

High Level Design & Low Level Design

**Document Control :**

**Project Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | **Approver Signature** |
| 19.10.2022 | 1.0 | Group 4 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Index**

1. Introduction 4

1.1. Project Purpose 4

1.2. Project Scope 4

1.3. Intended Use 4

1.4. Project Scope 4

2. Design Overview 5

2.1. Tasks and the assigned persons 5

3. Detailed System Design 6

3.1. DFD 0 & DFD 1 7

3.2. Flow charts 8

3.3 Use cases 9

3.4 Pseudocode 10

3.5 Validations 13

4. Detailed features and Requirements 13

4.1 Functional requirements 13

5. Reference 17

# Introduction

The introduction of the High Level Design (HLD) and Low Level Design (LLD) provides an overview of the entire document with purpose, intended audience, scope and key objectives. The purpose of this High Level Design (HLD) Document is to add the necessary detail to the **Remote Backup Utility Application’s** description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level. This application will manage backing up files from client to the server side.

**Project Purpose**

The purpose of this project is to make backing up of files simple and elegant. The client side code is very simple and based on options it performs various tasks. It can perform full, incremental, versioned and scheduled backups

## Intended Audience

This document is intended to be read by clients, designers , program and solution teams.

The intended audience for this application are anyone who want to backup their files remotely.

## Project Scope

This project helps user who want to backup their files to a particular server. The user and perform any of the 4 tasks for both files and directories i.e full backup incremental backup versioned backup or scheduled backup.

## Key Project Objectives

* Select the option for type of the backup
* Connect with the server
* Read files from the client
* Send the data to the server
* Write the received data from client in the backup folder

## Design Overview

Model City Vaccination Drive comprises of following modules:

|  |  |
| --- | --- |
| Name of the Module | Sending files from client |
| Handled by | Sriram Repaka |
| Description | A function to read files and send the data  of the file to the server |

|  |  |
| --- | --- |
| Name of the Module | Reading the arguments and formatting  the filenames at client |
| Handled by | Sriram Repaka |
| Description | Reading the arguments and making the  file path |

|  |  |
| --- | --- |
| Name of the Module | Full backup client |
| Handled by | Shagir Husain |
| Description | This module is responsible for full  backup of files and directories |

|  |  |
| --- | --- |
| Name of the Module | Incremental backup client |
| Handled by | Prinshu Raj |
| Description | This module checks if the particular files have been modified since last backup and backups only it has been updated. For  both files and directories |

|  |  |
| --- | --- |
| Name of the Module | Versioned backup client |
| Handled by | Sarthak Srivasthava |
| Description | This module reads files and sends them  to server. |

|  |  |
| --- | --- |
| Name of the Module | Sheduled Backup client |
| Handled by | Sriram Repaka |
| Description | This module takes user input creates a  cronjob |

|  |  |
| --- | --- |
| Name of the Module | Full backup server |
| Handled by | Shagir Husain |
| Description | Writes the received files to the backup  folder |

|  |  |
| --- | --- |
| Name of the Module | Incremental backup server |
| Handled by | Aryan Tiwari |
| Description | This simply updates the received files in  the backup folder |

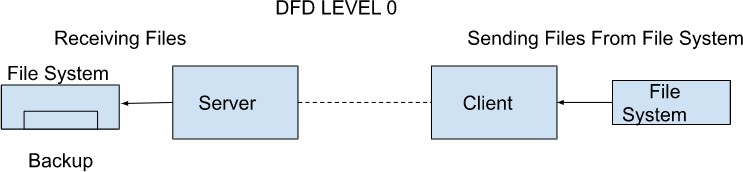
|  |  |
| --- | --- |
| Name of the Module | Versioned Backup server |
| Handled by | Tushar Gautam |
| Description | Makes a copy of the files in the backup folder with time appended to the  filename |

|  |  |
| --- | --- |
| Name of the Module | Scheduled Backup server |
| Handled by | Sriram Repaka |
| Description | Acknowledges the user that a scheduled  backup has been added. |

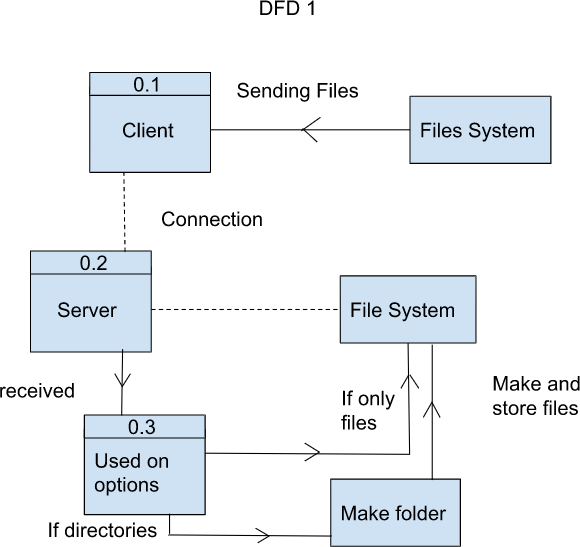
# Detailed System Design

## Data Flow Diagram

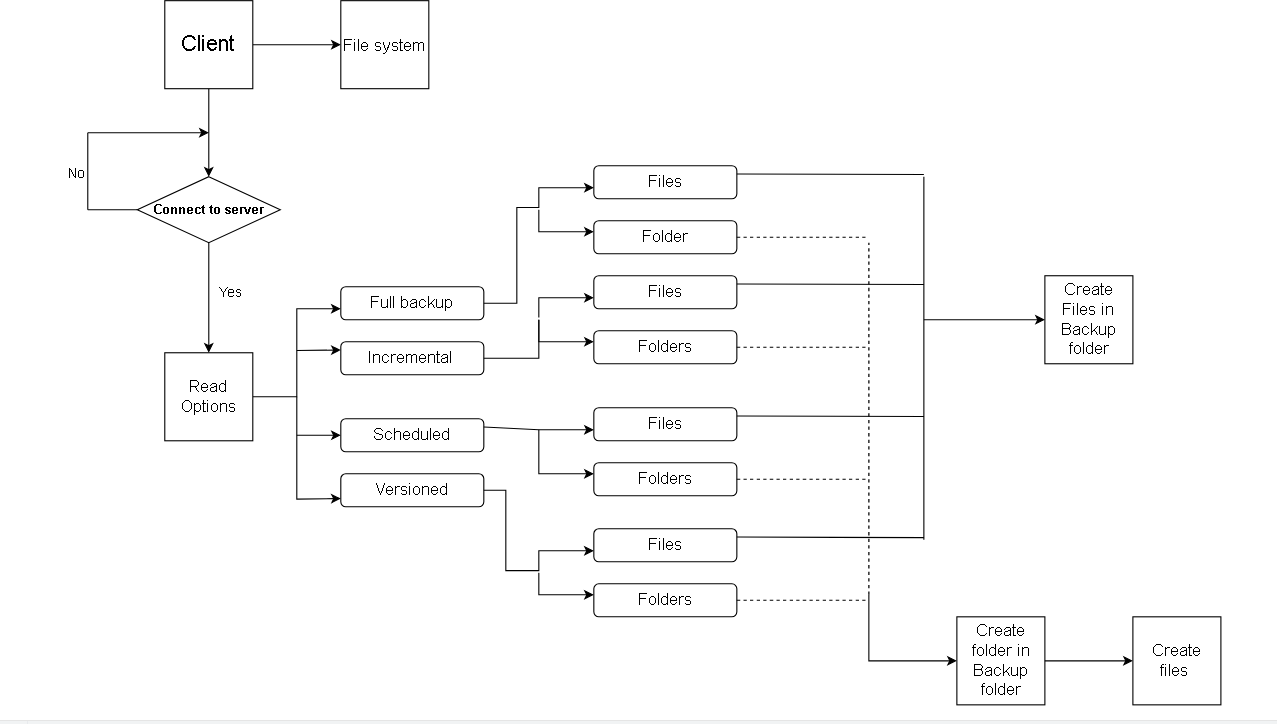
3.1.1. Level 0

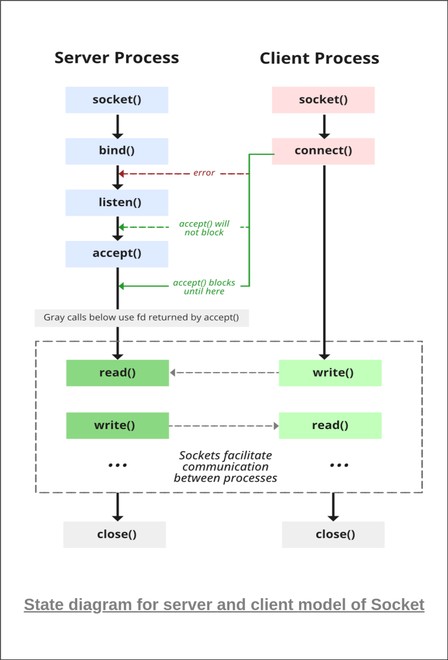


3.1.2 level 1



## Flowcharts

**Usecase Diagram**



**Pseudocode**

**Client**

Pesudocode for Remote Backup Utility First function: MAIN FUNCTION

-----Creating Sockets to send and recieve information -----

----main function begins-----

BEGIN

Reading current time storing the MON & DOM in a string CREATING A SOCKET

Creatinga socket and acknowledging the user CONNECTING TO THE SERVER

checking weather the socket is connected or not IF successful

acknowledge the user

ELSE

END

READ argc, argv

//validating arguments

//assigning argv[1] argv[2] to option and directory

//sending argc argv[1] argv[2] to the server

STARTING A IF ELSE LADDER FOR ALL THE OPTIONS IF OPTION == -ff

IF ARGC != number THEN

PRINT "error!! enter correct number of arguments"

END ELSE

Directory

reading argv[3] argv[4] argv[argc-1]

sending the argv[]'s (filenames) to server

making the file pathname by conatinating argv[]'s and calling the SEND\_FILE() with socket name and filename

ELSE IF OPTION == -fd

IF ARGC != number THEN

PRINT "error!! enter correct number of arguments"

END ELSE

reading argv[2] argv[3] argv[argc-1]

SENDing the directory paths to the server OPENING the argv[]'s(directories)

READING all the filenames and concatinating to directories to

create filepath

calling the SEND\_FILE() with socket name and filepath

ELSE IF OPTION == -if

IF ARGC != number THEN

PRINT "error!! enter correct number of arguments"

END ELSE

Directory

reading argv[3] argv[4] argv[argc-1]

making the file pathname by conatinating argv[]'s and

check wheather the file has been modified today IF yes

calling the SEND\_FILE() with socket name and

filename

ELSE IF OPTION == -id

IF ARGC != number THEN

PRINT "error!! enter correct number of arguments"

END ELSE

filenames

reading argv[3] argv[4] argv[argc-1]

SENDing the directory paths to the server

OPENING the directory and reading all the filenames

making the file pathname by conatinating Directories and

check weather the file has been modified or not IF yes

calling the SEND\_FILE() with socket name and

filename

ELSE IF OPTION == -vf

IF ARGC != number THEN

END ELSE

Directory

PRINT "error!! enter correct number of arguments"

reading argv[2] argv[3] argv[argc-1]

making the file pathname by conatinating argv[]'s and calling the SEND\_FILE() with socket name and filename

ELSE IF OPTION == -vd

IF ARGC != number THEN

PRINT "error!! enter correct number of arguments"

END ELSE

filenames

reading argv[3] argv[4] argv[argc-1]

SENDing the directory paths to the server

OPENING the directory and reading all the filenames

making the file pathname by conatinating Directories and calling the SEND\_FILE() with name and socketname

ELSE IF OPTION == -sf

IF ARGC != number THEN

PRINT "error!! enter correct number of arguments"

END ELSE

reading argv[2] argv[3] argv[argc-1]

creating a cronjob based on argv's concayinating the cronjob to the config file

copying the cronjob schedule "\* \* \* \* \*" to string SENDing the schedule string to the server

ELSE IF OPTION == -sd

IF ARGC != number THEN

PRINT "error!! enter correct number of arguments"

END ELSE

reading argv[3] argv[4] argv[argc-1]

creating a cronjob based on argv's concayinating the cronjob to the config file

copying the cronjob schedule "\* \* \* \* \*" to string SENDing the schedule string to the server

----main function ends----

----function definitions----

FUNCTION SEND\_FILE

PASS IN: FILENAME, SOCKETNAME PASS OUT: NOTHING

OPEN FILE USEING the FILENAME

read the contents of the file

SEND the contents of the file to the server using teh SOCKETNAME FLIGHT\_LIST -> TAIL is NULL

ENDFUNCTION

**Validations**

* Users must enter the IP address and PORT numbers in the client and server codes
* Users must enter the backup directory in the server code
* The users must enter valid filenames and directories.

# Detailed Features and Requirements

## Functional Requirements

* + 1. Full backup for files full():

(client)

Firstly we check whether the option is -ff or not and whether more than 4 arguments have been passed or not. The directory of the files specified is passed as the 3rd argument and files are specified from the 4th argument onwards. Once we check the said conditions we start by sending the filenames to the server and then we concatenate the directories and the filename then we call the send\_file() function.

(server)

Coming to the server side we firstly receive the filenames then we concatenate the backup directory and the filename essentially creating the path of the backup file we also add the backup\_of tag to the filename received from the client. After creating the path we call the write\_file() function to write the data received from the client using the send\_file() function.

* + 1. Full backup for folders full\_dir():

(client)

Firstly we check whether the option is -fd or not and whether more than 3 arguments have been passed or not. The directory is passed from the 3rd argument. Once we check the said conditions we start by sending the Directory paths to the server and then using opendir() and readdir() we go through all the filenames in the specified directories. Then we concatenate the directory and the filenames and we call the send\_file() function.

(server)

Coming to the server side we firstly receive the Directory paths, then we concatenate the backup directory and the paths received. Now we use mkdir() to create the sub-directory in the backup directory essentially creating the directories for the files we are going to receive from the client. After creating the Directories simply repeat what we did for full backup of files.

* + 1. Incremental backup for files incremental():

(client)

Firstly we check whether the option is -if or not and whether more than 4 arguments have been passed or not. The directory of the files specified is passed as the 3rd argument and files are specified from the 4th argument onwards. Once we check the said conditions we start by concatenating the directories and the filename then using filestat we create a file status variable then using st\_ctime we get the time when the file was modified and it it was modifies today if it was modifies today then we first send the filename to server and then we call the send\_file() function to send only that particular file.

(server)

Coming to the server side we firstly receive the filenames then we concatenate the backup directory and the filename which will be creating the path of the backup file we also add the "backup\_of" tag to the filename received from the client. After creating the path we call the write\_file() function to write the data received from the client using the send\_file() function.

* + 1. Incremental backup for folders incremental\_dir():

(client)

Firstly we check whether the option is -id or not and whether more than 3 arguments have been passed or not. The directory is passed from the 3rd argument. Once we check the said conditions we start by sending the Directory paths to the server and then using opendir() and readdir() we go through all the filenames in the specified directories and check weather they have been updated today. If yes then we concatenate the directory and the filenames and we call the send\_file() function.

(server)

Coming to the server side we firstly receive the Directory paths, then we concatenate the backup directory and the paths received. Now we use mkdir() to create the sub-directory in the backup directory which will be creating the directories for the files we are going to receive from the client. After creating the Directories simply repeat what we did for full/incremental backup of files

* + 1. Versioned backup for files versioned():

(client)

Firstly we check whether the option is -ff or not and whether more than 4 arguments have been passed or not. The directory of the files specified is passed as the 3rd argument and files are specified from the 4th argument onwards. Once we check the said conditions we start by sending the filenames to the server and then we concatenate the directories and the filename then we call the send\_file() function.

(server)

Coming to the server side we firstly receive the filenames then we concatenate the present time till the minutes to the filename then we concatenate the backup directory and the filename essentially creating the path of the backup file we also add the "backup\_at" and "\_of" tag to the filename received from the client. After creating the path we call the write\_file() function to write the data received from the client using the send\_file() function.

* + 1. Versioned backup for folders versioned\_dir():

(client)

Firstly we check whether the option is -vd or not and whether more than 3 arguments have been passed or not. Then directory is passed as the 3rd argument. Once we check the said conditions we start by sending the Directory paths to the server and then using opendir() and readdir() we go through all the filenames in the specified directories. Then we concatenate the directory and the filenames and we call the send\_file() function.

(server)

Coming to the server side we firstly receive the Directory paths, then we concatenate the backup directory and the paths received. Now we use mkdir() to create the sub-directory in the backup directory essentially creating the directories for the files we are going to receive from the client. After creating the Directories we simply repeat what we did for versioned backup for files. We add the time to the filename and create the versioned file of the received files.

* + 1. Scheduled backup for files scheduled():

(client)

We scan the arguments first which include the timing of the backup i.e the schedule of the back up in the form of "mm hh dom mon dow" the we start creating a string for the cronjob we create the cronjob such that at the scheduled time the we cd to the directory where our client is and then we run the client executable file with -ff option with the files specified. We then append this cron job to the config file. I have to mention that the config file already has a cronjob that updates the contab every hour. The client also sends the cronjob time to the server

(Server)

Server receives the cornjob time and it appends its config file which tells when the server should run so as to be available for the client at that time.

* + 1. Scheduled backup for folders scheduled\_dir():

(client)

We scan the arguments first which include the timing of the backup i.e the schedule of the back up in the form of "mm hh dom mon dow" the we start creating a string for the cronjob we create the cronjob such that at the scheduled time the we cd to the directory where our client is and then we run the client executable file with -fd option with the directories specified. We then append this cron job to the config file.

(Server)

Server receives the cornjob time and it appends its config file which telling when the server should run so as to be available for the client at that time.

1. **Reference**

**<https://idiotdeveloper.com/file-transfer-using-tcp-socket-in-c/>**

**[https://stackoverflow.com/questions/55765083/running-gcov-on-a-](https://stackoverflow.com/questions/55765083/running-gcov-on-a-program-with-arguments) [program-with-arguments](https://stackoverflow.com/questions/55765083/running-gcov-on-a-program-with-arguments)**

**<https://www.geeksforgeeks.org/socket-programming-cc/>**