Integration of Modbus C Library Using Go

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology
in

Computer Science and Engineering/Information Technology

By

Mayank Sharma (151313)

Under the supervision of

Mr. Yuvaraj Laxman Patil

to

Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Certificate

I hereby declare that the work presented in this report entitled "**Implementation of Modbus C library Using golang "** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from February 17 2019 to May 17 2019 under the supervision of **Mr Yuvaraj Laxman Patil** Architect Wipro limited The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Mayank Sharma

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Yuvaraj Laxman Patil	**Mr Vijay Natarajan**
Architect	Competency Manager
Wipro Limited	Wipro Limited
Dated:	Dated

Acknowledgements

I express my profound gratitude and indebtedness to of **Mr Yuvaraj Laxman Patil** Architect Wipro limited for introducing the present topic and for their inspiring intellectual guidance, constructive criticism and valuable suggestion throughout this journey.

I am also thankful to all **Mr Vijay Natarajan** Competency Manager Wipro Limited for his constant motivation and helping me bring in improvements in the project.

Finally I would like to thank my family and friends for their constant support. Without their contribution it would have been impossible to complete my work.

Date
Mayank Sharma

Table of Contents

1. Introduction……………………………………………………………………..1

1.1 Introduction…………………………………...……………………………1

1.2 Problem Statement……………………………..………..…………….…1

1.3 Objectives……………………………………………………….............2

1.4 Methodology…………………………………………………………….2

1.5 Organization of the Report……………………………………………...3

2. Literature Survey……………………………………………………………...4

2.1 Modbus Protocol…………………………………………………...….4

2.1.1 Transactions on Modbus Networks………..………………………..…5

2.1.2 Query Response Cycle……………………………………….………..…6

2.1.3 Serial Transmission Modes………………………………………………7

2.1.3.1 ASCII Mode…………………………………….………………….…7

2.1.3.2 RTU Mode……………………………………………….……8

2.1.4 Modbus Message Framing……………………………………….…8

2.1.5 Contents of Data field…………………………………………………..10

2.1.6 Contents of Error Checking Field…………………………..………..10

2.1.7 Serial Transmission of Characters………………………………………11

2.1.8 Error Checking methods…………………………………………….12

2.1.8.1 Parity Checking…………………………………………........13

2.1.8.2 LRC Checking…………………………………………………14

2.1.8.3 CRC Checking…………………………………..…………14

2.2 Modbus TCP/IP……………………………………………….………….15

2.2.1 Determinism…………………………………………………………...17

2.3 Introduction to golang……………………………………………………….19

2.3.1 Why use Golang…………………………………………………19

3. System Development

3.1 Closer look at the Implementation…………………………………………...27

3.2 Issues with implementing Cgo………………………………………........29

3.3 Why use Cgo despite the issues…………………………………………...32

3.4 Dealing with the issues……………………………………………………...33

4. Performance Analysis……………………………………………………...36

5. Conclusions…………………………….…………………………..38

5.1 Applications…………………………………………………………….…38

5.2 Future Scope……………..………………………………………………..39

6. References……………………..……………………………………………40

7. APPENDICES…………………………………………………………………41

List of Figures

Fig 2.1 Modbus Protocol Application……………………………………………..……5
Fig 2.2 Master Slave Query Response Cycle……………………………………………...6
Fig 2.3 ASCII Message Frame………………………………………………….................9
Fig 2.4 RTU Message Frame…………………………………………………………...9
Fig 2.5 Bit Order (ASII)…………………………………………………………………11
Fig 2.6 Bit Order (RTU)…………………………………………………...............11
Fig 2.7 Modbus TCP data packet……………………………………………………...16
Fig 2.8 Golang logo……………………………………………………………..............18
Fig 2.9 Working of Goroutines…………………………………………….…………21
Fig 2.10 Compilation sequence in Java and Python………………………….................23
Fig 2.11 Compilation sequence in C/C++……………………………………..................23

Fig 3.1 Sample CGo code…………………………………………………………..…27
Fig 3.2 Cgo header file...…...28
Fig 3.3 Cgo function call…………………………………….……………………...…28
Fig 3.4 Cgo function call alternate method…………….……………………………...29
Fig 3.5 Go to C……………..……………………………………………………………29
Fig 3.6 Strings methods in C…………………….………………………………….…29
Fig 3.7 use of C.free……………………...……………………………………………30

Fig 4.1 Cgo code for testing……………………………………………………………..37
Fig 4.2 Cgo function life cycle………………………………………………..............38

List of Tables

Table 4.1 Performance analysis FFI……………………………………………………37

Table 4.2 Performance details Cgo function call……………………………………….37

Tables 4.3 Cgo function lifecycle…………………………………………………………38

List of Graphs

Graph 2.1 Moore Law analysis……………………………………………………………19
Graph 2.2 Comparison of code structure and concurrency in programming languages…………………………………………………………………………..22
Graph 2.3 Comparison of efficiency vs Human Readability in programming languages…………………………………………………………………………..24
Graph 2.4 Use of Golang in various domains over the course of past 3 years……………………………………………………………………………….26

Chapter 1. Introduction

1.1 Introduction

Modbus is a serial communication protocol which was developed by Modicon in order to be used by the logic controllers. It is used to transmit information over the serial lines connecting electronic devices. It is an open source protocol and is used by various manufacturers to implement into their own equipment.

Golang commonly known as Go is a Statically typed programming language which was developed by Google. It is very similar to C programming language in terms of syntax but is accompanied by various powerful tools that make it stand out over other programming languages. Memory safety, structural typing, garbage collection and CSP- Style concurrency are some of its specialties.

Cgo helps Golang packages to call on C code. Given a file implementing Go code, Cgo outputs both C and Go files which is a result of a combined Cgo package. Although Cgo is hard to implement and can have multiple complications, sometimes it is the only option the user has to take.

1.2 Problem Statement

Implementation of Modbus C Library using Golang:
Golang is a new programming language that is developed by GOOGLE. It is designed keeping in mind the requirements of ever growing technological needs. Modbus is a industrial grade protocol that is being used by all the devices. The task to accomplish is to implement this Modbus protocol into Golang.

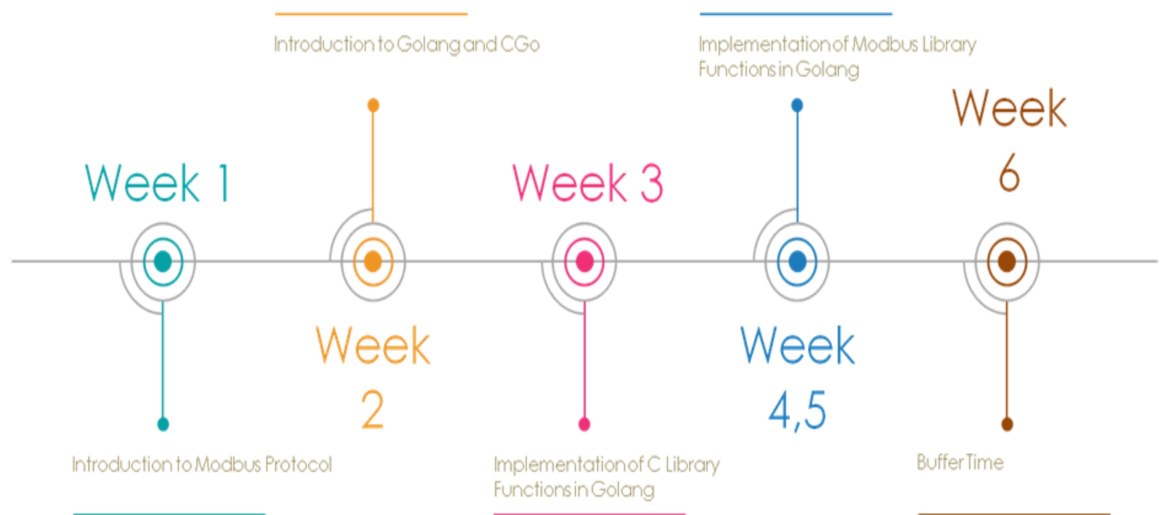
1.3 Objectives

The following objectives are set up during the course of working of this project

- Learning of C and Golang
- Detailed analysis of the TCP Modbus Protocol and C Modbus library
- Analysis of C go including its implementation taking into consideration the complications involved.
- Working with C Modbus Library functions in CGO

1.4 Methodology

Work timeline



The working can be further divided into following phases

Phase 1:

Initially one has to track back to learning of C language and its basics. Then one needs a detailed knowledge of the Modbus Protocol. The types of Modbus and how the protocol sends packets from one device to another. The devices that are involved and how they communicate amongst each other. What are the packet designs and the error checking methods that are implemented?

Phase 2

Golang. What is new about the language and its syntax. What are the advantages and disadvantages of the language over pre existing static languages. What is Cgo? And how its is implemented? Testing of various functions, data types, pointers and function calls passed from C to Golang

Phase 3

Looking out for complications and errors involved in implementation of CGo. Knowing the need to implement Cgo and not directly using Golang. Working on methods that could counter the same. Keeping in mind the above complications implementations of Modbus C library functions into Golang.

1.5 Organization

The report is divide into 5 chapters. **Chapter 1** introduces us to the project. The timeline on how the work has been done and the methods applied. **Chapter 2** is a detailed Literary Survey of the Modbus and Golang language. **Chapter 3** analysis how the system development took place and what were the problems faced and how they were resolved. **Chapter 4** includes the performance analysis of Golang when incorporated with C. **Chapter 5** includes the future scope and applications of the project. Finally in the end **References** and the codes applied are included in the **Appendices**.

2. Literature Survey

2.1 Modbus Protocol

Modicon programmable controllers are able to contact each other and other devices too over a variety of networks. These networks include Modicon Modbus, Modbus Plus Industrial Networks, MAP and Ethernet. These networks are accessed using built in ports that are available in the controllers or using network adapters and gateways that Modicon itself provides.

The common language that is used for communication between all the Modicon controller is called Modbus protocol. This protocol is a definition of the message structure that the controller understands and uses, no matter what type of network is used for communication. This protocol includes information on the process how the controller is able to access another device, process on how it will respond to calls from other This convention is an inner standard utilized by the Modicon controllers utilized for parsing the messages.

Amid the correspondence between the gadgets the convention characterizes on how the gadgets can perceive the gadget addresses, message connected to it, the activity that should be taken and is additionally ready to get helpful data from it. In the event that the solicitation requests an answer, an answer message is built by the controller and sent utilizing the Modbus convention. When working on different networks , message that contains Modbus Protocol are implemented into a frame or packet like structure to be used in the network.

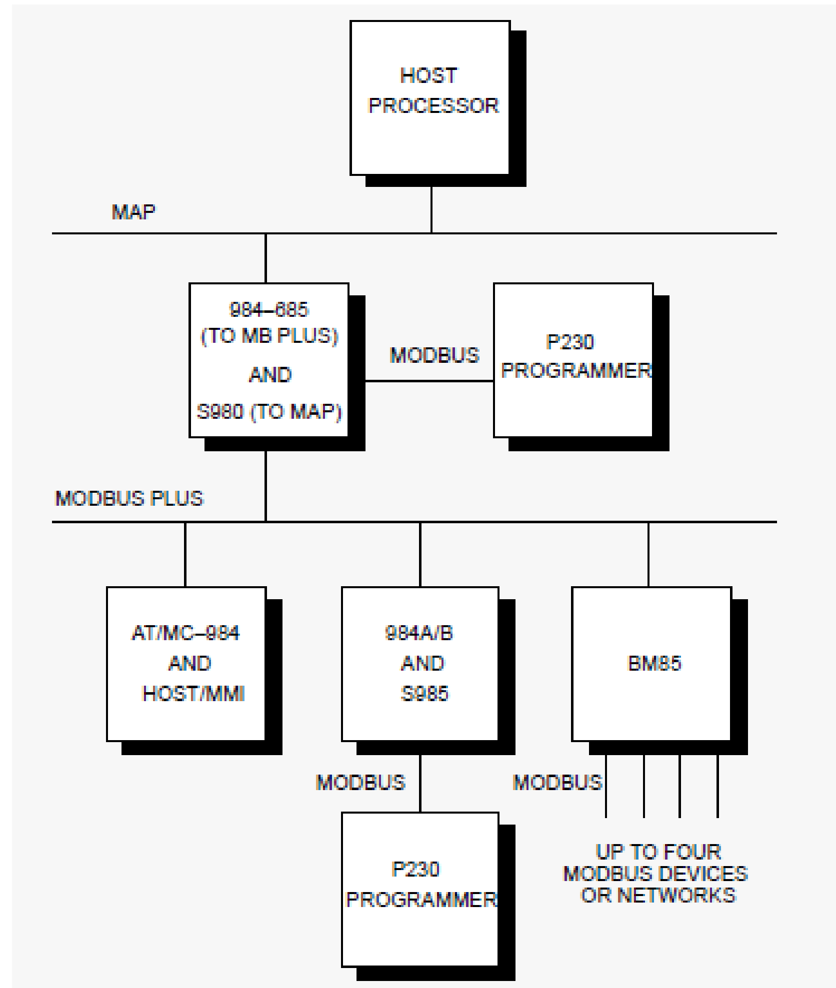


Fig 2.1 Modbus Protocol Application

Fig1 shows how devices are connected in a hierarchy of networks that implement widely different techniques. The Modbus Protocol provides a common language for all these devices.

2.1.1 Transactions on Modbus Networks

Standard Modbus ports on Modicon controllers use an RS–232C compatible serial interface that defines connector pinouts, cabling, signal levels, transmission baud rates, and parity checking.

The controllers are connected using a master slave configuration where only one device has the ability to take control over the transactions (MASTER). The other devices (SLAVES) take in the command from the Master and respond accordingly. Master devices contains programming panels and host processes while slaves only include programmable controllers.

The master can contact any slave or can initiate a broadcast message to all the slaves at one instant. Slaves only respond to messages that are addressed to them individually and not to the ones that are broadcasted to them. The slaves response is also built in using the Modbus Protocol. This master slave principle is also applied at the message level.

2.1.2 Query-Response Cycle

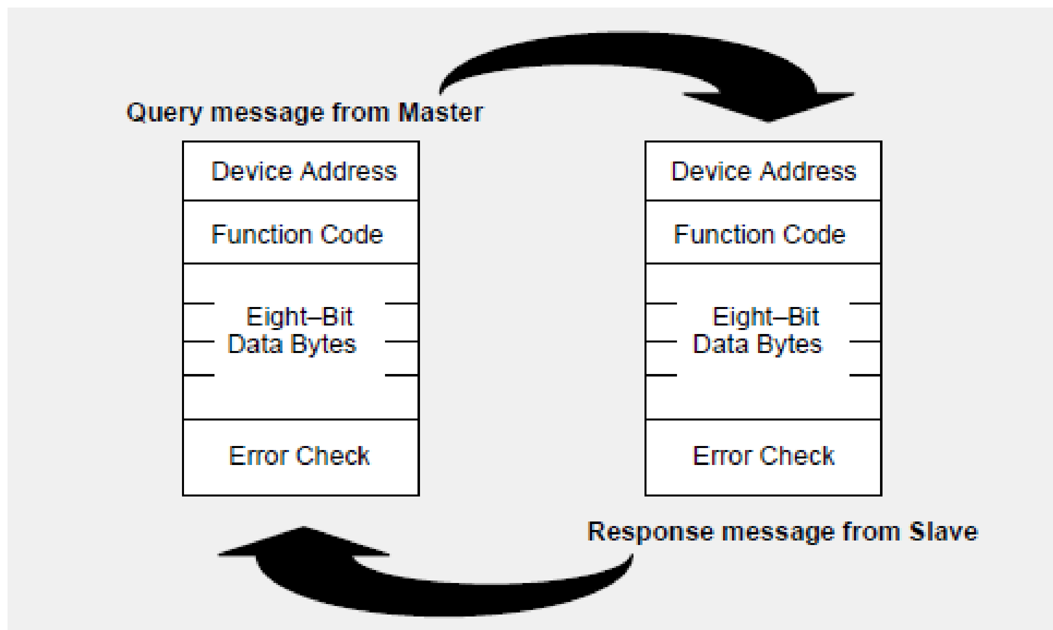


Fig 2.2 Master-Slave Query-Response Cycle

The Query: The code in the query tells the slave that has to be addressed about the action that has to be done. The data will also include the information that the slave will be needing in order to complete the request. **Function Code** will help the slave in reading the local registers. Error check field helps in validating the integrity of the message.

2.1.3 Serial Transmission Modes

Controllers can be used to communicate on standard Modbus networks using the following protocols:

- ASCII
- RTI

Note: The serial parameter and modes need to be same for all devices that are on a Modbus Network.

2.1.3.1 ASCII Mode (American Standard Code for Information Interchange)

In ASCII mode every 8bit in a message is sent in the form of two ASCII characters. This mode helps in providing the time interval of up to 1 sec to in occur between characters without any errors.

2.1.3.2 RTU Mode(Remote Terminal Unit)

In case where the controllers are connected using RTU mode, every 8 bit contains two 4-bit hexadecimal characters. This mode is more advantageous due to the availability of greater character density which allows better data throughput when compared to ASCII (comparing in same baud rate).

Format

2.1.4 Modbus Message Framing

In any of the serial transmission modes mentioned in 2.1.3 a Modbus message is sent by the device into frame, and it already knows the beginning and the end. This helps the receiver to begin at starting of the message, whether the message has been completed or not and the device or all the devices (in case of broadcast message) the message has been sent to. It can also determine whether any partial message has been sent or whether any error has been incurred.

ASCII FRAME

In case of ASCII framing the message always starts with a 'colon' character and ends with a 'carriage return-line feed (CRLF) pair.

The characters that are transferred are hexadecimal (0-9,A-F). The devices look for the colon character in order to confirm. After one message is received the devices decodes the next field. This is done to determine the addressed device. The interval between each characters is one second and if in any case more than one second is taken an error is generated.

START	ADDRESS	FUNCTION	DATA	LRC CHECK	END
1 CHAR :	2 CHARS	2 CHARS	*n* CHARS	2 CHARS	2 CHARS CRLF

Fig 2.3 ASCII Message frame

RTU Framing

Messages in RTU framing begins with an interval of at least 3.5 character times. Device address is the first field that is being transmitted. RTU framing allows hexadecimal characters for transmission (0-9, A-F). Networked devices continuously keeps on looking for the message even during the 'silent intervals'. When the devices receives their first field, each of the device decodes it and check whether it the desired addressed device.

The silent interval also marks the end of the message because after this 3.5 character time a new message can be transmitted.

It is to be taken care that the entire message is transmitted in a continuous stream because if the interval exceeds 3.5v character time the receiving device will think that the message is incomplete and flush the entire incomplete message. Then the next received message will be assumed to be address field of a new message.

Similarly if a new message is transmitted earlier than 3.5 character time the receiving device will not be able to differentiate between new and the old message and consider the new message as a part of old one.

START	ADDRESS	FUNCTION	DATA	CRC CHECK	END
T1–T2–T3–T4	8 BITS	8 BITS	*n* x 8 BITS	16 BITS	T1–T2–T3–T4

Fig2.4 RTU message Frame

2.1.5 Contents of the data field

Data field consists of a set of two hexadecimal digits (ranging 00 to FF).As per requirement of the serial transmission mode they can either be constructed from a pair of ASII characters or from one single RTU character depending upon serial transmission mode of the network.

The messages that are sent by the master to the slave also consists of information or instructions that a salve must abide by. This additional information includes register addresses, the number of items that need to be handled, and the number of actual data bytes in the field.

If no error occurs during the response data field from the slave will consist of the data requested by the master else it will include the exception code that the master application will use to determine its next step.

In some cases data field can be of zero length (nonexistent).

2.1.6 Contents of the Error Checking Field

The contents of error checking field are again dependent upon the method tha has been applied.

ASCII

In case of ASCII mode the error checking field consists of two ASCII characters. These characters are a result of LRC (Longitudinal Redundancy Check) calculation which is done on the message contents excluding CRLF characters and colon at the beginning of the message.

RTU

In case of RTU mode the error checking field consists of 16-bit value. These 16-bit values are implemented as two 8-bit bytes. These bits are a result of a CYC (cyclic Redundancy Check) calculation which is done on message contents.

The CRC field is attached to the end of the message. After this is done the low-order byte of the field gets traversed first and then the high-order byte. The CRC high order byte is the last byte that is being sent.

2.1.7 Serial Transmission of Characters

Messages transmitted using the Modbus serial networks are sent in the following manner (left to right)

LEAST SIGNIFICANT BIT (LSB). . . . . MOST SIGNIFICANT BIT (MSB)

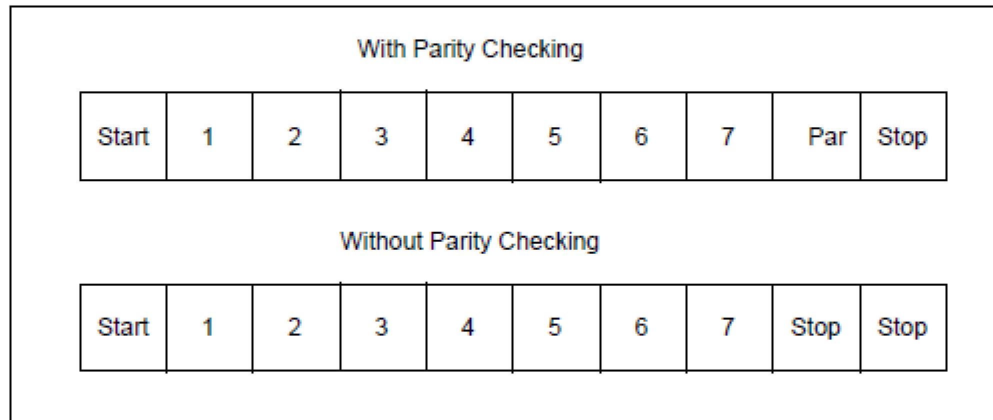


Fig2.5 Bit order (ASCII)

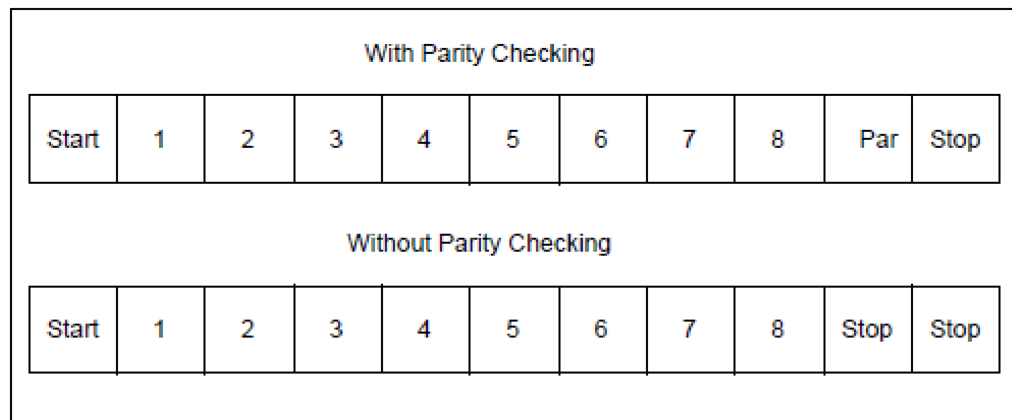


Fig2.6 Bit order RTU

2.1.8 Error Checking Methods

Two kinds of error checking is used by Standard Modbus serial networks

1. Parity Checking (even or odd)
1. Frame Checking (LRC or CRC)

Both of these methods are implemented in the master device where the device implements these methods before transmitting the methods to the slaves. When the slave receives the messages it checks for every character received by it and also the entire frame.

The master is programmed in such a manner that it waits for a significant amount of time before aborting a transaction. This time will be set taking into consideration the average response time of a slave.

For example if a master sends a message and the slaves detect that it consists of an error the slave will not respond to the message. Thus the master will keep on waiting and eventually the transaction will lead to a timeout and the master will then handle the error.

If in any case the master has addressed a message to an nonexistent slave there will again be no response thus leading to a timeout.

In case of networks like MAP or Modbus Plus frame checking occurs at a level above the Modbus. In such cases error checking methods such as LRC and CRC do not validate.

2.1.8.1 Parity Checking

Users have the authorization to configure controllers for even or odd checking or in some cases no parity checking. This helps in deciding the parity bit of each character.

Whatever the parity gets decided (even or odd) the number of occurrence of 1 bits is counted and then the parity bit will be set accordingly as per selection.

For example

If the following data bits are present in a RTU character frame (1111 0010) the number of 1s present is 5. In case of even checking parity bit will be assigned 1 to make the count to even (6) and will be set to 0 in case of odd checking to limit the count to odd.

When the message is received the receiver again checks the count of 1 present and will impose an error if the count does not match the pre set settings.

Drawback

Parity check can only work if odd number of bits is missing from the message. For example in odd parity check message consisting of 5 1s drops two 1s during the transaction the count of 1s will still remain even thus leading to false message transmission without the detection of an error.

When nothing is specified the master does not transmit any parity bit neither any checking is done.

2.1.8.2 LRC Checking

ASCII mode messaging consists of an error-checking field implementing the Longitudinal Redundancy Check. In this case the LRC field will check at the beginning and at the end for colon and the CRLF pair respectively. It is implemented even if parity check method is implemented or not.

The length of a LRC field is ! Byte, consisting of 8-bit binary value. This LRC value is computed by the master which attaches this LRC bit to the message. The receiver or the slave also has to compute the received LRC and then compare its value with the original value to check whether there is an error or not.

LRC is the sum of all the successive 8-bits not including any carriers and then applying two's compliment to the result. This procedure is implemented on the ASCII message.

The colon character at the beginning of the message and the ending CRLF pair is not included in the calculations.

2.1.8.3 CRC checking

A Cyclic Redundancy Check based field is included in the RTU mode messages which checks the entire message. t is implemented even if parity check method is implemented or not.

The calculations regarding the CRC values are performed by the master which then attaches the CRC to the message before transmission. The slaves or the receiver on receiving the message again computes the value and checks whether it matches with the original value and then produces the error accordingly.

The procedure starts with loading first 16-bit register to all 1s. Then successive 8-bit bytes of the message are applied to the current contents of the register. These 8 bits are the ones used to generate the calculations for the LRC. The bits at the starting and at the end, the parity bits are not included in the CRC calculations.

During the implementations of the calculations every 8-bit is OR ed with the register contents. The generated outcome is added to the Least Significant Bit. The most significant bit is is filled with 0 bit. The receiver receives the message and takes out the LSB. If LSB comes out to be 1, the the register has to perform exclusive OR with the current fixed value. Otherwise no such calculation needs to be done.

The above process continues until all the eight shifts have been completed. After the process gets completed the next *-bit is exclusive Or with the current value of the register. The final bits that are generated after all the bytes of the message are is the CRC value we are looking for.

2.2 Modbus TCP/IP

Modbus TCP/IP is a Modbus RTU protocol that has a TCP interface running on the Ethernet. The modbus message structure is the application protocol which is a standard for organizing the data not depending upon the medium used to transmit the data but the TCP/IP provides the transmission medium for the messaging.

In simple words, it allows transfers of binary data amongst the computers. It is a standard that is also used in the world wide web. The function of the TCP is to check whether all the data packets are received by the receiver and ensure that are received in correct order. IP ensures that the packets are received by the correct user and correctly routed.

Note; The TCP/IS is is only meant for transportation of the packets and is not responsible for the kind of data being transported, how the receiver will be interpreting the received data. The above mentioned issues are handled by the application protocol which is Modbus in this case.

Modbus TCP/IP uses both TCP/IP and Ethernet to take the Modbus messages from one device to another i.e. Modbus is combining a Physical Network using a network protocol.

Modbus TCP encapsulates a simple Modbus data frame into a TCP frame, not including the Modbus checksum.

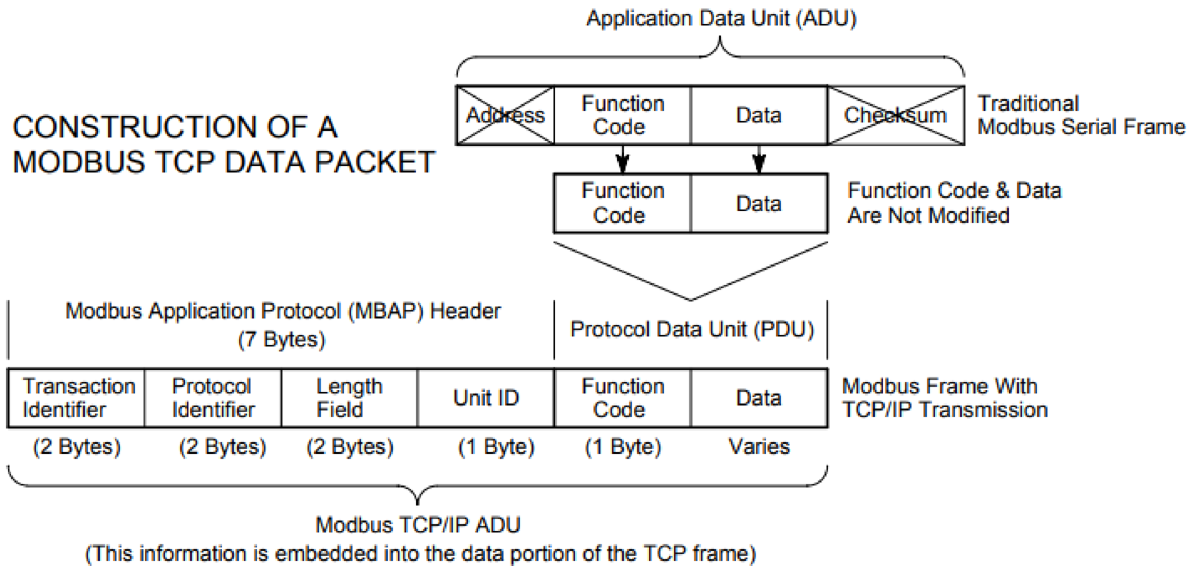


Fig 2.7 Modbus TCP data Packet

The Modbus commands along with the user data are embedded into the data container of the TCP protocol keeping in check no data should not be altered. Standard Ethernet TCP/IP link layer is implemented to check the errors instead of Modbus error Checking field (check sum).

The data of the Modbus TCP/IP is integrated into a data field of a TCP frame and is transferred using a TCP protocol 502. This TCP port is specially reserved for Modbus Applications. All the Modbus masters and slave communicate using the port 502.
Like Ethernet, Modbus is free of cost and can be accessed by anyone and is widely accepted and supported by many manufactures and implemented in industrial equipments. It is easy to implement and is a prime contender to become the industrial standard for communication.

2.2.1 Determinism

Determinism is defined as the potential of a Communication Protocol to assure that the message sent by the sender is received by the receiver in a finite amount of time. This is very important for critical control applications.

Earlier Ethernet was not considered as a reliable communication medium because of the following issues it had

- Non deterministic Approach
- Low Durability

Ethernet equipment was not designed keeping in mind the harsh industrial requirements. It was meant to be used in office environments. But with the passage of time new industry level Ethernets were made available that could also connect to connecters, shielded cables and switches.

The non-deterministic behavior of the Ethernet is due to the arbitration protocol used by the Ethernet for the transmission of data. Arbitration protocol is basically CSMA/CD (Carrier Sense Multiple Access with collision detection).

In CSMA/CD when the device will send data frame at any time the device will check whether the line is idle and available for data transmission. If the sender finds an available line it will then proceed to send the data. This method is considered as non-deterministic because data can only be transferred when a line is available for transmission. Hence the transmission is now dependent on the line and cannot be considered as deterministic. The waiting for free lines can result in unpredictable waiting. With this approach in the use, data transfer can also face collisions.

Since most control systems cannot rely on such delays and unpredictability because the avg. packet time transmission is less than 100ms. This is the reason why it is not considered to be implied on critical control systems.

Ethernet can be made more deterministic by the use of Ethernet switches that are faster to establish a connection. These switches have increased bandwidth that help in breaking of networks into several smaller networks in order to avoid collision.

2.3 Introduction to Golang

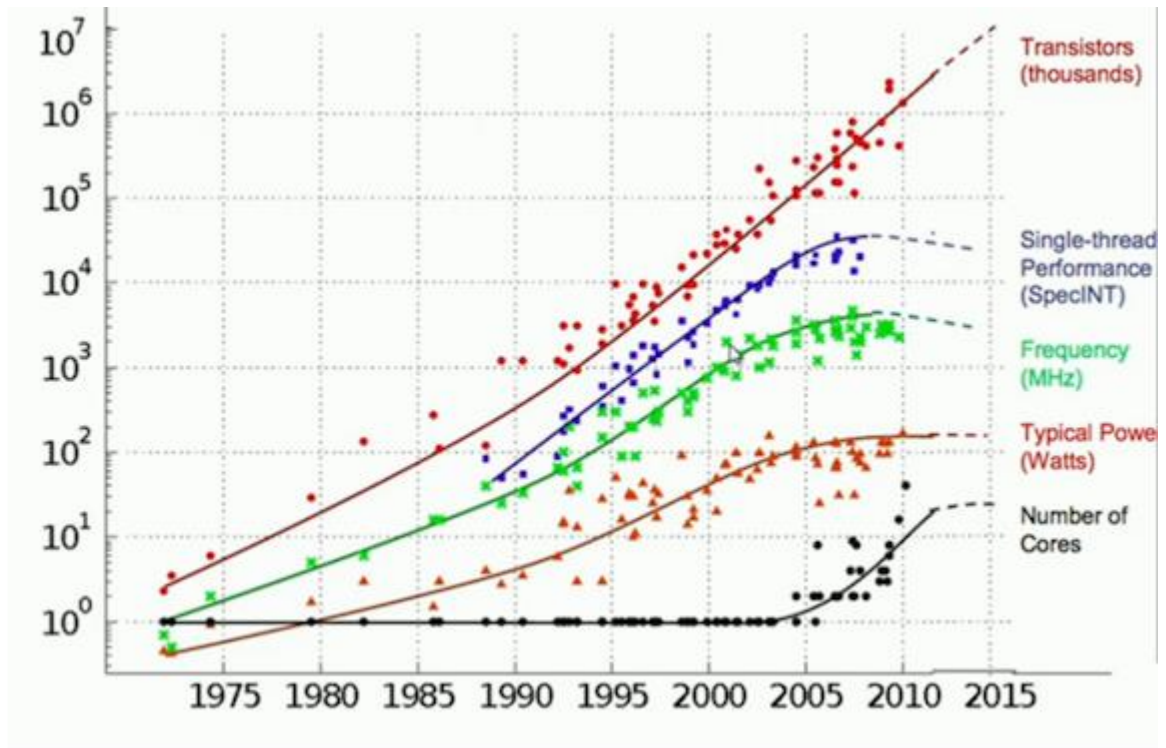
Fig 2.8 Golang logo

Golang is an procedural programming language developed in the year 2007 by Robert Greisemer, Rob Pike and Ken Thompson at Google. It was launched as a open source programming language in the year 2009. Go syntax is similar to C but it uses the best features of all the languages keeping the mind needs of fast changing technological requirements.

2.3.1 Why use Golang?

Hardware Limitations

The first Pentium processor had a clock speed of 3.0 GHZ and was introduced in the year 2004. The Macbook pro introduced by the apple in the year had a clock speed of 2.9 GhZ which in comparison is not much when compared to the time that has elapsed.



Graph 2.1 Moore law analysis (1975-15)

Moore Law has been slowing down with the passage of time. For almost a decade single-thread performances and the frequency of the processors have come to stagnation. This cannot be tackled by increasing the number of transistors as with the decrease in size quantum properties comes into play. In order to tackle the situation the following steps were taken

- Addition of more cores to the processors
- Hyper Threading'
- Addition of more cache memory

The following improvements have their own drawbacks. Introduction of more and more cache has physical limitations as more cache means slower processing, Adding more and more cores has a limit too plus its adds insanely to the cost.

This implies that one cannot rely too much on the hardware improvements and one has to switch on to efficient software to keep the balance maintained, Unfortunately the available programming languages do not have the capability to do the same.

Go has goroutines !!

As discussed more and more cores are added to the processors for faster performance and the same can be expected in the near future. Modern day applications uses multiple micro-services therefore the language that is used to develop the software needs to support concurrency and should also match with the increasing number of cores.

The languages that we use today incl. Java and Python were created in the year when only single threaded environment were available. These languages support multithreading but they are not able to deal with the following

- Thread-locking
- Race conditions
- Deadlocks
- Concurrent Execution

Golang was introduced in the year 2009. By that time multi-core processors were already introduced and hence was build with concurrency in mind. Golang has goroutines instead of threads that consume merely 2kb of memory. Since memory consumptions is so low as compared to other languages (JAVA thread consumes nearly 1MB of memory from the heap) millions of goroutines can be implemented at an instant.

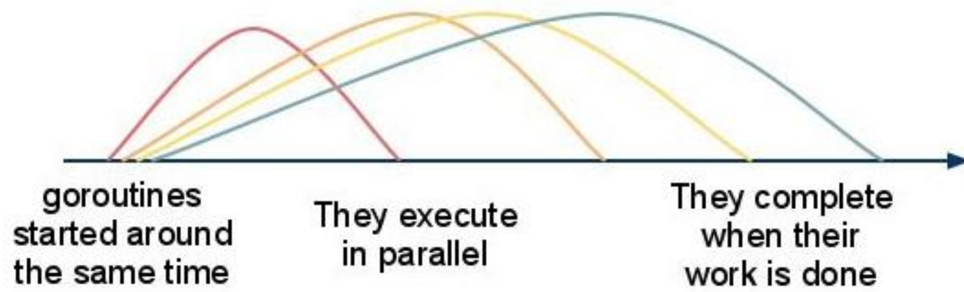
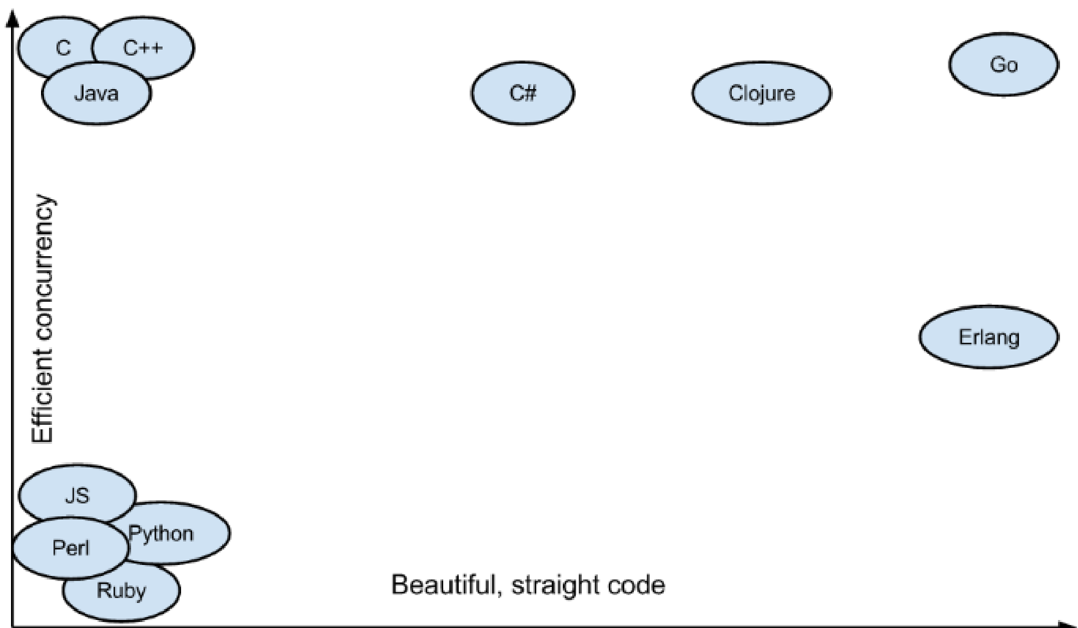


Fig2.9 Working of Goroutines

Goroutines has other benefits too

- Goroutines use memory only when it is needed. Goroutines consists of growable segmented stacks.
- Goroutines are much faster than threads.
- In built primitives for communication
- Helps in avoiding mutex locking



Graph 2.2 Comparison of code structure and concurrency in programming languages

Go runs directly on the hardware

Languages like C/C++ have better performance as compared to JAVA and Python because they are compiled and not interpreted.

Processors only understand binaries. In case of JAVA or any JVM based programming language the compiled code is converted into byte-code which the JVM understands and that the virtual machine is able to run on top of the Operating system. The virtual machine interprets the bytecodes and converts them into binaries. This process happens during the execution.

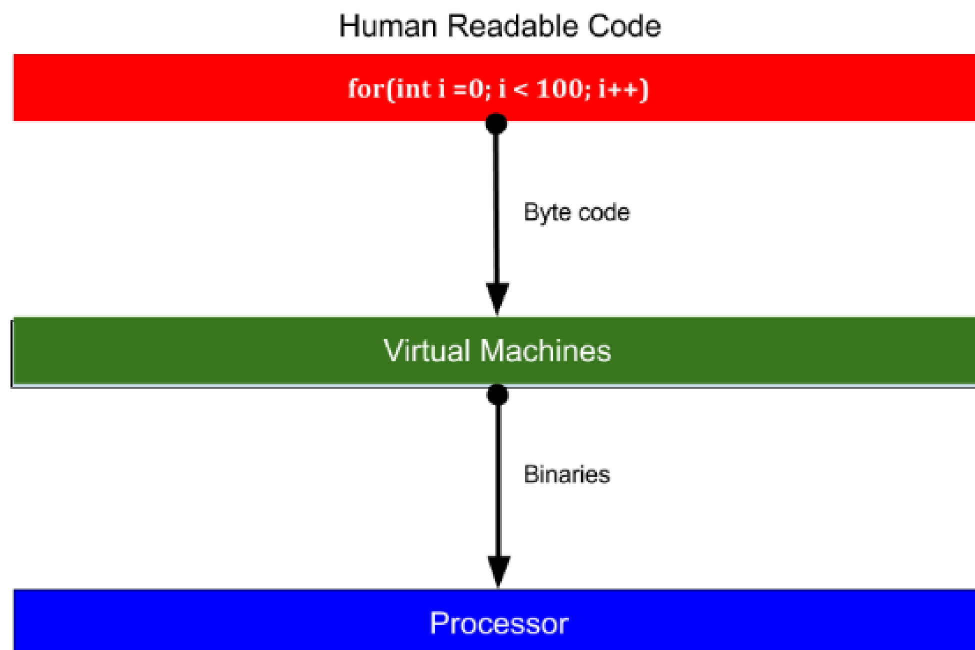


Fig 2.10 Compilation sequence in Java and Python

In case of C/C++ Virtual Machines are not required which removes one step from the execution when compared to languages like JAVA and Python.

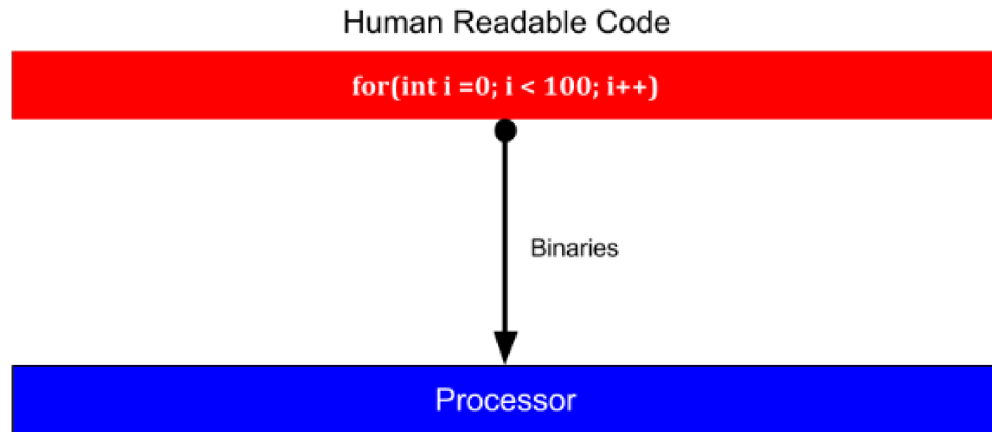
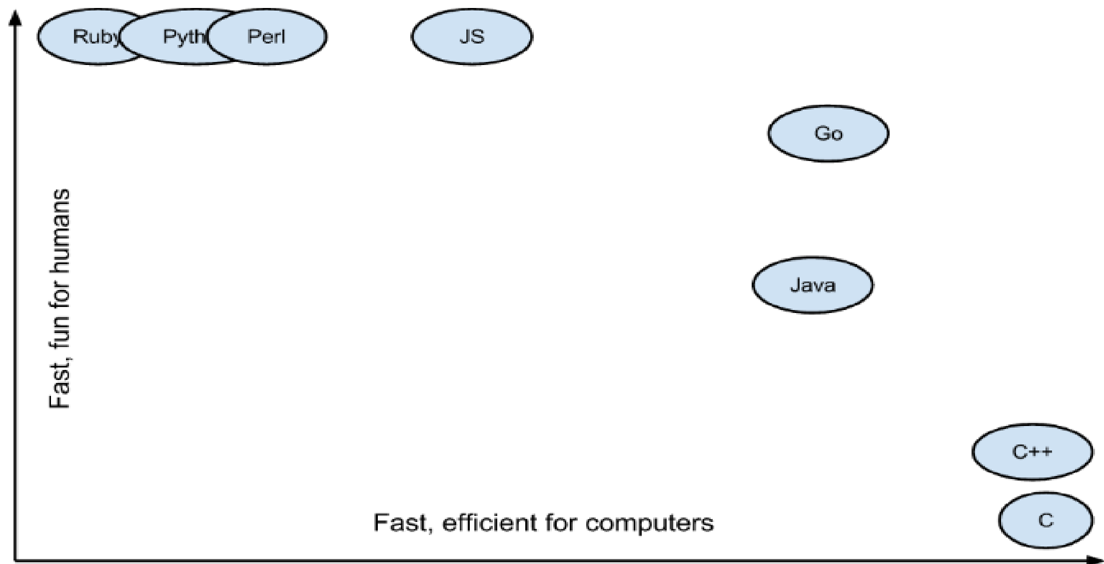


Fig 2.11 Compilation sequence in C/C++

Go Code is easy to maintain

Go code has been constructed keeping in mind that one section of the code does not affect the other drastically. Go makes the following changes in the code structure in comparison to other programming language.

- Go code is divided into packages and no classes exist
- Go does not support inheritance. In languages supporting inheritance if changes are made to the base class, changes are required at the inheriting classes.
- Go does not support constructors
- No annotations
- No generics
- No exceptions



Graph 2.3 Comparison of efficiency vs Human Readability in programming languages

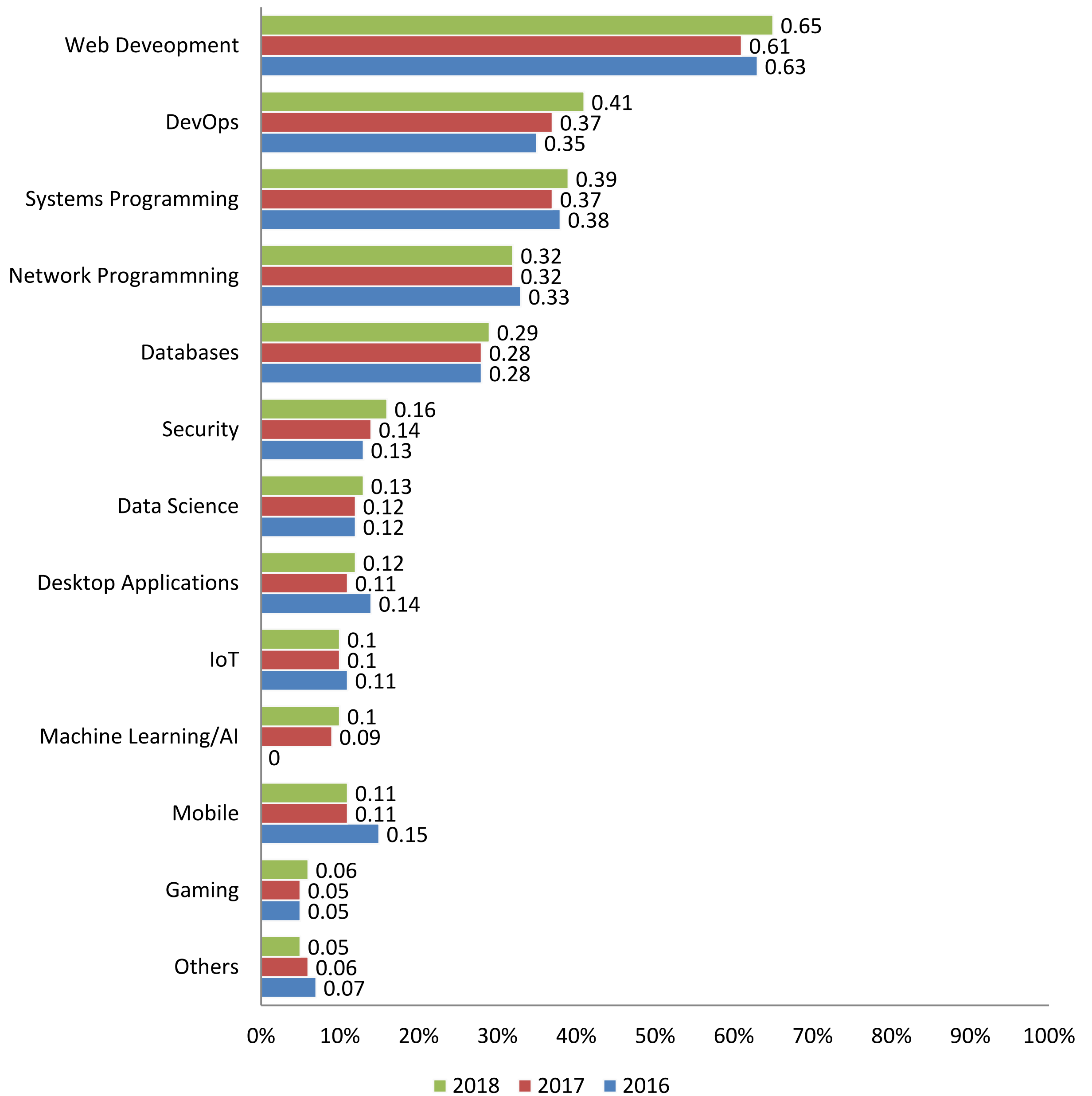
Golang is a cross-Platform

Golang has a Garbage Collector

It is kind of a automatic memory management system that helps in making concurrency more efficient.

Golang is already used by many Tech Giants

Golang is already in use by the following big companies Google, Youtube, Apple, Dropbox, Docker, BBC, The Economist, IBM, Twitter and Facebook.



Graph 2.4 Use of golang in various domains over the course of past 3 years

3. System Development

Writing of the modbus protocol completely from the scratch can be a tedious process. It can result in enormous use of time and resources. Luckily golang gives the uses to implement C in golang also called as Cgo. Cgo lets the Golang packages implement C code.

Suppose we are given a Golang source file that contains implemented C functions, Cgo will convert it into single package.

```
package rand

/*
#include <stdlib.h>
*/
import "C"

func Random() int {
    return int(C.random())
}

func Seed(i int) {
    C.srandom(C.uint(i))
}
```

Fig 3.1 Sample Cgo code

In the above example we see rand package imports random() function. But no such package exists in the golang library. This is because we have implemented C in the go language.

3.1 Closer look at the implementation

```
/*
#include <stdlib.h>
*/
import "C"
```

Fig 3.2 Cgo header files

Any comments that have been made before import “C” are recognized by cgo and acts as directives for the file. The remaining lines will act as a directive for the file during compilation.

In case this the comments are only #include <stdlib.h> but it can be anything from a complete C code to C functions and their definitions.

The random function in C library returns a long type which is accessed by golang using **C.long** but it cannot be accessed directly by golang as it is a C type. Thus golang converts it into Go types.

```
func Random() int {
    return int(C.random())
}
```

Fig 3.3 Cgo function call

Or another way to show this conversion is as follows

```
func Random() int {
    var r C.long = C.random()
    return int(r)
}
```

Fig 3.4 Cgo function call alternate method

The seed function works opposite to that of random function. It takes a golang integer converts it into a C unsigned integer and then passes it to the functio.

```
func Seed(i int) {
    C.srandom(C.uint(i))
}
```

Fig 3.5 Go to C

Using C Strings in golang

C language does not have explicit string types rather they are terminated by a zero at the end.

Thus strings in order to be used in golang needs to be converted. In order to complete these conversions the following functions are use C.Cstring, C.GoString and C.GoStringN.

```
package print

// #include <stdio.h>
// #include <stdlib.h>
import "C"
import "unsafe"

func Print(s string) {
    cs := C.CString(s)
    C.fputs(cs, (*C.FILE)(C.stdout))
    C.free(unsafe.Pointer(cs))
}
```

Fig 3.6 String methods in Cgo

Golang has its own garbage collector thus any memory allocations made by C is unknown to Golang. Therefore care must be taken care whenever **C.Cstring** is used. Because the user should always deallocate the memory using **C.free.** The function returns a pointer at the starting of the array therefore it is converted into an unsafe pointer.

```
func Print(s string) {
    cs := C.CString(s)
    defer C.free(unsafe.Pointer(cs))
    C.fputs(cs, (*C.FILE)(C.stdout))
}
```

Fig 3.7 use of C.free

3.2 Issues with implementing Cgo

Slower build times

Whenever C is implied in golang the compilation process becomes more complicated. The following things occour during the compilation

- Cgo tools is called for the process of compilation.
- System C compiler is invoked every time it sees a C code in the code.
- The compilation units merge to form a .o file.

The following steps are gone through every time a Cgo file is made to run. Usually the go tool is able to parallelize some of the work but the compile time increases with due to the involvement of C libraries.

Complicated Builds

When golang was released one of the main objectives of the developers was to give it a self describing build process that is that is source of the program should be sufficient to build the project.

When only using Golang we need golang tools in order to compile the project but with the introduction of Cgo we need a separate C compiler too and also the project will now include all the C libraries.

Cross Compiling is cumbersome

Golang has the best cross compilation support as compared to any other language. Initially Cgo is not enabled when cross compiling takes place. If your code is written in Go only then the system has no issue in compiling the code. The problem arises when C gets involved. In that case either one has to give up the idea of cross compilation. On the other hand one can also look out for cross compilation C toolchains to complete the task.

For example we are building a product that contains client and server operating over a TCP protocol and it has to be run on a SaaS model. In this case cross compilation will not play any major role in the compilation. But in case the same product has to be integrated into every client device then it would be a difficult task for them to cross compile.

Whenever you have implemented C code in Golang you will have to deal with all the cross compilation issues mentioned above and also need to take care about the fact that the implemented C code works fine on the new platforms that are now supported by go. This also includes to the limited debugging options that will be left.

Tools goes missing

Golang contains some amazing tools to work on

- Race Detector tool
- pprof for code profiling
- fuzz testing
- source analysis tool

But none of these tools work when Cgo is implemented.

In simple words Cgo is not the combination of two languages rather it is an intersection. This is a combo of memory safety in C and the debugging ability of golang

Performance issues

C code and Golang code are two completely different things and should not be mixed up. C golang is acts as a transverse boundary between the both. The working of the code depends upon where the C code has been placed.

The C code is not able to understand Golang calling conversion. So whenever a code gets called the details must also be stored in a go routine stack, then it has to switch back to C stack again. But now the C code has no idea about the way it was being invoked.

If while compilation of the code some issue in occurs, golang tries to recover it and at least print the reason why the error is being caused.

C code calls the shots

Go code and C code needs to set the rules on how the distribution of resources will take place, how will the address space gets shared, who will be handling the signals and the threads. These rules are actually Go's assumption to how the C code is working. On the other hand the C code also devise that it is always running on one single thread or it is not able work in a multi thread environment at all.

We need to note that the code we are writing is not taking its logic from the C library but Golang has code has to exist with the C code because Golang itself has no substitute to that code and is forced to use functions from the C library.

Complicated Deployment

Single, static binary these keywords play a major role when we talk about C go deployment. WE can build build go projects into deb or a rpm but it completely depends upon the environment we are working on and also keeping in mind that all other dependencies are included in the project. We can include these dependencies using install dependencies and then throw away the problem to OS system package manager.

Golang gives us the power to compile our entire code statically but when C go is being involved issues are created during build times and deployment life cycle.

3.3 Why use Go despite the issues?

As seen in the above mentioned section use of Golang can be the worst deal a programmer has to make. But at times C go is the only option one has to take

For example golang has no equivalent of Modbus library. Writing the code for golang from scratch will increase the cost many fold not the mention the time and resources included.

Following are the some instances where Cgo is the only route:

- No equivalent library available in Golan to resolve the issue
- Codes that need a frontend to be developed
- Requirement of SDK and other libraries
- The logic is already implemented in C and can be risky to implement in any other languag

3.4 Dealing with the issues

Know the C

Whenever one is working with C go one cannot ignore the importance of C in the task. If one is not proficient in writing C codes then it should be prioritized before switching on to golang. If it is a library or a header file that needs to be implemented on golang the functions implemented in the library needs to be clear.

Use the MakeFile option

During the compilation of the program it is not necessary to use the *make* command. One can also use commands such as *cmake* or *bazel.* Commands such as *go generate* or *go build* can come in handy but the project now also includes C code. Here we need to consider the compilation on C code as well hence the C code should be compiled separately or it should be buildable by golang.

Isolation of the code

The implementation of C go can be simplified by isolating the c code and wrapping it into a separate package or project. By isolating the C code we can build the C code and the go file separately. This will help us in retaining the much of the tools of both the languages and the bugs can be resolute easily in both the files separately.
Note: The overall project should compile as a single file and C and golang should be working hand in hand.

Use of Static C libraries

Implementation of static C libraries can not be always implemented. But one should always take the opportunity wherever possible. When working with build flags the situation can often be tricky as it is used to work with shared libraries.

C bridged Functions

The need for writing bridge functions come up now or then because of the following reasons

- Go does not have the capability to call macros functions. So in order to resolve this issue we need to write C functions that help us in calling the macros and then we implement those functions in golang. This a tricky affair and needs to be implemented cautiously when too many macros are to be used.

- It is not a good practice to call C function repeatedly. If possible and much necessary the user can call the repeated C functions using loops or the performance can also be improved by calling multiple C functions in a single go call.

- Golang does not allow typecasting which is usually a habit of most of the programmers. But while implementing C you can do typecasting by wraaping the C library.

Use of Test Wrappers

Since it is not possible to call Cgo directly from Golang test codes in order to test our C codes we need to call them in golang and then test their functionality. This can seem a little bit too much work but can prove fruitful during the debugging process.

Memory Management

C lets you control your memory allocation dynamically using malloc calloc and free functions. Some library functions also provide memory management tools that are implemented as soon the code is called. But when are using both C and golang the following practices need to be taken care in mind

- Make sure whenever memory allocations are required they are always done through golang and then passed onto C language. This practice cannot be used to call references to other Golang memory. This is useful when you are using gobytes in your code.
- *Defer* statement can be used to clean up the temp. memory used up by C in the program. (keep in mind how the how the C lifecycle operates)
- Avoid passing the C memory outside Golang function unless it is the only option left. In that case use structures in C to perform the task but still do not forget to deallocate the memory.

Concurrency

As mentioned in the previous section golang was designed keeping in mind the advantages of concurrency. Passing C structures into Go routines is not an easy task and can lead onto a lot of issues. The solution to this problem depends upon the library that has been implemented into your program. The library may have a thread local storage that is using C or it may be using resources from thread ID and the user cannot determine which thread Go routines are working on. In order to resolve this you can use the function runtime.*LockOSThread().* This may not be the best solution but can be very useful at times.

4. Performance Analysis

Consider a simple Golang code

```
1  package main
2
3  //int C_func(int a,int b) {
4  //    return a+b;
5  //}
6  import "C"
7
8  func main()
9  {
10     a,b:=C.int(10),C.int(10)
11     C:=c.C_func(a,b)
12     println(a,b,c)
13 }
```

Fig 4.1 CGo code for testing

Table 4.1 Performance analysis FFI

Function Call	Time Taken
C	2.214ns
Java FFI	9.03ns
Rust FFI	2.861ns
LuaJIT FFI	18.03ns
Node.js FFI	18.21ns
Cgo	76.00ns

FFI: Foreign Function Interface

Table 4.2 Performance details Cgo function call

Name	Old time/op	New time/op	Delta
CgoCall-4	63.2ns +- 2%	57.4ns+- 0%	-9.31%

The compiler goes through the following steps in order during the compilation of Go files that call in C functions

Table 4.3 Cgo function lifecycle

Language	Function caller/called	Process
Go	C.C_func	Written Golang Code
ASM	Runtime.[asm]cgocall	Call for convention tramp
C	C.C_func	Argument unpacking
	C_func	Original C funtion

The C part of the code is compiled by a separate gcc compiler and the golang part is compiled separately

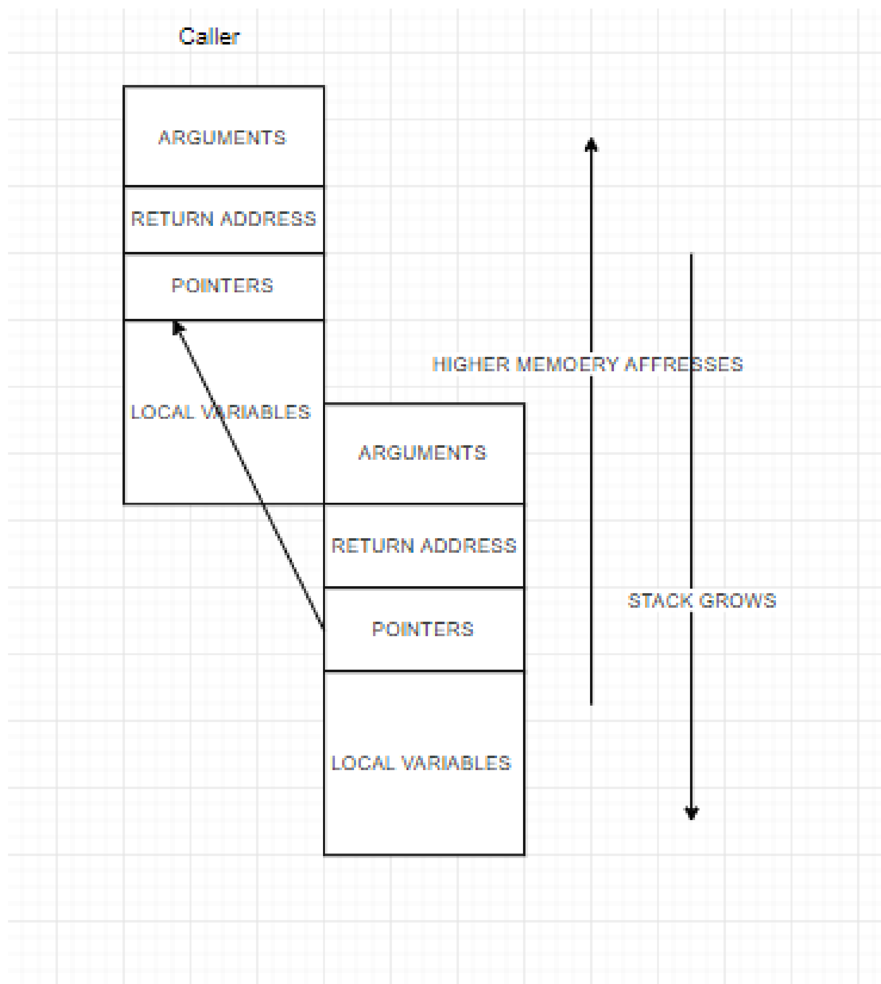


Fig 4.2 Cgo function life Cycle

5. Conclusions

5.1 Applications:

Master Slave applications that are able to communicate amongst smart devices and devices that are used to monitor field devices are the primary applications of the Modbus protocol. It can also be used in devices that implement wireless networking. Because of this feature it can also be implemented at oil and gas stations. The industry level standardization gives Modbus the ability to be used in infrastructure, energy apps and transportation. The common feature of the Modbus protocol remains to be the messaging and packet delivery.

Some other areas that can implement the Modbus protocol are:

- Heathcare systems

  Hospitals can use Modbus to keep track of the temperature of every room in a single interface making the job handy.

- Transportation

  Modbus can be used to detect change in traffic behavior by comparing it to standard traffic patterns

- Home Automation

  Home automation requires transfer of data from form one sensor to another and this can be performed easily with the help of Modbus protocol.

- Other industries

  Modbus can be used in any of those industries that require connection between devices and sending of data between the devices.

  Industries that have already been using the Modbus protocol includes the oil and petroleum industry, wind, solar and hydro electricity.

5.2 Future Scope

Modbus is a protocol that has been into the industry since the last century. But the question here to conquer is Will it survive the future? Is the use of Modbus declining or it will still remain poplar in the future?

The answer can be found in the properties of Modbus. It is used in every communicating device. It is east to implement and not much investment is required for the same.

Consider a boat that is full of expertise in data flow and communication. The mast for such a ship will always be Modbus protocol. Every time even a non expert who has no knowledge about system arch or the working of network switches can always turn on to Modbus protocol. Modbus will survive even when every communicating device will be boasting a power processor and a common Operating system.

No other protocol supports the amount of devices as much of it is covered by Modbus. So use of Modbus protocol means no compatibility issues and no worries regarding future updates. Working with some other protocols can lead to further investments in the form of cables and cards used for the connection. Modbus is very economical and uses simple standard Ethernet for connections.

References

- Modicon, Inc Industrial Automation Systems, “Modicon Modbus Protocol” ,2000, pg 2-31
- ACROMAG INCORPORATED, “BusWorks 900EN Series 10/100M Industrial Ethernet I/0 Modules”, 2005,Acromag INC USA, pg 2-15
- Golang.org , GOOGLE, 2012, Creative Commons Attribution 3.0
- Dave Cheney, “Cgo is not go”, The acme of foolishness, 2016
- Karl Matthias, O’Reilley media Dublin Ireland, “Cgo: When and (Usually) When Not to Use it”, 2017

APPENDICES

IMPLEMENTATION OF C LIBRARIES USING GOLANG

Code:

```
package main

      //#include<stdio.h>
      //void inC() {
      //    printf("I am in C code now!\n");
      //}
      import "C"
      import "fmt"

      func main() {
          fmt.Println("I am in Go code now!")
          C.inC()
      }
```

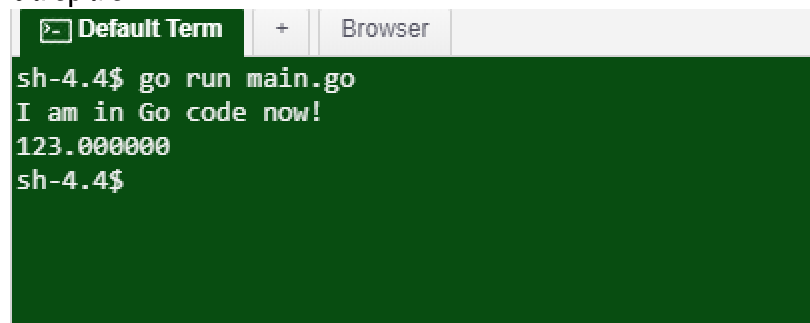
Code:

```
package main

//#include<stdio.h>
//#include<math.h>
//void inC() {
//    double a=123;
//    printf("%lf \n",a);
//}
import "C"
import "fmt"

func main() {
    fmt.Println("I am in Go code now!")
    C.inC()
}
```

Output



Code

```
package main
//#cgo CFLAGS: -g -Wall
//#include<stdlib.h>
//#include<string.h>
//#include<stdio.h>
import "C"
import "fmt"


func main() {
    str1 := C.CString("mayank")
    str2 := C.CString("mayank")
    x :=C.strcmp(str1,str2)
    if x != 0 {
        fmt.Println("not an integer")
    } else {
        fmt.Println("integer")
    }
}
```

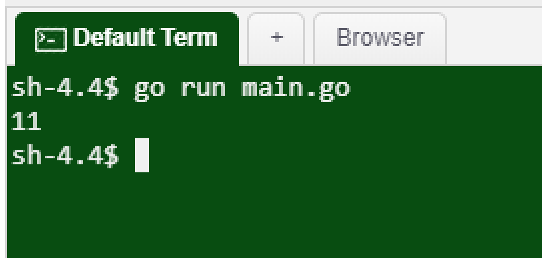
Default Term | + | Browser

```
sh-4.4$ go run main.go
integer
sh-4.4$
```

Code:

```
package main
//#include<stdio.h>
//int inC(int a,int b){
    //return (a+b);
//}
import "C"
import "fmt"

func main() {
   x:=C.int(5)
   y:=C.int(6)
   res:=C.inC(x,y)
   fmt.Println(res)
}
```



Code:

```
package main

  /*
  #include<stdio.h>
  #include<stdlib.h>
  struct Gretee {
    const char *name;
    int year;
  };

  int greet(struct Gretee *g,char *out) {
    int n;
    n=sprintf(out,"Greetings, %s from %d! \n",g->name,g->year);
    return n;
  }
  */
      import "C"
      import "fmt"


      func main() {
          name := C.CString("MAYANK")
       defer C.free(unsafe.Pointer(name))

       year := C.struct_Gretee{
         name: name,
         year: year,
       }
     ptr:=C.malloc(C.sizeof_char * 1024)
     defer C.free(unsafe.Pointer(ptr))

     size :=C.greet(&g,(*C.char)(ptr))

     b:=C.GoBytes(ptr,size)
     fmt.Println(string(b))
      }
```

Default Term | + | Browser

```
sh-4.4$ go run main.go
# command-line-arguments
./main.go:23: undefined: unsafe in unsafe.Pointer
./main.go:27: undefined: year
./main.go:30: undefined: unsafe in unsafe.Pointer
./main.go:32: undefined: g
sh-4.4$
```

```
package main

import "net"
import "fmt"
import "bufio"
import "strings" // only needed below for sample processing

func main() {

  fmt.Println("Launching server...")

  // listen on all interfaces
  ln, _ := net.Listen("tcp", ":8081")

  // accept connection on port
  conn, _ := ln.Accept()

  // run loop forever (or until ctrl-c)
  for {
    // will listen for message to process ending in newline (\n)
    message, _ := bufio.NewReader(conn).ReadString('\n')
    // output message received
    fmt.Print("Message Received:", string(message))
    // sample process for string received
    newmessage := strings.ToUpper(message)
    // send new string back to client
    conn.Write([]byte(newmessage + "\n"))
  }

Client.go
package main
```

```
import "net"
import "fmt"
import "bufio"
import "os"

func main() {

  // connect to this socket
  conn, _ := net.Dial("tcp", "127.0.0.1:8081")
  for {
    // read in input from stdin
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Text to send: ")
    text, _ := reader.ReadString('\n')
    // send to socket
    fmt.Fprintf(conn, text + "\n")
    // listen for reply
    message, _ := bufio.NewReader(conn).ReadString('\n')
    fmt.Print("Message from server: "+message)
  }
}

package cgoexample

/*
#include <stdio.h>
#include <stdlib.h>

void myprint(char* s) {
      printf("%s\n", s);
}
*/
import "C"

import "unsafe"

func Example() {
      cs := C.CString("Hello from stdio\n")
      C.myprint(cs)
      C.free(unsafe.Pointer(cs))
}
package gocallback

import "fmt"
```