidget* make_hbox( gchar *ip ){

    GtkWidget *box = NULL;
    // create pointer to label; similarly create pointer to all buttons to be fit in hbox
    GtkWidget *ip_label;

    // create a new hbox
    box = gtk_hbox_new ( FALSE, 0 );
    ip_label = gtk_label_new ( g_strdup( ip ) );
    // pack the ip label in box
    gtk_box_pack_start ( box , ip_label, FALSE, FALSE, 3 );
    // show ip label in GUI
    gtk_widget_show ( ip_label );
    // create a separator
    vsep = gtk_vseparator_new ();
    // similarly pack separator and show it in GUI
    // similarly add all necessary widgets to GUI
    // create a new Info structure and get pointer to it
    Info *info = (Info *)malloc( sizeof(Info) );
    // add ip label to this structure using pointer
    info -> ip_label = ip_label;
    // similarly add all necessary fields in this Info structure
    // connect the callbacks to each widget as follow
    // here on clicking abort button on_abort_button_clicked callback has been assigned
    g_signal_connect( abort_button, "clicked", G_CALLBACK(on_abort_button_clicked),
(gpointer)info );
    .
    .
    .
    info -> hbox = box;
//return the box
    return box;

}

void add_to_iplist ( gchar *ip ){

    GtkSeparator *hsep;
    if( mydata.vbox == NULL )
        mydata.vbox = gtk_vbox_new ( FALSE, 0 );

    GtkWidget *hbox = make_hbox( ip );

    // pack hbox obtained in vbox
    // show hbox and original vbox
}

/* handler for event when navigate button in toolbar is clicked */
void
on_navigate_clicked (GtkToolButton *toolbutton, gpointer user_data)
{

```

```

        g_print("on navigate clicked\n");
// using multicaster multicast
// truncate VOLUNTEER_TXT_PATH file
        if( truncate(VOLUNTEER_TXT_PATH, 0 ) == -1 ){
// if error occurred in truncating file show error
        }
// now multicast
        multicast();
        gtk_widget_set_sensitive ( (GtkToolButton *)user_data, TRUE );
    }

/* handler for event when refresh button in toolbar is clicked */
void
on_refresh_clicked (GtkToolButton *toolbutton, gpointer user_data)
{

    gtk_widget_set_sensitive ( toolbutton, FALSE );
    char ip[SHMSIZE];
    FILE *ifp = fopen( VOLUNTEER_TXT_PATH, "r" );
    if( ifp == NULL ){
// show error that an error occurred in opening file VOLUNTEER_TXT_PATH
    }else{
        while( fgets( ip, SHMSIZE, ifp ) != NULL ){
// if volunteer is not denied we can add him in iplist
            if( is_denied( ip ) == 0 )
                add_to_iplist( ip );
// otherwise ip is not allowed to act as volunteer
        }
    }
}

```

Filename : multicast_watchman.c

```

// include necessary files
//check weather given ip is denied services from our computer
//if it is denied services it will be present in deny_list array

int is_denied( char *ip ){
    int i = 0, ret = 0;
    for( i = 0; i < dl_size; i++ ){
        if( strcmp( deny_list[i], ip ) == 0 ){
            ret = 1;
            break;
        }
    }
    return ret;
}

//thread to reply back to needy
void *cal_reply( void *multicaster_ip ){

    char *sender_ip = (char *)multicaster_ip;

```



```

// get the number of cores present in our cpu
int cores = get_number_cores ( );
/*
allocate memory and initialise pointers for storing minute, five_minute, fifteen_minute cpu usages like
below
*/

float *minute = (float *)malloc( sizeof(float) );
//be careful average load can be greater than 100% so truncate it

if( get_avg_load ( minute, five_minute, fifteen_minute) ){
// calculate minute, five_minute, fifteen_minute avg as below
minute_avg = (*minute)*100.0/cores;
// open file to apply policy
FILE *ifp = fopen(POLICY_TXT,POLICY_TXT_MODE);
if( ifp == NULL ){
// show error that policy file could not be opened and we shall use default policy
}else{
// read policy for minute , five_minute, fifteen_minute cpu threshold values
fscanf(ifp, "%f %f %f", &policy_minute_avg, &policy_five_minute_avg,
&policy_fifteen_minute_avg );
fclose (ifp);
}
//instead make sender a separate process or thread
char reply[30];
if( minute_avg < policy_minute_avg && five_minute_avg < policy_five_minute_avg &&
fifteen_minute_avg < policy_fifteen_minute_avg ){
//if all conditions in policy file are met reply yes for help
strcpy( reply, "__READY_TO_HELP__");
}else{
strcpy( reply,"__NOT_READY_TO_HELP__");
}
//send back the reply
if( send_msg ( sender_ip, reply ) == 0 )
printf("replied: %s\n", reply );

}else{
/*
tell user that some error occurred in calculating five , five_minute, fifteen_minute average and exit
*/
}
}
int main( )
{
//load the deny list
char temp[SHMSIZE];
FILE *ifp = fopen(NEEEDY_DENY_LIST,"r");
if( ifp == NULL ){
// error in opening needy.deny file
}else{
//load deny list before doing anything
}

printf("multicast_watchman is running\n");
struct sockaddr_in addr, sender_addr;
int fd, nbytes,addrlen;

```

```

struct ip_mreq mreq;
char msgbuf[MSGBUFSIZE];

u_int yes=1;      /** MODIFICATION TO ORIGINAL */

/* create what looks like an ordinary UDP socket */
if ((fd=socket(AF_INET,SOCK_DGRAM,0)) < 0) {
    perror("multicast_watchman.c: Could not create socket\n");
    exit(1);
}
// reuse the socket
if( setsockopt( fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes) ) < 0 ){
// show error that socket could not be reused
}
// set fields of addr
memset( &addr, 0, sizeof(addr) );
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(HELP_PORT);

// bind the socket address to file descriptor which can be read or written to in future
if( bind( fd, (struct sockaddr *)&addr, sizeof(addr) ) < 0 ){
// show error that binding was not done and exit
}
// set options for adding to multicast address
mreq.imr_multiaddr.s_addr = inet_addr( HELP_GROUP );
mreq.imr_interface.s_addr = htonl(INADDR_ANY);
if( setsockopt( fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq)) < 0 ){
// error occurred in setting socket options exit
}

//run in an infinite loop and wait to any request from needy
while( 1 ){

    addrlen = sizeof( sender_addr );
// receive bytes from user
    if( (nbytes = recvfrom( fd, msgbuf, MSGBUFSIZE, 0, (struct sockaddr*)&sender_addr,
(socklen_t*)&addrlen ) ) < 0 ){
        perror("Could not recvfrom multicaster\n");
        exit(1);
    }
    if( strcmp(msgbuf,"__HELP__") == 0 ){

//get sender ip
        char *sender_ip = malloc( strlen(inet_ntoa( sender_addr.sin_addr )) + 1);
        strcpy( sender_ip, inet_ntoa( sender_addr.sin_addr ) );

//create a new thread and send multicaster ip to it

        if( is_denied( sender_ip ) == 0 ){
// if needy is not denied for help
            pthread_t thr;
// create a new thread
            if( pthread_create( &thr, NULL, cal_reply, (void *)sender_ip ) < 0 ){
// show error stating failure of thread creation
            }

```

```

// wait for thread to join
pthread_join( thr, NULL );
    }else{
//else sender_ip is denied for help and reply back __NOT_READY_TO_HELP__
    send_msg( sender_ip, "__NOT_READY_TO_HELP__" );
    }

    }else{
// we got message that was not expected
    }
    msgbuf[0] = '\0';
    }
    return 0;
}

```

Filename : watchman.c

```

// include necessary header files
// be careful with global defined variables as anyone can change them
// initialise hash table
GHashTable *ipfd = NULL, *ipcwd = NULL;

// thread handler for populating volunteer list
void *populate_volunteer_list( void *ip_addr ){

    char *ip = ( char * )ip_addr;

    FILE *ofp = fopen(VOLUNTEER_TXT_PATH, VOLUNTEER_TXT_MODE);
    if( ofp != NULL ){
// write to volunteer file
        fprintf( ofp,"%s\n", ip);
        fclose( ofp );
    }

    return ;
}

int receive_msg( int my_fd ){

// initialise necessary data structures like process_mask, sender_fd, cwd[], temp_arr[]
    struct sockaddr_in sender_addr;
    int size_sockaddr_in = sizeof(struct sockaddr_in);

    //accept incoming connection
    sender_fd = accept( my_fd, (struct sockaddr*)&sender_addr, &size_sockaddr_in );

    if( sender_fd < 0 ){
// show error that connection could not be accepted
    }
//get the ip of msg sender
    char *vol_ip = malloc( strlen( inet_ntoa(sender_addr.sin_addr) ) + 1 );
    strcpy( vol_ip, inet_ntoa(sender_addr.sin_addr) );

// now read from sender_fd
while( ( read_size = recv( sender_fd, sender_msg, MAX_SENDER_MSG_SIZE, 0 ) ) > 0 ){
    sender_msg[read_size] = '\0';
}

```

```

/*
check if any file is already opened corresponding to given ip_addr if it's open write to it
*/

        if( does_ip_exist(ipfd,vol_ip) == TRUE )
            fd = find_fd( ipfd, vol_ip );

//if message is to close that file then close it
        if( strcmp(sender_msg,"__FILE_CLOSE__") == 0 ){

//===== remove entry from ipfd =====
            if( fd > 0 ){
                close( fd );
                ipfd_rem_pair( ipfd, vol_ip );
            }
            else if( fd > 0 ){
//otherwise write to that file
                if( write( fd, sender_msg, read_size ) != read_size ){
// show error in writing to file descriptor fd
                }
            }

        }

        if( read_size == -1 ){
// show error that error in reading from fd
        }else{
            //used by sender side
            if( strcmp(sender_msg,"__READY_TO_HELP__") == 0 ){
                printf("received order: __READY_TO_HELP__ from: %s\n", vol_ip);
                //create a thread that will put ip in volunteer file
                //WRITE TO FILE
                if( does_ip_exist(ipcwd, vol_ip) == FALSE ){
                    ipcwd_insert_pair( ipcwd, vol_ip, "GARBAGE");
                    pthread_t writer;

                    if( pthread_create( &writer, NULL, populate_volunteer_list, (void*)vol_ip )
< 0 ){
// error in creating thread show this error
                    }
                    pthread_join( writer, NULL );
                }else{
//volunteer is already being helped by me
//so we will not put him in volunteer list
                }

            }
            else if( strcmp( sender_msg, "__PREV_ENTRY_FOUND_IPCWD__" ) == 0 ){
//show message "Either you are already helping / being helped by same machine\n"
            }
            else if( strcmp( sender_msg, "__PREV_ENTRY_FOUND_IPFD__" ) == 0 ){
// show message "Either you are already helping / being helped by same machine\n"
// stop sending further messages
            }
        }
    }
}

```

```

        else if( strcmp( sender_msg, "__NO_SCRIPTS_DIR_FOUND__" ) == 0 ){
// show message "No scripts dir found in task\n"
// stop sending further messages
        }
        //used by receiver side
        //initialise the dir creation /tmp/needy/ip_of_needy
        else if( strcmp( sender_msg, "__INIT_OVERWRITE_DIR_NEEDY__" ) == 0 ){

            if( does_ip_exist( ipcwd, vol_ip ) == TRUE ){
                if( strcmp( find_cwd(ipcwd, vol_ip), "GARBAGE" ) != 0 )
                    send_msg( vol_ip, "__PREV_ENTRY_FOUND_IPCWD__");

            }

            sprintf( temp_arr, BASE_NEEDY"%s", vol_ip );
            if( is_dir_exist( temp_arr ) == 1 ){
                if( chdir( temp_arr ) != 0 ){
// show error in changing directory
                }
            }else if( is_dir_exist( temp_arr ) == 0 ){
//if dir doesn't exist
//path is not initialised

                if( init_path( vol_ip, BASE_NEEDY ) < 0 )
                    return -1; //else path is initialised

                if( chdir( temp_arr ) != 0 ){
// show error in changing directory
                }
            }

/*
check using cwd and create own string by guess or append and compare and then show message
*/
//===== get cwd and put in ipcwd =====
            if( getcwd(cwd, sizeof(cwd)) == NULL ){
// show error in getting cwd
            }
// insert <ip,cwd> in ipcwd hash table
            ipcwd_insert_pair( ipcwd, vol_ip, cwd );
        }
//delete the previous dir and start fresh
        else if( strcmp( sender_msg, "__INIT_DELETE_DIR_NEEDY__" ) == 0 ){
            if( does_ip_exist( ipcwd, vol_ip ) == TRUE ){
                if( strcmp( find_cwd(ipcwd, vol_ip), "GARBAGE" ) != 0 )
// send message that previous entry found in hash table
                    send_msg( vol_ip, "__PREV_ENTRY_FOUND_IPCWD__");

            }

            sprintf( temp_arr, BASE_NEEDY"%s", vol_ip );
            if( is_dir_exist( temp_arr ) == 1 ){
//remove dir below /tmp/needy/vol_ip <---also get removed
                remove_dir( temp_arr );
                init_path( vol_ip, BASE_NEEDY );

                if( chdir( temp_arr ) != 0 ){
// show error in changing dir

```

```

    }

    }else if( is_dir_exist( temp_arr ) == 0 ){
//path is not initialised
        if( init_path( vol_ip, BASE_NEEDY ) < 0 )
            return -1; //else path is initialised

        if( chdir( temp_arr ) != 0 ){
// show error in changing dir
        }
    }
// ===== put into ipcwd =====
    if( getcwd(cwd, sizeof(cwd)) == NULL ){
//error in getting cwd
    }
    ipcwd_insert_pair( ipcwd, vol_ip, cwd );
}
//delete previous dir on needy side /tmp/vol/ip/output
else if( strcmp( sender_msg, "__INIT_DELETE_DIR_VOL__" ) == 0 ){

    sprintf( temp_arr, BASE_VOL "%s", vol_ip );
    if( is_dir_exist( temp_arr ) == 1 ){
        //remove dir below /tmp/needy/vol_ip <---also get removed
        remove_dir( temp_arr );
        init_path( vol_ip, BASE_VOL );

        if( chdir( temp_arr ) != 0 ){
            fprintf(stderr, "%s[%d]: Error %d in changing dir to %s
>%s\n", __FILE__, __LINE__, errno, temp_arr, strerror(errno) );
        }
    }

    }else if( is_dir_exist( temp_arr ) == 0 ){
//path is not initialised
        if( init_path( vol_ip, BASE_VOL ) < 0 )
            return -1;

//else path is initialised and we can proceed
// chdir() & show error if fail and return
    }
//===== if no entry <ip,cwd> found then =====
//===== put cwd in ipcwd( needy side ) =====
// get cwd and put in ipcwd hashtable
    ipcwd_insert_pair( ipcwd, vol_ip, cwd );
}
//move one dir higher
else if( strcmp(sender_msg, "__DIR_UP__" ) == 0 ){
// get the name of directory above
    strcpy( cwd, find_cwd( ipcwd, vol_ip ) );
    char *pch = strrchr( cwd, '/');
    cwd[ pch-cwd ] = '\0';

// chdir to directory
    above if got error show it
    if( getcwd(cwd, sizeof(cwd)) == NULL ){
// show error if getcwd not works
    }
// ===== update cwd in ipcwd =====
    ipcwd_insert_pair( ipcwd, vol_ip, cwd );

```

```

    }
    //close opened file
    else if( strcmp(sender_msg,"__FILE_CLOSE__") == 0 ){
        //===== remove entry from ipfd =====
        printf("received order: __FILE_CLOSE__ from: %s\n", vol_ip);
        fd = find_fd( ipfd, vol_ip );
        if( fd > 0 ){
            close( fd );
            ipfd_rem_pair( ipfd, vol_ip );
        }
    }
    //order to abort current task
    else if( strcmp(sender_msg,"__ABORT__") == 0 ){
//close the running script corresponding to given ip_addr
// close fd corresponding to ip and remove entry from ipfd
        fd = find_fd( ipfd, vol_ip );
        printf("got fd: %d", fd );
        if( fd > 0 ){
            close(fd);
            ipfd_rem_pair( ipfd, vol_ip );
        }
    }
//remove the dir /tmp/needy/ip
        sprintf( temp_arr, BASE_NEEDY"/%s", vol_ip );
// if dir temp_arr exists remove it and
// remove entry from ipcwd
// reply back with "__FINISH__" message
        send_msg( vol_ip, "__FINISH__");
//show entries
        show_entries();
    }
    //if finish signal is got
    else if( strcmp( sender_msg,"__FINISH__" ) == 0 ){
        printf("Ending communication with volunteer\n");
// ===== remove entry from ipcwd(on needy side ) =====
        fd = find_fd( ipfd, vol_ip );
        if( fd > 0 ){
            close( fd );
            ipfd_rem_pair( ipfd, vol_ip );
        }
        ipcwd_rem_pair( ipcwd, vol_ip );
        show_entries( );
    }
    else if( strstr( sender_msg,"__DONE_SENDING_DATA__") != NULL ){
        sprintf( temp_arr, BASE_NEEDY"/%s", vol_ip );

/*
get start_dir name in start_dir by reading name after "__DONE_SENDING_DATA__"
*/

        if( is_dir_exist(temp_arr) == 1 ){
//scripts dir doesn't exist; nothing to do; may be delete data from given vol_ip
            char temp_arr2[ MAX_DIR_SIZE ];
//here name of the root folder needs to be added
//cwd is also not helpful as it is BASE now
            sprintf( temp_arr2, BASE_NEEDY"/%s/%s/scripts", vol_ip, start_dir );

```

```

        if( is_dir_exist( temp_arr2 ) == 0 ){
//===== __NO_SCRIPT_DIR_FOUND__ is returned if scripts dir not found
            send_msg( vol_ip, "__NO_SCRIPT_DIR_FOUND__");

        }else if( is_dir_exist( temp_arr2 ) == 1 ) {

//for all script files in scripts dir feed each of them to system
// feed scripts to shell
            feed_scripts( temp_arr2 );
//after scripts had run send back /tmp/needy/ip/start_dir
            sprintf( temp_arr2, BASE_NEEDY"/%s/%s/output", vol_ip,
start_dir );

            send_result( temp_arr2, vol_ip );
        }else{
            printf("some weired error occured \n");
        }

    }else{
//some error has occurred in creating dir above scripts i.e. /tmp/needy/vol_ip
    }
//===== check if there is any fd opened; close it; remove ipfd entry=====

        show_entries();
    }
    //open file for writing
    else if( strstr( sender_msg, "__FILE__" ) != NULL ){

        printf("received order: __FILE__ from: %s\n", vol_ip);
//===== check if file is already opened for ip =====
//===== if fd is found; __PREV_ENTRY_FOUND_HTFD__ is returned

// else get name of file
        char filename[MAX_DIR_NAME_SIZE];
        char *pch = strstr( sender_msg, "__FILE__");
        strcpy( filename, pch+9 );
//check if file already exists if it exists open it otherwise create it and open it and //overwrite it
        strcpy( cwd, find_cwd ( ipcwd, vol_ip ) );
        sprintf( temp_arr, "%s/%s", cwd, filename );
        printf("order to create file %s\n", temp_arr );
        process_mask = umask(0);
        fd = open( temp_arr, O_CREAT | O_WRONLY, FILE_PERM );
        umask( process_mask );

//===== insert <ip,fd> in ipfd
    }

    }
    //create a dir and cd to it
    else if( strstr( sender_msg, "__DIR__" ) != NULL ){
//retrieve name of dir
        char dirname[MAX_DIR_NAME_SIZE];
        char *pch = strstr( sender_msg, "__DIR__");
        strcpy( dirname, pch+8 );

//check for cwd if error
//===== get cwd from ipcwd =====
        strcpy( cwd, find_cwd(ipcwd, vol_ip) );

```



```

        sprintf( temp_arr, "%s/%s", cwd,dirname );
        if( is_dir_exist( temp_arr ) == 1 ){
//dir exist just cd to it
            if( chdir( temp_arr ) != 0 ){
                fprintf(stderr,"%s[%d]: Error %d in changing dir to %s
>%s\n",__FILE__,__LINE__,errno,temp_arr,strerror(errno) );
                return -1;
            }
        }else if( is_dir_exist( temp_arr ) == 0 ){
//dir doesn't exist; create it
        }
        if( getcwd(cwd, sizeof(cwd)) == NULL ){
//===== update <ip, cwd> =====
        }
        ipcwd_insert_pair( ipcwd, vol_ip, cwd )
        }
// close file descriptor and      free up dynamic allocated memory
close( sender_fd );
free( vol_ip );
return 0;
}

int est_con( ){

    int my_fd = socket( AF_INET, SOCK_STREAM, 0);
    if( my_fd == -1 ){
//show error in creating socket
    }
// set fields for my_addr structure
    struct sockaddr_in my_addr;
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MY_PORT);
    char *my_ip_addr = get_my_ip_addr();
    if( my_ip_addr == NULL ){
        printf("could not find your ip on eth0 interface. You must set MY_ADDR to your ip and
recompile\n");
    }
    my_addr.sin_addr.s_addr = inet_addr( MY_ADDR );
    }else{
        my_addr.sin_addr.s_addr = inet_addr( my_ip_addr
        )
    }
    memset( &(my_addr.sin_zero),'\0', 8 );

    if( bind(my_fd, (struct sockaddr*)&my_addr, sizeof(my_addr) ) < 0 ){
// show error      "Could not bind to port\n"
    }

    listen( my_fd, MAX_ALLOWED_CLIENTS );

//
    printf("Waiting for sender!\n");
//

    return my_fd;
}

int main( ){

```

```

int fd = est_con();
ipfd = create_hash_table();
ipcwd = create_hash_table();
//when it starts it must multicast __FINISH__ which should be received by
//watchman to delete any previous ipfd and ipcwd entries which is not possible
//but at least it can send __FINISH__ to volunteer list after reading
//file /tmp/volunteer.txt
while(1){
    receive_msg( fd );    //lets run it without breaking
}
return 0;
}

```

A.2 Modifying and compiling the source code

Because this project is made using Anjuta IDE which creates its own configurations files for handling project; So you can open distapp.anjuta with Anjuta. It will show all included headers and source files for all programs in navigation pane. You can edit any source file. To compile the program in following steps:

1. open a terminal.
2. cd to project root directory.
3. Type ./configure
4. Note that you may have to make configure file executable before step 3 which can be done by command: `chmod +x configure`
5. Type make
6. cd to src file
7. Run ./distapp in one terminal window; it will open application GUI window
8. In another terminal window run ./watchman
9. In another terminal window run ./multicast_watchman
10. If you don't want to send any job; you don't have to use step 7; but steps 8, 9 are necessary to help needy

REFERENCES

- [1] Ssh client-server communication:
http://h71000.www7.hp.com/doc/83final/ba548_90007/ch01s04.html
- [2] Remote desktop protocol: http://en.wikipedia.org/wiki/Remote_Desktop_Protocol
- [3] Virtual Network computing:
http://en.wikipedia.org/wiki/Virtual_Network_Computing
- [4] Anjuta Tutorial:developer.gnome.org/anjuta-build-tutorial/
- [5] Glade Tutorial: <https://live.gnome.org/Glade/Tutorials>
- [6] Alexandre Duret-Lutz “Using GNU Autotools” May 16, 2010
- [7] Makefile tutorial: <http://www.opussoftware.com/tutorial/TutMakefile.htm>
- [8] Autotools tutorial: <http://markuskimius.wikidot.com/programming:tut:autotools:2>
- [9] IBM nfs: <http://www.ibm.com/developerworks/library/l-network-filesystems/>
- [10] Tcp and Udp: Data communication and Networking by Behrouz Forouzan chapter 23
- [11] Kameswari Chebrolu “IITK Socket programming” Dept. of Electrical Engineering, IIT Kanpur
- [12] Blaise Barney, Lawrence Livermore National Laboratory “Posix thread programming”: <https://computing.llnl.gov/tutorials/pthreads/>
- [13] Brian Masney, “Introduction to multithreaded programming”
<http://www.linuxjournal.com/article/3138>
- [14] Mark mitchell, Alex samuel, Jeffrey oldham “Advanced Linux Programming” chapter 3, 4, 5, 7
- [15] Robert Love “Linux System Programming” chapter 2, 5, 7
- [16] IBM Glib Tutorial: <http://www.ibm.com/developerworks/linux/tutorials/l-glib/>
- [17] GObject Library Reference: <http://developer.gnome.org/gobject/stable/>
- [18] Get the ip address of host machine: <http://stackoverflow.com/questions/212528/get-the-ip-address-of-the-machine>

[19] Recursively delete directory in c linux:

<http://stackoverflow.com/questions/3833581/recursive-file-delete-in-c-on-linux>

[20] GTK basic info: <http://www.gtk.org/>

[21] GCC information: http://en.wikipedia.org/wiki/GNU_Compiler_Collection

[22] Autotools flow diagram: <http://en.wikipedia.org/wiki/File:Autoconf-automake-process.svg>