QT BASED ONLINE DATA CAPTURING UNDER LINUX PLATFORM

This internship report submitted in partial fulfillment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Adusumilli Aditya Koushik (221910401001)

Under the esteemed guidance of

Mr . S Francis Xavier

Assistant Professor

DEPARTMENT OF ELECTRICAL, ELECTRONICS AND COMMUNICATION ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

(Estd. u/s 3 of UGC act 1956 & Accredited by NAAC with "A+" Grade)

Hyderabad - 502329

(2019 - 2023)

DEPARTMENT OF ELECTRICAL, ELECTRONICS AND COMMUNICATION ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project work entitled "**QT BASED ONLINE DATA CAPTURING UNDER LINUX PLATFORM**" is a bonafide work carried out by **Adusumilli Aditya Koushik (221910401001)** submitted in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Electronics and Communication Engineering**, GITAM School of Technology, GITAM (Deemed to be University), Hyderabad has successfully completed the industrial training at **"Research Centre Imarat (RCI-DRDO)"** during the academic year 2019-2023.

INTERNSHIP GUIDE	**HEAD OF THE DEPARTMENT**
Mr. S Francis Xavier	**Dr. T Madhavi**
Assistant Professor	**Professor**
Dept. of EECE	**Dept. of EECE**
GITAM School of Technology	**GITAM School of Technology**

DEPARTMENT OF ELECTRICAL, ELECTRONICS AND COMMUNICATION ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

We hereby declare that the project work entitled "**QT BASED ONLINE DATA CAPTURING UNDER LINUX PLATFORM**" is an original work done in the Department of Electrical, Electronics, and Communication Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of **B.Tech. in Electronics and Communication Engineering**. The work has not been submitted to any other college or university for the award of any degree or diploma.

Registration No.	Name	Signature
221910401001	Adusumilli Aditya Koushik	

ISO 9001:2008 Lab

आर सी आई

ISO 9001 REGISTERED

DNV

MGMT. SYS. RvA C024

DNV Certification B.V., The Netherlands

RESEARCH CENTRE IMARAT
Defence R&D Organisation (DRDO)
HYDERABAD – 500 069
Phone No : 040 – 24306164
Fax No : 040 – 24306160

TO WHOM IT MAY CONCERN

This is to certify that following students from GITAM University, Hyderabad:

1. **SAI RAM BADRI VISHAL K**
2. **ADUSUMILLI ADITYA KOUSHIK**
3. **AKULA SAI SRUJAN**
4. **PASUPULATI VIJAY KRISHNA**

Of fourth year B.Tech Electronics and Communication Engineering (ECE) have been successfully completed 12 weeks internship.

During this period their conduct and progress found satisfactory.

Date: 27/06/2022.

[illegible]

(Rajesh Shankar Karvande)
Scientist 'F', DHILS, RCI
Hyderabad
Ph. No. 09490956052

राजेश शंकर करवंदे / RAJESH SHANKAR KARVANDE
वैज्ञानिक / Scientist
डी.आर.डी.ओ., रक्षा मंत्रालय, भारत सरकार
DRDO, Ministry of Defence, Govt. of India
आर सी आई, हैदराबाद /RCI, Hyderabad-500 069.

ACKNOWLEDGEMENT

The internship opportunity I had at Research Centre Imarat (RCI) was a great chance for learning and professional development. I am grateful for having a chance to meet so many wonderful people and professionals who led us through this internship period. I express my deepest gratitude and special thanks to Mr. Rajesh Shankar Karvande, Scientist F, who guided me and kept me on the correct path and allowed me to carry out my project at their esteemed organization and extend during the training.

I would like to thank our project guide Mr. S Francis Xavier, Assistant Professor Dept. of EECE for his stimulating guidance and profuse assistance. I shall always cherish our association with him for his guidance, encouragement, and valuable suggestions throughout the progress of this work. I consider it a great privilege to work under his guidance and constant support.

I would like to thank our Internship Coordinator Ms. MVN Madhavi Latha, Assistant Professor, for helping to complete the project by taking frequent reviews and for their valuable suggestions throughout the progress of this work.

I consider it a privilege to express our deepest gratitude to Prof. T Madhavi, Head of the Department, Department of EECE, for her valuable suggestions and constant motivation that immensely helped us to successfully complete the project work.

Prof. N Seetharamaiah, Associate Director, School of Technology, for inspiring us to learn new technologies and tools.

I perceive this opportunity as a significant milestone in our career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on improvement to attain desired career objectives.

ABSTRACT

Hardware-In-Loop simulation technology is a unique facility where all the software along with hardware is simulated. The project work given to us is to develop the system which is Ethernet-based telemetry data capturing during HILS runs. These speeds help in transferring many data formats like video and audio to the receiver on the ground with minimal delay.

To display the values on the screen which we get from the OBC of the missile, we developed a GUI with appropriate parameters using a C++ programming language in Qt5 Creator software. We also wrote server-client code to establish communication via ethernet cable and load data into a text file. The data in this file is shared with Qt GUI. Ubuntu OS is used for the development of the entire project.

After the integration, the data captured from the OBC is displayed and updated in real time until the connection is closed.

CONTENTS

List of Figures

List of Tables

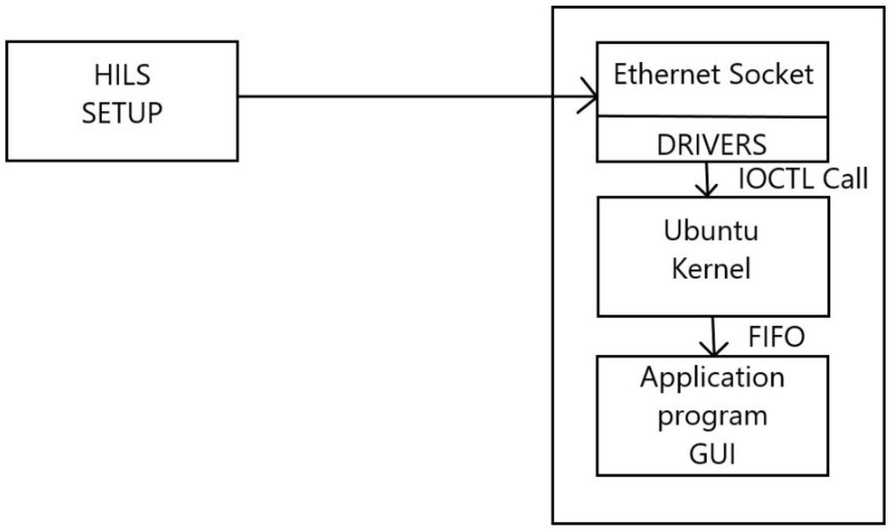


Fig 1.1 Introduction

1. UBUNTU

Just like Windows, iOS, and Mac OS, Linux is an operating system. In fact, one of the popular platforms Android is powered by the Linux operating system. An operating system is a software that manages all of the hardware resources associated with your desktop or laptop. To put it simply, the operating system manages the communication between your software and your hardware. Without the operating system (OS), it is very difficult to communicate with the processor.

1.1. UBUNTU INTRODUCTION

Ubuntu is a Linux-based operating system. It is designed for computers, smartphones, and network servers. The system is developed by a UK-based company called Canonical Ltd. All the principles used to develop the Ubuntu software are based on the principles of Open-Source software development.

In the Linux system, the shell is a command-line interface that interprets a user's commands and script files and tells the server's operating system what to do with them. After logging into your server with SSH, you'll be connected to a remote shell, or command prompt, where you can issue commands to the server. Two common ways to set up your Ubuntu servers are through SSH keys or by installing 'Webmin' on your ubuntu server, SSH, or secure shell, is an encrypted protocol used to administer and communicate with servers.

1.2. FEATURES OF UBUNTU

- The desktop version of Ubuntu supports all the normal software on Windows such as Firefox, Chrome, VLC, etc.
- It supports the office suite called LibreOffice.
- Ubuntu has an in-built email software called Thunderbird, which gives the user access to emails such as Exchange, Gmail, Hotmail, etc.
- There are a host of free applications for users to view and edit photos.
- There are also applications to manage videos and it also allows the users to share videos. It is easy to find content on Ubuntu with the smart search facility.

1.3. UBUNTU FLAVOURS

Flavours of Ubuntu offer a unique way to experience Ubuntu, each with its own choice of default applications and settings. Ubuntu flavours are owned and developed by members of our global community and backed by the full Ubuntu archive for packages and updates.

1.4. LINUX TERMINOLOGY AND COMMANDS

A. Terminology:

The Linux OS system incorporates several different components, including:

Bootloader: A bootloader is responsible for managing the boot process of the computer and for starting the Linux kernel. It can also be used to manage systems that boot more than one OS.

Kernel: The core of the Linux system, the kernel handles network access, schedules processes or applications, manages basic peripheral devices and oversees all file system services. The Linux kernel is the software that interfaces directly with the computer hardware.

Daemons: This is a program that runs in the background, handling requests for service. A web server running on a Linux server depends on a daemon, usually named httpd, to listen for web server requests.

Graphical server: This is the software that controls how graphics are displayed on a computer. Without a graphical server, users can only interact with the Linux system through a command-line interface. The X Windows system, also known as X11 or X, is the most common graphical server for Linux, though not the only one. X runs as a server daemon on the system and is called upon by applications when graphical output is required.

Applications: This is the software that is installed during and after the initial Linux installation. Most Linux distributions include thousands of different applications, including both for a networked server and for desktop use.

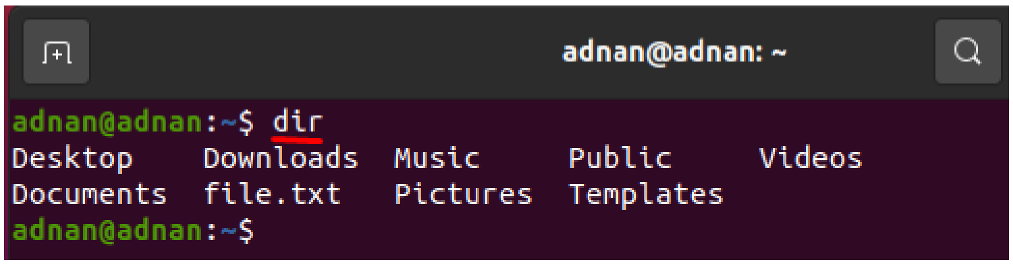
B. LINUX Commands

Command 1: pwd

This command refers to the present working directory in which you are operating; in simpler words, in which your terminal is open. To check PWD, execute the 'pwd' keyword in your terminal and hit enter; the command of PWD is written below along with the result of that command.

Command 2: dir

The dir command is used to print (on the terminal) all the available directories in the present working directory.

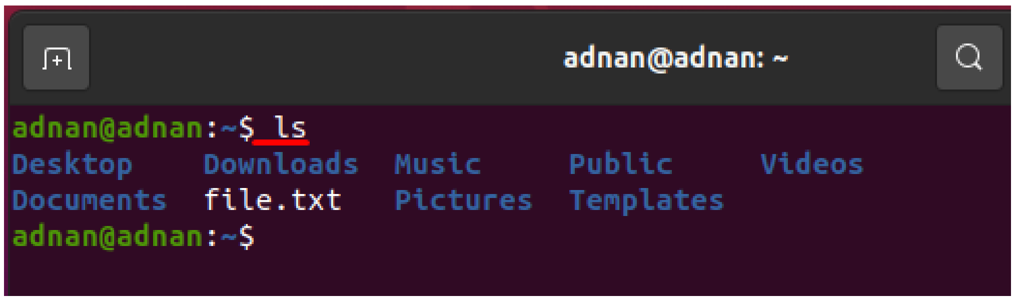


```
adnan@adnan: ~
adnan@adnan:~$ dir
Desktop     Downloads  Music     Public     Videos
Documents   file.txt   Pictures  Templates
adnan@adnan:~$
```

Fig. 1.2 dir command

Command 3: ls

This command is used to list down all the directories and files inside the present working directory (or you can give the path of a specific directory); the ls command can be executed as shown below

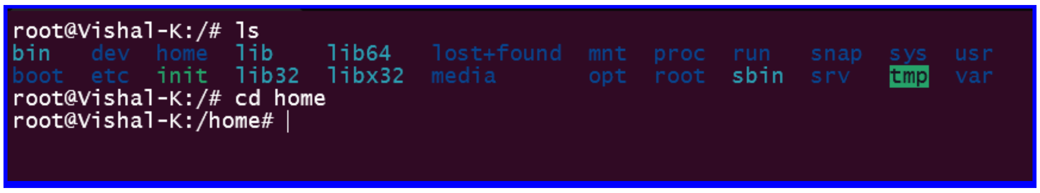


```
adnan@adnan: ~
adnan@adnan:~$ ls
Desktop     Downloads  Music     Public     Videos
Documents   file.txt   Pictures  Templates
adnan@adnan:~$
```

Fig. 1.3 ls command

Command 4: cd

One of the most used commands of Ubuntu; you can change the directories in the terminal using the “cd” command.



```
root@Vishal-K:/# ls
bin   dev  home  lib    lib64   lost+found  mnt  proc  run   snap  sys  usr
boot  etc  init  lib32  libx32  media       opt  root  sbin  srv   tmp  var
root@Vishal-K:/# cd home
root@Vishal-K:/home# |
```

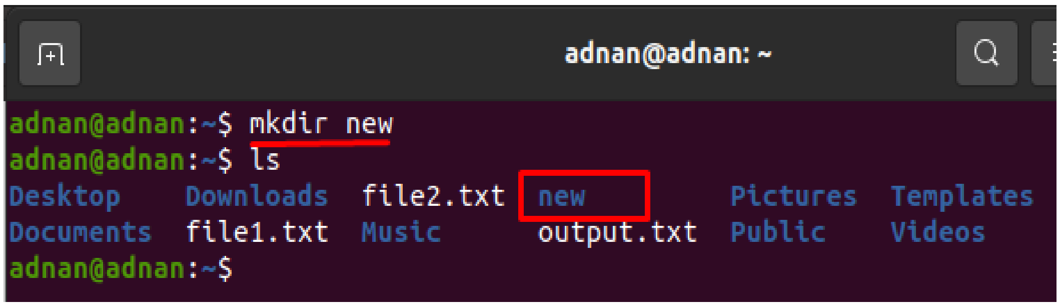
Fig. 1.4 cd command

Command 5: cat

This command is used to show the content of any file: For instance, the following command will display the content inside “**file1.txt**”.

Command 6: mkdir

The above-mentioned command will make a directory in your pwd; for example, the following command will make the directory “**new**” in pwd.

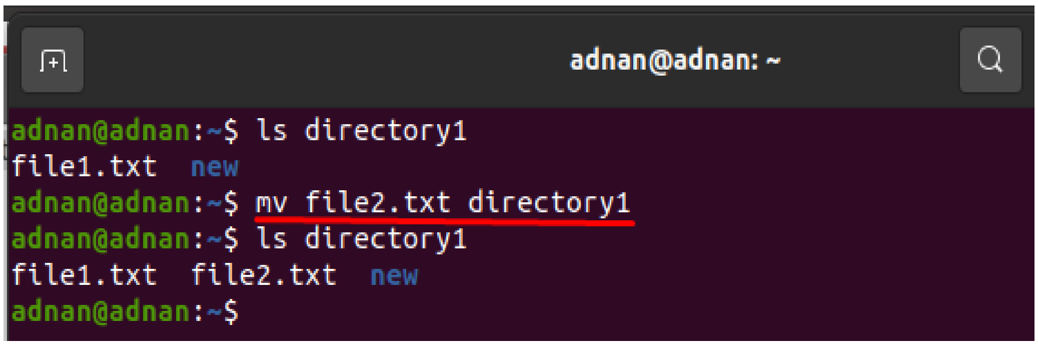


```
adnan@adnan:~$ mkdir new
adnan@adnan:~$ ls
Desktop    Downloads  file2.txt  new         Pictures  Templates
Documents  file1.txt  Music      output.txt  Public    Videos
adnan@adnan:~$
```

Fig. 1.5 mkdir command

Command 7: mv

You can use this command to move files around the computer, and you can also rename files or directories inside a specific directory: the command given below will move the “**file2.txt**” to “**directory1**”.



```
adnan@adnan:~$ ls directory1
file1.txt  new
adnan@adnan:~$ mv file2.txt directory1
adnan@adnan:~$ ls directory1
file1.txt  file2.txt  new
adnan@adnan:~$
```

Fig. 1.6 mv command

Command 8: uname

You can use the command to get the release number, version of Linux, and much more. The “**-a**” flag is used to get detailed information.

Command 9: apt-get or -apt

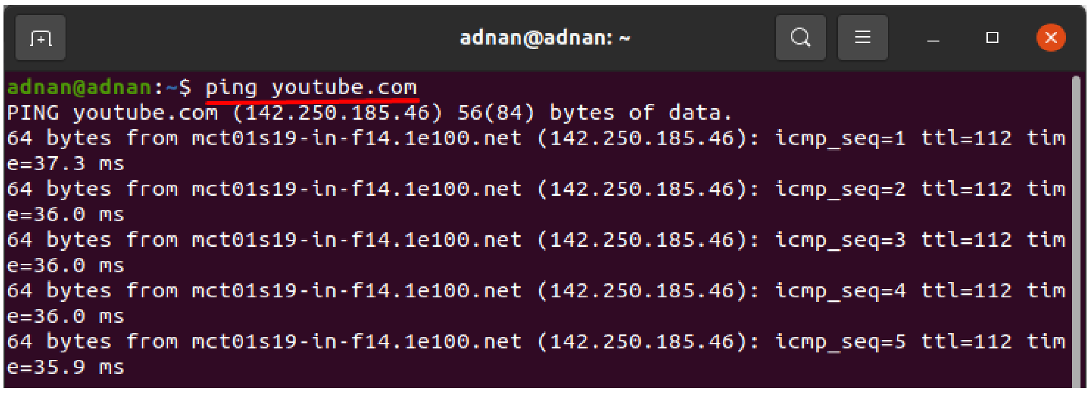
This is one of the most important and most used commands of Ubuntu that works with Ubuntu Advanced Packaging Tool (APT); you can use this “**-apt-get**” or “**-apt**” to install or remove packages, or you can perform other maintenance tasks. The “**apt**” require ‘sudo’ privileges to successfully execute the command.

Command 10: man

The man command will help you to get the complete user manual of any specific command; for instance, the following command will list down the detailed usage of the “**cat**” command:

Command 11: ping

You can use the ping command to check the connectivity to your server; for example, the command below will ping to YouTube and also prints the response time:



```
adnan@adnan:~$ ping youtube.com
PING youtube.com (142.250.185.46) 56(84) bytes of data.
64 bytes from mct01s19-in-f14.1e100.net (142.250.185.46): icmp_seq=1 ttl=112 tim
e=37.3 ms
64 bytes from mct01s19-in-f14.1e100.net (142.250.185.46): icmp_seq=2 ttl=112 tim
e=36.0 ms
64 bytes from mct01s19-in-f14.1e100.net (142.250.185.46): icmp_seq=3 ttl=112 tim
e=36.0 ms
64 bytes from mct01s19-in-f14.1e100.net (142.250.185.46): icmp_seq=4 ttl=112 tim
e=36.0 ms
64 bytes from mct01s19-in-f14.1e100.net (142.250.185.46): icmp_seq=5 ttl=112 tim
e=35.9 ms
```

Fig. 1.7 ping command

1.5. HOW THE LINUX OPERATING SYSTEM WORKS

The Linux OS follows a modular design that is the key to its many variations and distributions. All Linux distributions are based on the Linux kernel, but they can differ depending on factors such as:

Kernel version: Distributions can be configured with more recent versions to incorporate newer features or with older versions to be more stable.

Kernel modules: This is software that can be loaded and unloaded into the kernel to extend functionality without rebooting. Kernel modules are often used to support:

1. Device drivers, use code that controls how attached devices operate;
2. File system drivers, use code that controls how the kernel works with different file systems.
3. System calls, use code that controls how programs request services from the kernel.

Configuration options: Linux kernels compiled with configuration options set to include only device or file system drivers are used for some specialized distributions; for example, compiling a kernel for a wireless device without any wired network device drivers.

The Linux kernel is the one thing that all systems running Linux have in common. Linux works by: While the kernel may be almost identical with some divergence for configuration and compilation differences the user experience can vary widely, depending on how the Linux system is being used.

For example, some Linux use cases with widely different user experiences include:

Thin clients: which enable users to access a rich desktop environment from a lightweight device. This includes Raspberry Pi single-card computers and Google Chromebooks.

Depending on the application, Linux can be optimized for different purposes such as:

1. Networking performance
2. Computation performance
3. Deployment on specific hardware platforms
4. Deployment on systems with limited memory, storage or computing

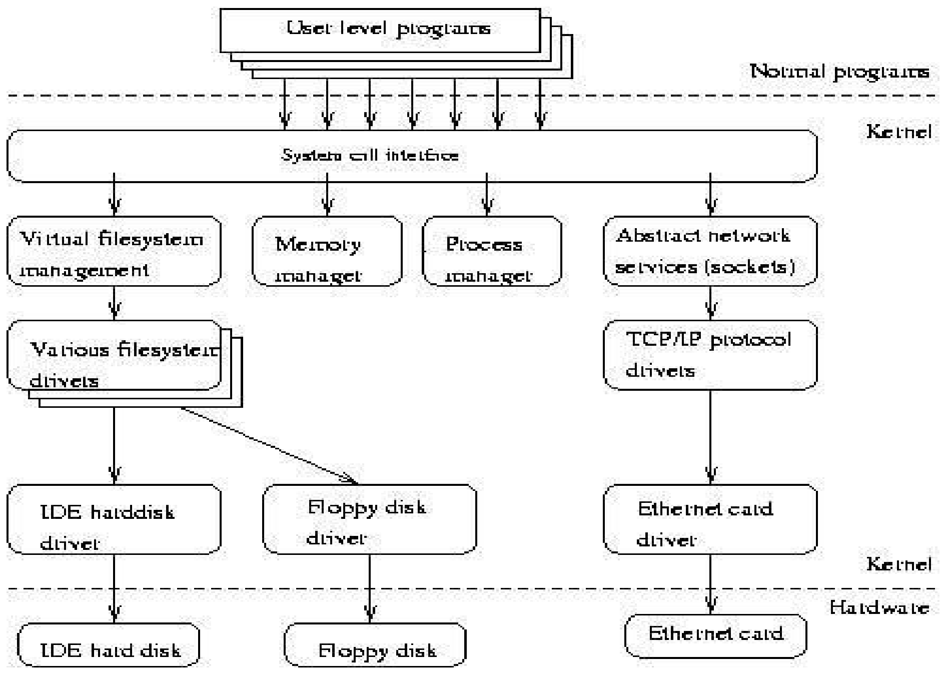


Fig. 1.8 Working of Linux OS

1.6. THE PROS AND CONS OF USING LINUX

1. Open-source software
2. Licensing costs
3. Reliability
4. Support costs
5. Proprietary software

1.7. WHY LINUX IS BETTER THAN WINDOWS

Advantages of Linux over Windows

1. Open-Source Nature
2. Secure and private
3. Can revive older computers
4. Perfect for programmers
5. Software Updates
6. Variety of distributions
7. Free to Use
8. Better Community Support
9. Reliability

2. SOCKET PROGRAMMING

2.1. INTRODUCTION

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket

The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP. A wide range of operating systems support socket APIs. i5/OS sockets support multiple means of transport and networking protocols. Socket system functions and socket network functions are threadsafe.

Programmers who use Integrated Language Environment (ILE) C can refer to this topic collection to develop socket applications. You can also code to the sockets API from other ILE languages, such as RPG.

2.2. TCP and UDP

User Datagram Protocol (UDP)

UDP is a simple transport-layer protocol. The application writes a message to a UDP socket, which is then encapsulated in a UDP datagram, which is further encapsulated in an IP datagram, which is sent to the destination.

There is no guarantee that a UDP will reach the destination, that the order of the datagrams will be preserved across the network or that datagrams arrive only once.

The problem of UDP is its lack of reliability: if a datagram reaches its final destination but the checksum detects an error, or if the datagram is dropped in the network, it is not automatically retransmitted. Each UDP datagram is characterized by a length. The length of a datagram is passed to the receiving application along with the data.

No connection is established between the client and the server and, for this reason, we say that UDP provides a connection-less service. It is described in RFC 768.

Transmission Control Protocol (TCP)

TCP provides a connection-oriented service since it is based on connections between clients and servers. TCP provides reliability. When a TCP client sends data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically retransmits the data and waits for a longer period of time.

We have mentioned that UDP datagrams are characterized by a length. TCP is instead a byte-stream protocol, without any boundaries at all.

TCP is described in RFC 793, RFC 1323, RFC 2581 and RFC 3390.

Hostname and port are used to specify transport endpoints.

1. **Socket**: the communication object.
2. **TCP properties:** reliable, connection-oriented, byte-stream, connection established before application-level protocols exchange information, two-way communication.
3. **UDP properties:** unreliable, packet-switched, packet data, no connection overhead, application-level protocols exchange information immediately, two-way communication.

A socket connection is a 4-tuple -- (HostA, PortA, HostB, PortB) -- uniquely defining the connection.

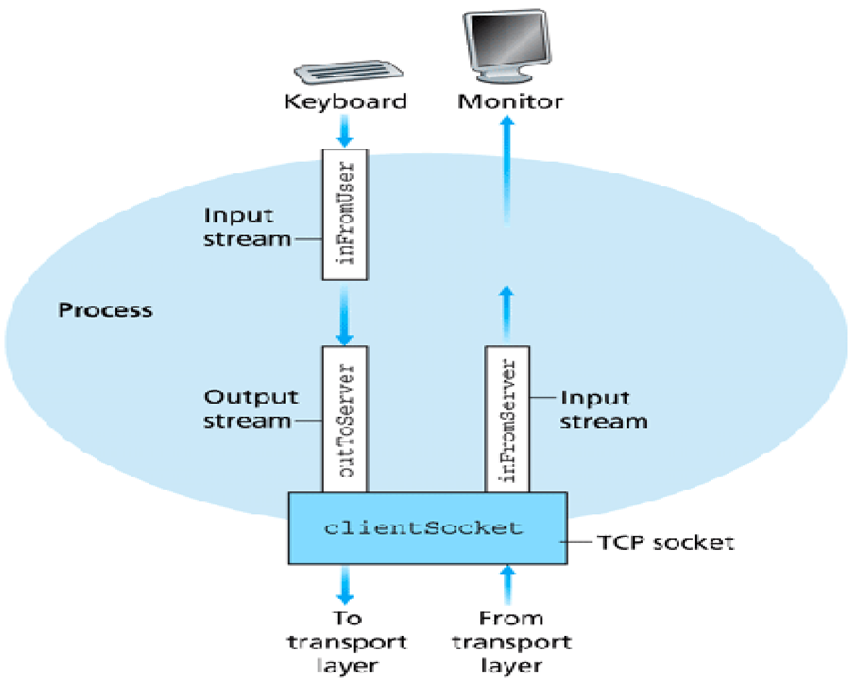


Fig 2.1 TCP client communication model

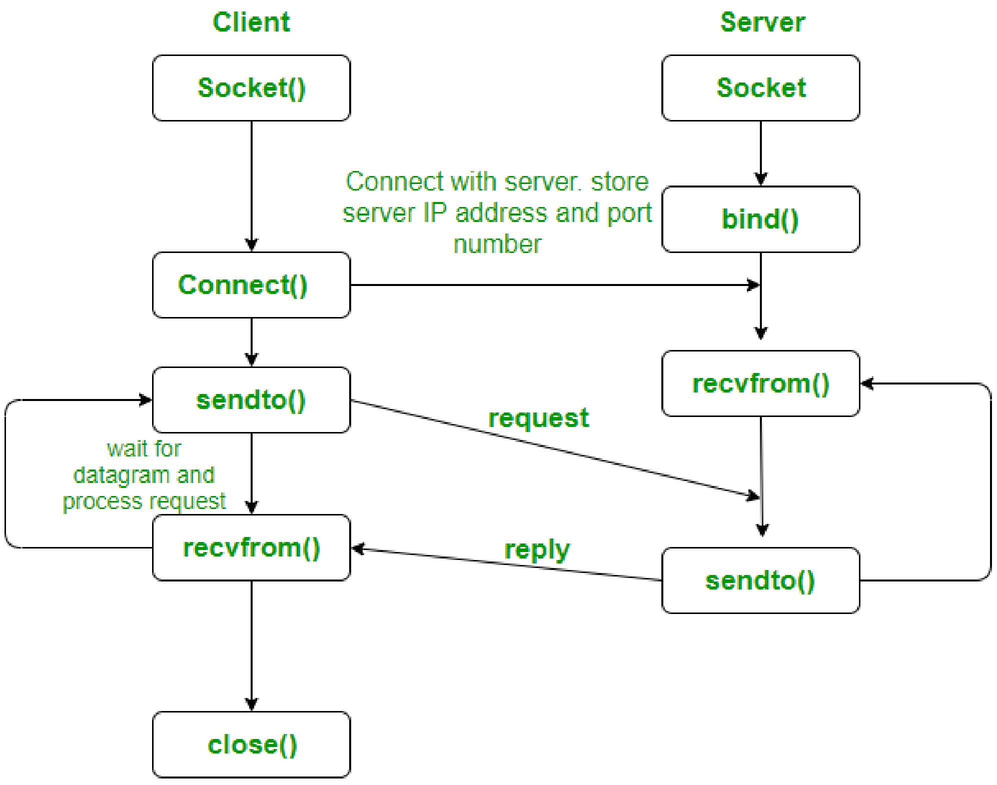

Client
Server
Socket()
Socket
Connect with server. store server IP address and port number
bind()
Connect()
recvfrom()
sendto()
request
wait for datagram and process request
recvfrom()
reply
sendto()
close()

Fig 2.2 UDP Server communication model

A socket programming interface provides the routines required for inter-process communication between applications, either on the local system or spread in a distributed, TCP/IP-based network environment. Once a peer-to-peer connection is established, a socket descriptor is used to uniquely identify the connection. The socket descriptor itself is a task-specific numerical value.

One end of a peer-to-peer connection of a TCP/IP-based distributed network application described by a socket is uniquely defined by

- Internet address
- Communication protocol
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- Port
- user defined ports

2.3. Client-Server Relationship:

The client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients.

A client does not share any of its resources but requests a server's content or service function.

Clients, therefore, initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client-server model are Email, network printing, and the World Wide Web.

The Client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients,

The Server

To put simply, a server is just a computer. The real magic of a server lies within its role in providing data to other computers. Two ways a server may go about serving data to other computers is via a local area network, or a wide area network, both over the internet.

The Client

On the flip side of this relationship, we have the client. The client, too, is a computer in and of itself. This computer acts as a client when it is retrieving data from the server. For instance, when you check your emails on your laptop, your laptop becomes the client that accesses the data from a mail server.

The Relationship

Now that you have a better understanding of each part of this relationship, we can begin to dive deeper into its process. Just like many other relationships, each part serves a specific role. For instance, the server provides a function, or service, to one or many clients. These clients are initiating requests for this service. This relationship is only possible when both roles are in effect.

The sending back and forth of messages between the two is known as a request-response messaging pattern. The client will send a request and the server must return a response. The few disadvantages to a model such as this would be the ability for a server to get easily overloaded if too many requests are coming in from clients. Also, if the server does fail, then no requests can be granted.

Inference

The client server relationship is a very important relationship in the world of technology and computer science. If you are beginning a career in software engineering, it is important to understand the client server model and recognize both its advantages and drawbacks. In conclusion, most everything every application or website that is developed is supported by this model. The client server relationship is an important foundation to many applications.

2.4. Socket Programming

1. Create a socket:

The first thing to do is create a socket.

Here is a code sample:

```
#include<stdio.h>
#include<sys/socket.h>

int main(int argc , char *argv[])
{
	int socket_desc;
	socket_desc = socket(AF_INET , SOCK_STREAM , 0);

	if (socket_desc == -1)
	{
		printf("Could not create socket");
	}

	return 0;
}
```

2. Send data over socket:

Function ‘write’ will simply send the data. It needs the socket descriptor, the data to send and its size.Function ‘read’ will simply receive the sent data. It needs the socket descriptor, buffer and its size.

3. Close socket

Function close is used to close the socket. We need to include ‘unistd.h’ header file to use this ‘close()’ function.

Get IP address of hostname.

When connecting to a remote host, it is necessary to have its IP address. Function ‘gethostbyname’ is used for this purpose. It takes the domain name as the parameter and returns a structure of type ‘hostent’. This structure has the IP information. It is present in ‘netdb.h’. Let’s have a look at this structure,

```
#include<stdio.h> //printf
#include<string.h> //strcpy
#include<sys/socket.h>
#include<netdb.h>       //hostent
#include<arpa/inet.h>

int main(int argc , char *argv[])
{
        char *hostname = "www.google.com";
        char ip[100];
        struct hostent *he;
        struct in_addr **addr_list;
        int i;

        if ( (he = gethostbyname( hostname ) ) == NULL)
        {
                //gethostbyname failed
                herror("gethostbyname");
                return 1;
        }

            //Cast the h_addr_list to in_addr , since h_addr_list also has the ip address in
long format only
        addr_list = (struct in_addr **) he->h_addr_list;

        for (i = 0; addr_list[i] != NULL; i++)
        {
                //Return the first one;
                strcpy(ip , inet_ntoa(*addr_list[i]) );
        }

        printf("%s resolved to : %s" , hostname , ip);
        return 0;
```

3. QT SOFTWARE

3.1. Introduction to QT software:

Qt (pronounced "cute") is a cross-platform software for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as

Linux, Windows, macOS, Android or embedded systems with little or no change in the underlying codebase while still being a native application with native capabilities and speed.

Qt is currently being developed by The Qt Company, a publicly listed company, and the Qt Project under open-source governance, involving individual developers and organizations working to advance Qt. Qt is available under both commercial licenses and open-source GPL 2.0, GPL 3.0, and LGPL 3.0 licenses.

Qt is used for developing graphical user interfaces (GUIs) and multi-platform applications that run on all major desktop platforms and most mobile or embedded platforms. Most GUI programs created with Qt have a native-looking interface, in which case Qt is classified as a widget toolkit. Non-GUI programs can also be developed, such as command-line tools and consoles for servers. An example of such a non-GUI program using Qt is the Cutelyst web framework.

Qt supports various compilers, including the GCC C++ compiler, the Visual Studio suite, PHP via an extension for PHP5, and has extensive internationalization support. Qt also provides Qt Quick, that includes a declarative scripting language called QML that allows using JavaScript to provide the logic. With Qt Quick, rapid application development for mobile devices became possible, while logic can still be written with native code as well to achieve the best possible performance.

Other features include SQL database access, XML parsing, JSON parsing, thread management and network support

Fig. 3.1 Qt logo

3.2. QT Modules and Tools

QT modules:

Table 3.1 Qt Modules

MODULE	DESCRIPTION
Qt core	The only required Qt module, containing classes used by other modules, including the meta-object system, concurrency and threading, containers, event system, plugins and I/O facilities.
Qt GUI	The central GUI module. In Qt 5 this module now depends on OpenGL, but no longer contains any widget classes.
Qt Widget	Contains classes for classic widget-based GUI applications and the QSceneGraph classes. Was split off from QtGui n Qt 5.
Qt QML	Module for QML and javascript languages.
Qt Quick	The module for GUI applications is written using QML2.
Qt Quick controls	Widget-like controls for Qt Quick intended mainly for desktop applications.
Qt Quick layouts	Layouts for arranging items in Qt Quick.
Qt Network	Network abstraction layer. Complete with support for TCP, UDP, HTTP, TLS, SSL (in Qt 4) and SPDY
Qt Multimedia	Classes for audio, video, radio and camera functionality.
Qt Multimedia widgets	The widgets from Qt Multimedia.
Qt SQL	Contains classes for database integration using SQL.
Qt Test	Classes for unit testing Qt applications and libraries.

Qt Tools:

Qt comes with its own set of tools to ease cross-platform development, which can otherwise be cumbersome due to different sets of development tools.

Qt Creator is a cross-platform IDE for C++ and QML. Qt Designer's GUI layout/design functionality is integrated into the IDE, although Qt Designer can still be started as a standalone tool.

The following are official Qt tools:

-Qt Creator, Qt Linguist, Qt Assistant, Qt Designer, Qt Quick Designer, qmlscene qmake, rcc, moc

3.3. Signals and Slots:

In Qt, we have an alternative to the callback technique; we use signals and slots. A signal is emitted when a particular event occurs. Qt's widgets have many predefined signals, but we can always subclass widgets to add our own signals to them. A slot is a function that is called in response to a particular signal. Qt's widgets have many pre-defined slots, but it is common practice to subclass widgets and add your own slots so that you can handle the signals.

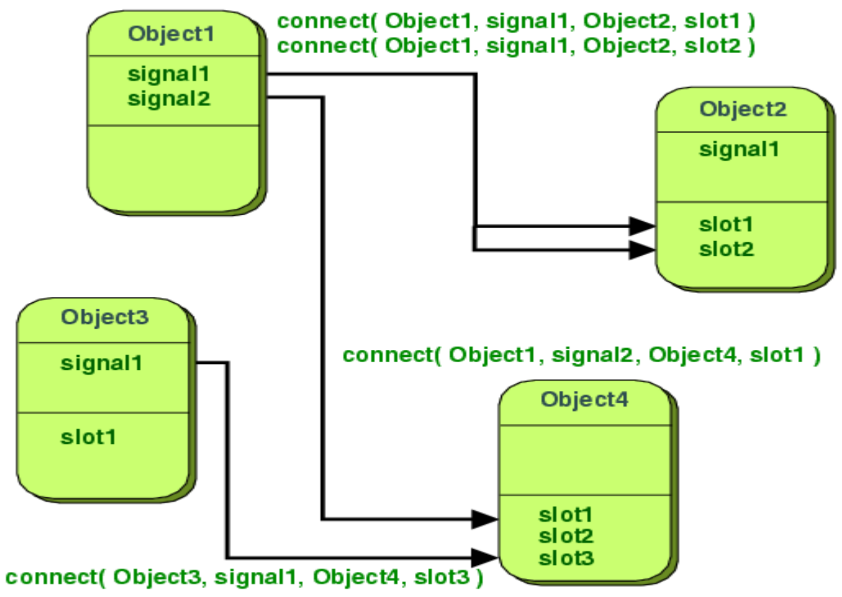

Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

Fig. 3.2 Signals and Slots

The signals and slots mechanism is type safe: The signature of a signal must match the signature of the receiving slot. (In fact, a slot may have a shorter signature than the signal it receives because it can ignore extra arguments.) Since the signatures are compatible, the compiler can help us detect type mismatches when using the function pointer-based syntax. The string-based SIGNAL and SLOT syntax will detect type mismatches at runtime. Signals and slots are loosely coupled: A class which emits a signal neither knows nor cares which slots receive the signal. Qt's signals and slots mechanism ensures that if you connect a signal to a slot, the slot will be called with the signal's parameters at the right time. Signals and slots can take any number of arguments of any type. They are completely type-safe.

Signals:

Signals are emitted by an object when its internal state has changed in some way that might be interesting to the object's client or owner. Signals are public access functions and can be emitted from anywhere, but we recommend only emitting them from the class that defines the signal and its subclasses.

When a signal is emitted, the slots connected to it are usually executed immediately, just like a normal function call. When this happens, the signals and slots mechanism is totally independent of any GUI event loop. Execution of the code following the emit statement will occur once all slots have returned. The situation is slightly different when using queued connections; in such a case, the code following the emit keyword will continue immediately, and the slots will be executed later. If several slots are connected to one signal, the slots will be executed one after the other, in the order, they have been connected, when the signal is emitted.

Slots:

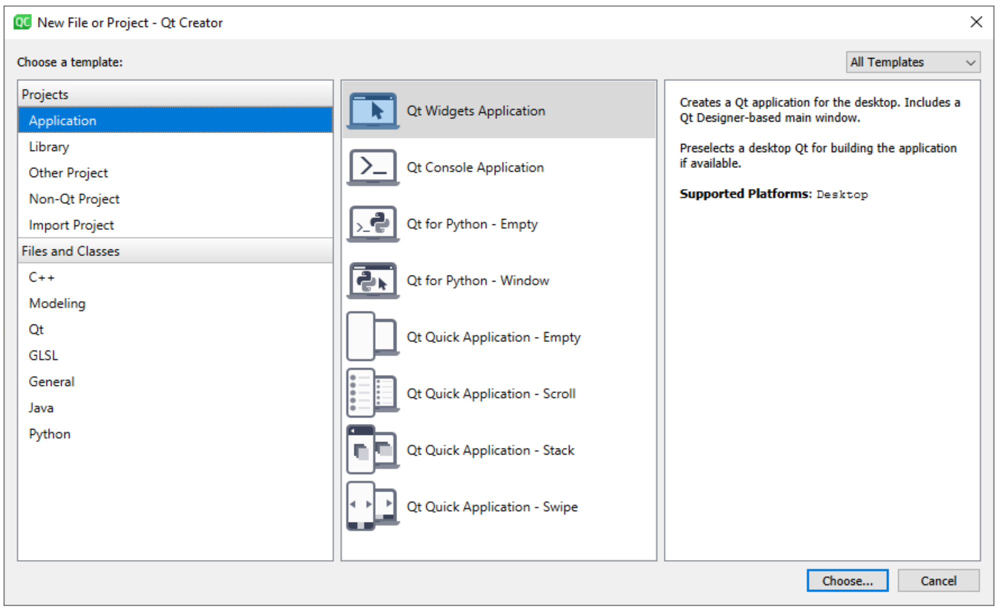
A slot is called when a signal connected to it is emitted. Slots are normal C++ functions and can be called normally; their only special feature is that signals can be connected to them.

Since slots are normal member functions, they follow the normal C++ rules when called directly. However, as slots, they can be invoked by any component, regardless of its access level, via a signal-slot connection.
Compared to callbacks, signals and slots are slightly slower because of the increased flexibility they provide, although the difference for real applications is insignificant. In general, emitting a signal that is connected to some slots, is approximately ten times slower than calling the receivers directly, with non-virtual function calls. This is the overhead required to locate the connection object, to safely iterate over all connections (i.e. checking that subsequent receivers have not been destroyed during the emission), and to marshal any parameters in a generic fashion. While ten non-virtual function calls may sound like a lot, it's much less overhead than any new or delete operation, for example. As soon as you perform a string, vector or list operation that behind the scene requires new or delete, the signals and slots overhead are only responsible for a very small proportion of the complete function call costs.

3.4. Programs and Examples:

Initializing a project in Qt Creator:

1. Click in File and select 'New Project'. Choose 'Qt Widgets Application' in the Application tab.



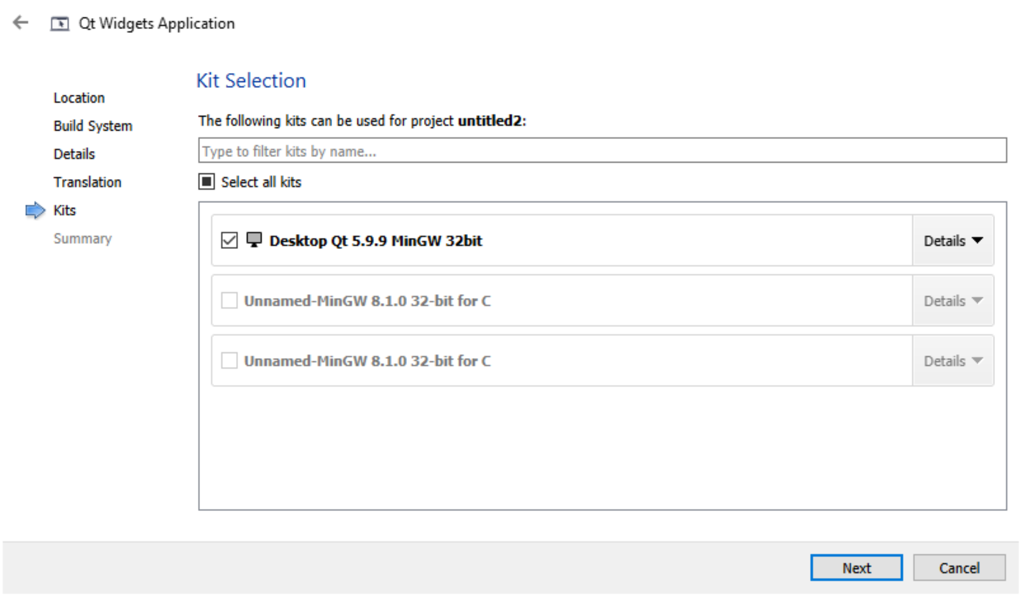
Creates a Qt application for the desktop. Includes a Qt Designer-based main window.

Preselects a desktop Qt for building the application if available.

Supported Platforms: `Desktop`

Fig. 3.3 Qt Project selection window

2. After naming your project, select qmake in 'Build System'.
3. In the details tab select Base Class as QMainWindow and click on next.
4. Make sure you select an appropriate kit for the execution of the project.



The following kits can be used for project **untitled2**:

Select all kits

☑ **Desktop Qt 5.9.9 MinGW 32bit**

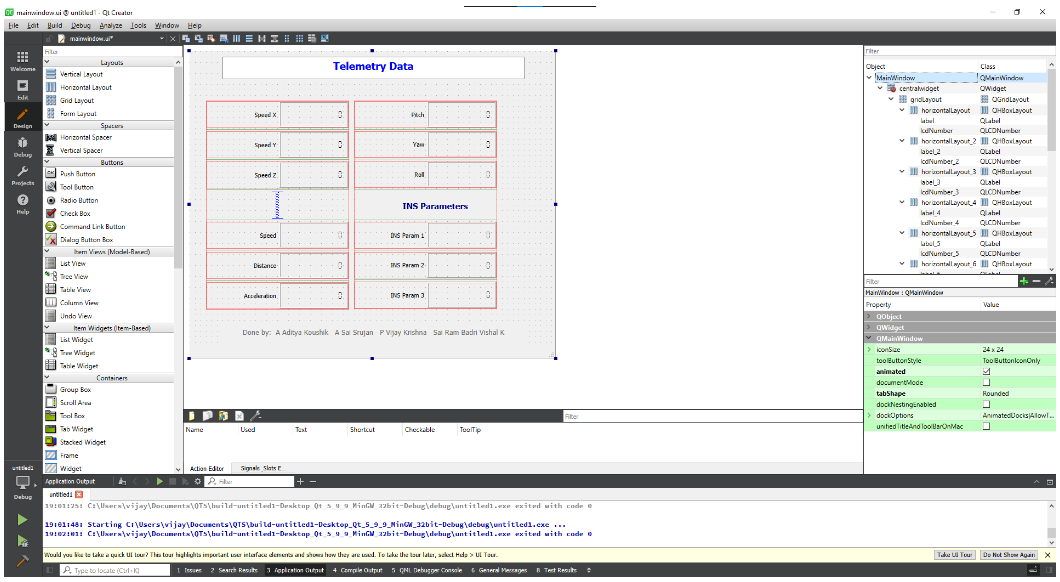
☐ Unnamed-MinGW 8.1.0 32-bit for C

☐ Unnamed-MinGW 8.1.0 32-bit for C

Fig. 3.4 Widget Creation

5. Click ‘Finish' in the final tab to start your project.

GUI created to Display Telemetry Data window:


mainwindow.ui @ untitled1 - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Telemetry Data
Speed X
Speed Y
Speed Z
Speed
Distance
Acceleration
Pitch
Yaw
Roll
INS Parameters
INS Param 1
INS Param 2
INS Param 3
Done by: A Aditya Koushik A Sai Srujan P Vijay Krishna Sai Ram Badri Vishal K
Application Output
untitled1
1 Issues 2 Search Results 3 Application Output 4 Compile Output 5 QML Debugger Console 6 General Messages 8 Test Results

```
19:01:25: C:\Users\vijay\Documents\QT5\build-untitled1-Desktop_Qt_5_9_9_MinGW_32bit-Debug\debug\untitled1.exe exited with code 0

19:01:48: Starting C:\Users\vijay\Documents\QT5\build-untitled1-Desktop_Qt_5_9_9_MinGW_32bit-Debug\debug\untitled1.exe ...
19:02:01: C:\Users\vijay\Documents\QT5\build-untitled1-Desktop_Qt_5_9_9_MinGW_32bit-Debug\debug\untitled1.exe exited with code 0
```

Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.

Fig.3.5 Qt Window

4. INTEGRATION

Ubuntu OS is used for the development of the entire project.

The server-client code is written to establish communication via ethernet cable and load data into a text file. The data in this file is shared with Qt GUI.

Client Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>

void error(const char*msg)
  {
    perror(msg);
    exit(0);
  }
int main()
  {
     int sockfd,portno,n;
    struct sockaddr_in serv_addr;
     struct hostent*server;
     char buffer[256];
     portno=atoi("1235");
     sockfd =socket(AF_INET,SOCK_STREAM,0);
    if(sockfd<0)
        error("ERROR opening socket");
  server=gethostbyname("127.0.0.1");
  if(server == NULL)
  {
  fprintf(stderr,"Error,no such host");
  }
  bzero((char*)&serv_addr,sizeof(serv_addr));
```

```
    serv_addr.sin_family=AF_INET;
    bcopy((char*)server->h_addr,(char*)&serv_addr.sin_addr.s_addr,server->h_length
);
    serv_addr.sin_port =htons(portno);
    if(connect(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
           error("Connection Failed");

          bzero(buffer,255);
             FILE *fp;
             int ch=0;
             fp =fopen("s.txt","w");
             int words;
          read(sockfd,&words,sizeof(int));
             while(ch!=words)
             {
                    read(sockfd,buffer,255);
                    fprintf(fp,"%s\n",buffer);
                    ch ++;
             }
fclose(fp);
close(sockfd);
return 0;
    }
```

Server Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<ctype.h>
void error(const char*msg)
{
        perror(msg);
        exit(1);
}
int main()
{
        int sockfd,newsockfd,portno,n;
        char buffer[255];
        struct sockaddr_in serv_addr,cli_addr;
        socklen_t clilen;
        sockfd=socket(AF_INET,SOCK_STREAM,0);
        if(sockfd<0)
{
        error("Error opening Socket.");
}
bzero((char*)&serv_addr,sizeof(serv_addr));
portno=atoi("1235");
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=INADDR_ANY;
serv_addr.sin_port=htons(portno);
if(bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr))<0)
       error("Binding Failed.");
listen(sockfd,5);
clilen=sizeof(cli_addr);
newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clilen);
if(newsockfd<0)
error("Error on Accept");

        bzero(buffer,255);
```

```
          FILE *f;
                    int words=0;
                    char c;
                    f=fopen("r.txt","r");
                    while((c=getc(f))!=EOF)
                    {
                              fscanf(f,"%s",buffer);
                              if(isspace(c) || c=='\t')
                              words++;
                    }
                    words++;
                    write(newsockfd,&words,sizeof(int));
                    rewind(f);
                    char ch;
                    while(ch != EOF)
                    {
                              fscanf(f,"%s",buffer);
                              write(newsockfd,buffer,255);
                              ch=fgetc(f);
                    }
fclose(f);
close(newsockfd);
close(sockfd);
return 0;
}
```

Qt Codes for interfacing widgets with data:

mainwindow.cpp:

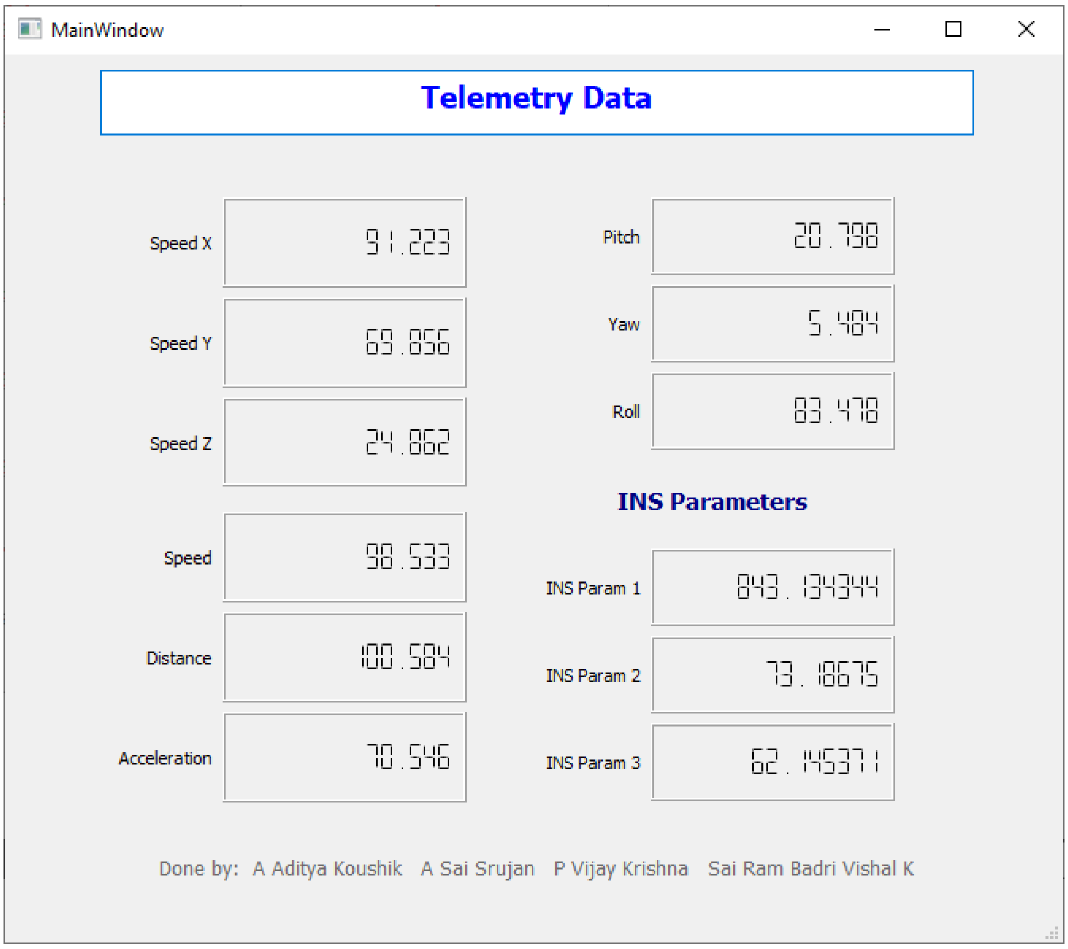
```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QFile>
#include <QTextStream>
#include <QMessageBox>
#include <QDebug>
#include <algorithm>
using namespace std;

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);
   timer=new QTimer(this);
   connect(timer,SIGNAL(timeout()),this,SLOT(myFun()));
   timer->start(100);
}

void MainWindow::myFun()
{
   string name;

      QFile file("C:\\Users\\vijay\\Documents\\QT5\\TextDocument.txt");
      if (!file.open(QIODevice::ReadOnly))
         QMessageBox::information(0,"info",file.errorString());

      QTextStream in(&file);
      ui->lcdNumber->display(in.readLine());
      ui->lcdNumber_2->display(in.readLine());
      ui->lcdNumber_3->display(in.readLine());
      ui->lcdNumber_4->display(in.readLine());
      ui->lcdNumber_5->display(in.readLine());
      ui->lcdNumber_6->display(in.readLine());
      ui->lcdNumber_7->display(in.readLine());
      ui->lcdNumber_8->display(in.readLine());
      ui->lcdNumber_9->display(in.readLine());
```

```
        ui->lcdNumber_10->display(in.readLine());
        ui->lcdNumber_11->display(in.readLine());
        ui->lcdNumber_12->display(in.readLine());
    }

    MainWindow::~MainWindow()
    {
      delete ui;  }
```

mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include<QTimer>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE


class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   MainWindow(QWidget *parent = nullptr);
   ~MainWindow();
public slots:
   void myFun();

private:
   Ui::MainWindow *ui;
   QTimer *timer;
};
#endif // MAINWINDOW_H
```

The Final Window


MainWindow

Telemetry Data

Speed X	91.223
Speed Y	69.856
Speed Z	24.862
Speed	98.533
Distance	100.584
Acceleration	70.546
Pitch	20.798
Yaw	5.484
Roll	83.478

INS Parameters

INS Param 1	843.134344
INS Param 2	73.18675
INS Param 3	62.145371

Done by: A Aditya Koushik A Sai Srujan P Vijay Krishna Sai Ram Badri Vishal K

Fig 4.1 Final Main Window

5. CONCLUSION

During the last 3 months, we worked on many technologies i.e., Ubuntu OS, Ethernet and QT. All three are different architectures. The socket programming based on Ethernet is done under the Ubuntu environment and packets have been tested. The GUI for the interactive display of parameters has been proved under the QT environment. The integration part is done at RCI and it has been demonstrated successfully. With this demonstration, our project is closed.

For future implementations, the exact design of the GUI will be done by RCI engineers and the packets will be developed as per their protocol. So, the project given to us is successfully completed.

6. REMARKS

In conclusion, the internship was a useful experience. We have found out what our strengths and weaknesses are; we gained new knowledge and skills and met many new people. Needless to say, the technical aspects of the work we have done are not flawless and could be improved provided enough time. As someone with no prior experience with Socket Programming, we believe our time spent in research and discovering it was well worth it and contributed to finding an acceptable solution to build a model. Two main things that we have learned are the importance of our time-management skills and self-motivation.

7. REFERENCES

[1] Summerfield, M. (2007). Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming (paperback). Pearson Education.

[2] Dalheimer, M. (2002). Programming with QT: Writing portable GUI applications on Unix and Win32. " O'Reilly Media, Inc.".

[3] QT C++ GUI Tutorial For Beginners. (n.d.). [Video]. YouTube. Retrieved November 6, 2022.

[4] Think and Learn. (2017, August 19). *Transferring a text file in Socket Programming in TCP | Socket Programming* | YouTube.