Embedded firmware device driver development, performance analysis and optimizations

Major Project Report

Submitted in Partial fulfillment of the requirements
For the degree of
Master of Technology (M.Tech.)
in
Electronics & Communication Engineering
Embedded Systems
by

Vaghela Maulik V.
12MECE26

Department of Electronics & Communication Engineering
Institute of Technology
Nirma University
Ahmedabad - 382481
May-2014

Embedded firmware device driver development, performance analysis and optimizations

Major Project Report

Submitted in Partial fulfillment of the requirements
For the degree of
Master of Technology (M.Tech.)
in
Electronics & Communication Engineering
Embedded Systems
by

Vaghela Maulik V.
12MECE26

Dr. Nagendra Gajjar
Internal Guide

Mr. Madhusudan Kallapur
External Guide

Department of Electronics & Communication Engineering
Institute Of Technology
Nirma University
Ahmedabad - 382481
May- 2014

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in Embedded systems at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgement has been made in the text to all other material used.

Maulik Vaghela
12MECE26

Certificate

This is to certify that the dissertation work entitled "Embedded firmware device driver development, performance analysis and optimization" submitted by Maulik Vaghela (12MECE26), towards fulfillment of the requirements for the degree of Master of Technology in Embedded Systems of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place:Ahmedabad

Dr.N.P.Gajjar
Internal Project Guide,
Institute of Technology,
Nirma University
Ahmedabad

Dr.N.P.Gajjar
Program coordinator,
Institute of Technology,
Nirma University,
Ahmedabad

Dr.D.K.Kothari
Section head EC
Institute of Technology,
Nirma University,
Ahmedabad

Dr.P.N.Tekwani
Head of EE Dept.
Institute of Technology,
Nirma University,
Ahmedabad

Dr.K.Kotecha
Director
Institute of Technology,
Nirma University,
Ahmedabad

Acknowledgement

I would also thank to my Project coordinator and my internal guide, Dr. N.P.Gajjar, program-coordinator, embedded systems, Institute of Technology, Nirma University, Ahmedabad for giving valuable support for project work.

I would like to express my sincere thanks to Mr. Madhusudan Kallapur and Mr. Dhaval Sharma to give me this opportunity. I enjoyed their vast knowledge and thankful to them for having a profound impact on this report.

I would like to thank my Colleagues Soham Gandhi, Brijesh Yadav and Nishanth, Intel Technology India Private Limited, Bangalore for their valuable guidance. Throughout the training, they have given me much valuable advice on project work which I am very lucky to benefit from. Without them, this project work would never have been completed.

I would also like to thank Dr.D.K.Kothari, Section head EC dept. and Dr. P.N.Tekwani, head of EE dept., Institute of Technology, Nirma University, Ahmedabad for providing me an opportunity to get an internship at Intel Technology India Private Limited, Bangalore.

I would also like to thank Dr. K.Kotecha, Director, Institute of Technology, Nirma University, Ahmedabad for providing me an opportunity to get an internship at Intel Technology India Private Limited, Bangalore.

I would like to thank my all faculty members for providing encouragement, ex-changing knowledge during my post-graduate program.

I also owe my colleagues and friends in the Intel, special thanks for helping me on this path and for making project at Intel more enjoyable.

Maulik Vaghela
12MECE26

Abstract

Now days smart devices such as mobile, wearable devices uses sensors to sense the environment surrounding them and makes decision based on sensors data, simplest example is smart phone which adjusts brightness of screen automatically based on ambient light. Sensors are designed to continuously sense the surrounding hence they will use CPU every time and it will cause problem for battery operated devices. Now day mobile boasts processors which are on par with desktop processors and hence use much more power and to process data of a sensors, this power is too much. In order to reduce battery consumption, idea of sensor hub is born. Sensor hub is a co-processor which has very low power consumption and operates at very less frequency (few hundred of mega Hz) thus reducing loading on main processor. It results in much power saving because main processor can go to sleep and low power sensor co-processor can keep running.

Sensor hub requires firmware which calculates data of sensors and sends it to main processor when requested by main processor. Firmware development requires development of device driver for various sensors, developing supporting API (Application Programming Interface), developing arbitration logic for various tasks such as frequency setting, range setting etc. and also some algorithms to process sensor data take various decision. Since firmware requires calculation in real time, performance of firmware should meet real time requirement. Performance should be measured accurately in order to meet timeliness and it can be done using RDTSC ticks and if performance doesnt meet requirement design needs to be changed to meet requirement. Memory is also limited in this embedded system so code needs to be fit in limited memory.

- Development of micro-drivers for various sensors:
  Firmware requires sensor drivers to access data of hardware sensors, so micro-driver for sensor is a must for any firmware. If current drivers have some bug or missing functionality that also needs to be fixed. Focus on development of micro-drivers (which is different from Linux device drivers) such as accelerometer, gyroscope, magnetometer etc., adding functionalities such as range arbitration, enabling dynamic interrupt mode etc., fix bugs in existing micro-drivers and test its functionality.

- Performance analysis and optimization :
  This task includes measuring performance of firmware accurately on early hardware prototype (FPGA) and optimizes it to meet certain timing requirements. Code also needs to be optimized to meet memory requirement of an embedded system.

Contents

i

ii

Declaration **iii**

Certificate **iv**

Acknowledgement **v**

Abstract **vi**

1 Introduction **1**
1.1 Introduction and motivation for project 1
1.2 Introduction to sensors 2
1.2.1 Types of sensors 2
1.2.2 Classification of physical sensors 2
1.3 Various terms related to sensors 3
1.4 Widely used physical sensors in various devices 4
1.5 Introduction to firmware 5
1.6 summary 5

2 Device driver development **7**
2.1 Introduction to device drivers 7
2.2 Developing simple drivers for Linux 8
2.2.1 Advantages Microdrivers over device drivers 10
2.3 Development of micro-drivers for accelerometer LSM303DLHC 10
2.4 Example code for accelerometer LSM303DLHC 12
2.4.1 Initialization of accelerometer LSM303DLHC 13
2.4.2 Configuration of accelerometer LSM303DLHC 16
2.4.3 Reading data of accelerometer LSM303DLHC 18
2.5 Results 20
2.6 summary 24

3 Performance analysis **25**
3.1 Performance measurement using clock function 25
3.2 Performance measurement using rdtsc instruction 28
3.3 summary 31

4 Code debugging using GDB **32**
4.1 Introduction to code debugging 32
4.2 summary 35

5 Secure coding **36**
5.1 Common vulnerabilities in code 36
5.2 Introduction to klocwork 42
5.3 Example of use of klocwork 43
5.4 summary 50

6 Conclusion and future scope **51**

7 References **53**

List of Figures

1.1 Real time firmware architecture 5

2.1 Operating system architecture 7
2.2 Example code for simple hello world module 9
2.3 Control register 1 structure 13
2.4 Control register 1 description 13
2.5 output data rate selection 14
2.6 Control register 2 structure 14
2.7 Control register 2 description 14
2.8 High pass filter mode configuration 14
2.9 Control register 4 structure 15
2.10 Control register 4 description 15
2.11 LSM303DLHC initialization code 16
2.12 LSM303DLHC configuration code part-1 17
2.13 LSM303DLHC configuration code part-2 18
2.14 LSM303DLHC status register structure 18
2.15 LSM303DLHC status register description 19
2.16 LSM303DLHC output registers 19
2.17 code to read data of LSM303DLHC 20
2.18 Data of accelerometer on sensor diagnostic tool 21
2.19 Data of magnetometer on sensor diagnostic tool 22
2.20 Data of gyrometer on sensor diagnostic tool 23

3.1 Use of clock library in example code 26
3.2 1ms time required to sort 6 strings 27
3.3 2ms time required to sort 10 strings 27
3.4 Use of rdtsc instruction in example code 29
3.5 ticks required to sort 6 strings 30
3.6 ticks required to sort 10 strings 31

4.1 help command in GDB 34
4.2 Breakpoint at specific line 34
4.3 Breakpoint at specific function 34
4.4 stepping through code 34
4.5 checking value of variable 35

5.1 Example code showing validation of an array index 37
5.2 Improper validation of an array index 38
5.3 Null termination in strings 39
5.4 Improper memory release 41

5.5 Example code(part1) 44
5.6 Example code(part2) 45
5.7 Creating new project in klocwork 46
5.8 Selecting location ad project type 46
5.9 Opening kwshell 47
5.10 Generating detailed report 47
5.11 Detailed report 48
5.12 Modified code to remove KW error 49

1 Opening a device in Linux 1
2 set I2C slave address to accel's I2C address 2
3 set I2C slave address to accel's I2C address 2

Chapter 1

Introduction

1.1 Introduction and motivation for project

Now days, sensors play an important role in every mobile devices and it will become more important for wearable devices in future. Sensor is device which converts physical quantities into electrical signal which can be understood by an electronic device today. Sometime signal needs to be converted from analog to digital. Sensors which are used in mobile devices are accelerometer, gyroscope, ambient light sensor, proximity sensor etc. Based on data of these sensor device takes certain decisions for example, based on data from gyroscope mobile can be used to rotate screen, based on data from ambient light sensor device can reduce or increase brightness of screen which results in power saving and is also comfortable for end user.

In order to process data of various sensors, one needs to develop an algorithm which will run and make decisions based on data from sensor. These algorithms may require computing power but mobile processor now days boasts a processing power which is much more than required. Most sensors are continuously powered on for a device, thus these algorithms will not allow processor to go to sleep mode. In this scenario battery life of a device will be reduced which is not desirable for battery powered embedded device. Imagine a wearable device such as smart watch whose battery lasts for a half a day.

To solve the problem idea of sensor hub is born. A sensor hub is a device which contains a micro-controller unit or sometimes DSP (Digital Signal Processing) unit which helps to integrate data from different sensors and process them. Sensor hub can also help to off-load these processing from main processor thus improving system performance. While idle main processor can also go into sleep state thus reducing battery consumption while sensor hub keeps running in background mode. Sensor hub acts as a co-processor and it consumes very less power because it doesn't require much computing power and it has less operating frequency (In order of few MHz (mega Hz)).

Sensor hub is currently emerging technology. There are very few sensor hubs in market. For example, Atmels sensor hub microcontroller is used in many mobile devices uses tiny AVR micro-controller and another is NXPs LPC18A microcontroller. Both can be used as a co-processor and can be used to develop sensor hub by user. Intel's sensor hub is based on x86 architecture and it features system which has Hardware, Firmware and also interface to operating system thus making system development easier.

1.2 Introduction to sensors

Sensors which are used in current devices (whether mobile device, wearable or any other device) are MEMs (MicroElectroMechanical systems) also known as micromachines. These sensors are based on nano-technology and contain components ranging from 1 to 100m. In first generation of MEMS sensor element was mostly based on silicon structure and sometimes chip was combined with external analog amplification. Second generation of MEMS sensors had sensor element, analog amplifier and ADC (analog to digital converter) on a single chip. Third generation of MEMS sensor also had temperature compensation on a same chip. In fourth generation of MEMS devices also have additional memory cells which can store calibration data and also have temperature compensation data on a same chip. Some sensor also provides FIFO (first in first out) memory in order to store few data samples of sensors.

1.2.1 Types of sensors

Mostly sensor is classified into two categories. Its not actual classification but to differentiate between actual sensor and sensor which takes action, there are two different categories.

- **Physical sensors**

  Physical sensor mean actual sensor part or sensor chip which converts physical quantity into electrical signals. Example of these sensors are Accelerometer (ST LSM303DLHC, ST LSM303D etc.), Gyroscope (ST L3GD30H, MPU 6050 etc.) etc. Data which comes from these chips is in digital form and it is called as raw data as its simply output based on physical quantity.

- **Virtual sensors**

  Virtual sensors consume data of actual sensor and apply some algorithm on top of it to produce some meaningful output. For example, actual output of magnetometer sensor is in form of (x, y, z) axis which tells magnetic fields in 3 directions. It doesnt provide any meaningful data to user. System developer can develop an algorithm which calculates north direction from this output and output of algorithm makes sense to user.

1.2.2 Classification of physical sensors

Physical sensors can be also divided into three categories.

- **Motion sensors**

  Motion sensors are used to determine motion and their data can be used to

perform different functionalities for example accelerometer data can be used to determine rotation of screen for mobile phones. It can be also used while playing gesture based games.

- **Location sensors**
  Location sensors are used to provide user exact location and can also be used to give direction for navigation. Best example of location sensor is GPS used in mobile phones. Now a days GLONASS (Global Navigation Satellite System) is replacing GPS because of worldwide compatibility and comparable precision.

- **Context sensors**
  Context sensors are used when user is directly involved. For example Proximity sensor is used when user is holding mobile near to ear and its data can be used to turn-off the screen. Other example is Human presence sensor which can be used to determine whether Human is present or not.

1.3 Various terms related to sensors

This section includes various terminology used in context with sensors, which can be helpful to understand datasheets of various sensors. It can also be useful to design a driver for various sensors.

- **Range**
  Sensors are designed to work over specific range. The design ranges are fixed and care must be taken because if range exceeds then it may result in permanent damage of a sensor parts or sometimes it may give unpredictable output. Thus it is mandatory to operate sensor in defined range.

- **Sensitivity**
  Sensitivity means change in output of a sensor per unit change in input quantity being measured by sensor. Sensitivity of a sensor may be constant or it can be programmable between certain values. For example, in case of accelerometer lsm303dlhc one can select sensitivity between 4 available values.

- **Calibration**
  Calibration is required for sensors. If accurate output is to be measured by a sensor than it is necessary to check output of sensor with accurately known input value. If output is changed than one needs to supply calibration offsets to driver which will help to produce accurate output.

- **Resolution**
  Resolution means smallest change that can be detected by sensor. Developer

needs to choose sensor part based on resolution required by an application.

1.4 Widely used physical sensors in various devices

This section describes various sensors which are widely used in various devices such as accelerometer, gyroscope, magnetometer and ambient light sensors.

- **Accelerometer**

  Accelerometer measures acceleration means velocity in particular direction. Unit of measurement is g, which is earths g-force acceleration. Thus Accelerometer at rest will measure acceleration g (=9.8 m/s^2) and at free fall it will measure 0. Accelerometers used in mobile are multi-axes accelerometer which can measure acceleration in multiple axes and thus we can find acceleration in particular direction and also can detect shock, vibration, free fall etc

  Example of accelerometer is ST LSM303DLHC. This accelerometer IC is from ST microelectronics and It is multi axes accelerometer. It can measure acceleration in 3 directions and it provides 16 bit output. It also features embedded FIFO which can store acceleration data and thus reducing loading on host processor.

- **Gyroscope**

  Gyroscope is used to measure rotation / angular velocity. Unit of measurement is DPS (degree per second). Gyroscopes are widely used in inertial navigation system such as in Hubble Telescope. It is also used in mobile for gesture recognition and to play games based on gesture. Gyroscope also can be multi-axes. Gyroscope can have different range such as 250dps, 500dps or 2000dps. We can configure it to different ranges to measure proper rotation.

  Example of gyroscope is ST L3GD20. This gyroscope IC is from ST microelectronics. It is 3-axis gyroscope and it gives data for yaw, pitch and roll. It gives 16-bit output for each axis. It has full scale range up to 2000dps.

- **Magnetometer**

  Magnetometer is device which can be used to measure magnetic field in particular direction and unit of measurement is gauss. Magnetometer data can be combined with accelerometer data to measure accurate north heading.

  Example of magnetometer is ST-LSM303D. This IC for Magnetometer is also from ST microelectronics. It combines Accelerometer and compass. For magnetometer it provides range up to 12 gauss. For accelerometer and magnetometer it provides 16 bit output and accelerometer data can be used for tilt compensation to give user accurate north heading even if device is inclined.

- **Ambient Light Sensor (ALS)**

  ALS is used to measure light luminance in surrounding environment and unit of measurement is LUX. It is used in mobile for auto-brightness adjustment.

1.5 Introduction to firmware

"Firmware is a program which is written to read only memory (ROM) of a computing device and it is used to run user program on the device."

Most of the electronic devices have firmware, for example laptop/desktop PC also have firmware which is known as boot loader or BIOS (Basic Input/ Output Systems) whose task is to initialize hardware of a computer and load operating system. In real time embedded systems contains code which runs always and it is programmed into ROM of a device which is also a firmware. For real time embedded system firmware must be able to handle timeliness of each function and so firmware development requires more effort because it must be tested for every possible event and since embedded system are mostly connected to hardware such as motor drives, actuators etc. failure of system may cause large penalties and may cause harm to people. For real time embedded firmware architecture is as follow:

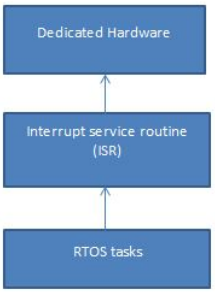


Figure 1.1: Real time firmware architecture

As can be seen from above figure, in real time embedded system the time deadliness should be pushed into hardware and interrupt service routines should be as small as possible otherwise most of the system time will go in processing an interrupt and sometimes system may miss out an important interrupt. Its important to understand requirement from user perspective and system must always meet this requirement. For example, some elevator may have option for simple game play but game should be running at low priority and safety of passenger is highest priority.

1.6 summary

In this chapter motivation for this project and sensor hub is described. Introduction to sensors is given where mainly two types of sensor physical and virtual are

described. Physical sensors are again divided in 3 types like location based, motion based and context based sensors. Some widely used terms related to sensors such as range, sensitivity, resolution etc is described and mostly used physical sensors like accelerometer, magnetometer, gyroscope and ambient light sensors are also described with an example.At last basic introduction of firmware is also give. In the next chapter device drivers which is most important part of a firmware is explained in detail.

Chapter 2

Device driver development

Device driver is most basic need of any firmware because any firmware needs to communicate the hardware device either to control it or get any information such as output data from device. This chapter first introduces the device driver concept which is widely used in Linux operating system as a kernel module and then new concept of micro-drivers is explained with an example of accelerometer sensor ST LSM303DLHC.

2.1 Introduction to device drivers

"Device driver is a program which is used to control the device attached to processor. "

Driver simplifies programming because it acts as a translator between Hardware device and programs / operating system which uses the device. Generally drivers are specific to hardware and operating system.

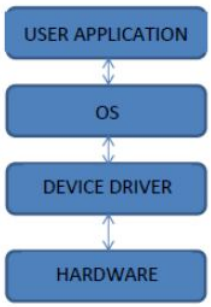


Figure 2.1: Operating system architecture

As shown in fig.2.1., when user application or operating system wants to use hardware it invokes routine in driver which sends command to hardware. Once hardware/device sends required data to driver, driver can call back routine to operating system or driver can send interrupt to calling process.

For any device drivers it is important to make distinction between kernel space and user space. Kernel space manages machine hardware in simple and efficient manner allowing simple programming interface. When drivers are written in kernel space then it provides interface between user and hardware as can be seen in figure 1. User space contains user application for example, GUI (Graphical User Interface) based application. These applications also need to interface with hardware but they cant directly interface with hardware, they have to call kernel supported functions.

2.2 Developing simple drivers for Linux

Kernel provides special functions or subroutines for user space to interact with kernel. From user point of view device can be seen as a file in UNIX systems. In kernel space also Linux provides several API to perform low level interaction directly with hardware and also transfer data from kernel to user space.

Linux kernel provides special APIs (Application Peripheral Interface) to load driver module to kernel, install module to kernel and once use of module is finished you can also remove this module from kernel. Basic commands are as follow

- # insmod hello.ko
  The insmod command allows the installation of the module in the kernel.

- # lsmod
  Using this command it is possible to check that the module has been installed correctly by looking at all installed modules.

- # rmmod hello

  Finally, the module can be removed from the kernel using this command. By issuing the lsmod command again, you can verify that the module is no longer in the kernel.

When device driver is loaded into kernel then some necessary tasks are performed first like resetting the device, reserving RAM, reserving interrupt, reserving input/output ports etc. These tasks are performed by two functions module_init and module_exit function which are in kernel space. These functions are same as insmod and rmmod of user space and both internally used kernel space functions.

As an example simple code of hello world is shown below. This code prints hello world when module is loaded into system and Bye world when we remove module from system.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("Dual BSD/GPL");

static int hello_module_init(void) {
        printk(" Hello world!\n");
        return 0;
}

static void hello_module_exit(void) {
        printk(" Bye, cruel world\n");
}

module_init(hello_module_init);
module_exit(hello_module_exit);
```

Figure 2.2: Example code for simple hello world module

Printk function is same as printf function in user mode but printk is used to print message in kernel mode. We can also give priority of message by putting ¡1¿ in printk , lower number means high priority. To give Hello world message high priority message should be written as follow:

Printk(“< 1 > Hello world ”); //< 1 > indicates high priority.

In Linux developers can develop driver as kernel module or loadable modules. Main advantage of loadable module is that It can be loaded while required and once its job is finished it can be removed from kernel thus saving memory for kernel.

There are also disadvantage of drivers. Device driver executes in kernel to achieve high performance and to use kernel services but it will decrease system reliability. It also makes programming of device drivers difficult because programmers cant use development tools for user programs. Any fault in kernel driver can cause entire operating system to fail. One solution for stated problem is to use user mode device drivers but it will reduce system performance and also we need to rewrite existing drivers because of new interfaces. Another solution to this problem is to use micro-drivers.

Micro-drivers solve the problem of reliability by moving drivers out of kernel and it will just keep critical code in kernel space thus maintaining almost same performance.

One can keep critical operation code such as I/O operation in kernel space thus allowing them to run at full speed while some management task such as initializing and configuring device can run in user space which will run at reduced speed. Thus it provides intermediate solution between user mode driver and kernel driver.

According to experiments done at University of Wisconsin-Madison[6] , 65% of driver code can be moved out of kernel without affecting common case performance.

2.2.1 Advantages Microdrivers over device drivers

- **User level programming and debugging**

  The user drivers can be compiled with standard user mode tools and it can also be debugged with standard user mode tools. Thus it allows source level debugger tools to debug driver code also.

- **Good common case performance**

  Micro-drivers can provide performance which is comparable to drivers which are fully executing in kernel mode. Thus it provides better performance than driver which are fully executing in user space.

- **Bug Isolation**

  Micro-drivers can isolate bugs better than traditional drivers since faulty user mode driver cannot crash an operating system. Even experiment[2] Shows that kernel survives null pointer dereference in user mode driver.

- **Compatibility**

  Micro-drivers are compatible with existing drivers thus we can reuse same driver code to convert into micro-driver. There is also tool available called Driver slicer which will divide driver code into kernel and user space. Thus it reduces programming effort to great extent.

2.3 Development of micro-drivers for accelerometer LSM303DLHC

In this section, micro-driver development of accelerometer LSM303DLHC is explained. LSM303DLHC is combination of accelerometer and magnetometer in a single IC (integrated circuit). LSM303DLHC chip is from ST microelectronics. Some of the features of LSM303DLHC are as follows [1]:

- 3 magnetic field channels and 3 acceleration channels
- From 1.3 to 8.1 gauss magnetic field full scale
- 2g/4g/8g/16g selectable full-scale
- 16 bit data output
- I2C serial interface
- Analog supply voltage 2.16 V to 3.6 V
- Power-down mode/ low-power mode

- 2 independent programmable interrupt generators for free-fall and motion detection
- Embedded temperature sensor
- Embedded FIFO
- 6D/4D orientation detection

Applications for LSM303DLHC are as follow[1]:

- Compensated compass
- Map rotation
- Position detection
- Motion-activated functions
- Free-fall detection
- Click/double click recognition
- Pedometer
- Intelligent power-saving for handheld devices
- Display orientation
- Gaming and virtual reality input devices
- Impact recognition and logging
- Vibration monitoring and compensation

To develop micro-drivers for sensor, one need to understand register structure for given IC and which values need to be written to register for given configurations. Every sensor driver must contain following sub-routines:

- Initialization

  Initialization means writing a default register values in some of the registers when sensor is initialized. When we turn on the sensors, its registers will contain some default values *mostlyallzeros* but if you want to modify some values you should modify it in initialization routine.

  For example, in case of accelerometer during initialization we can specify whether we want to enable FIFO mode or not, to operate accelerometer in high resolution mode or low resolution mode, default operating range, default operating frequency, enable or disable interrupt etc.

- Configuration

  Configuration means we can change various configuration of accelerometer during run time for example; we can change frequency of accelerometer after initialization also. This is required because some application requires accelerometer to be running at higher frequency, and then we should allow user or application to change frequency of accelerometer. Another example is range.

- Read data

  Once sensor is initialized and configured properly, data needs to be read continuously and at the same rate required by applications.

Sometimes some arbitration routine is also required. For example, when one or more application tries to configure frequency of sensor then sensor should be operated at highest required frequency and same in case of range also.

2.4 Example code for accelerometer LSM303DLHC

LSM303DLHC supports both I2C (Inter Integrated Connect) and SPI (Serial Peripheral Interface) interface, but in this example we'll use I2C as most of systems supports I2C interface and if some systems dont support it we can bit-bang I2C interface easily. I2C drivers for Linux operating system and bit-banging for I2C (using 8051 microcontroller) is explained in appendix A.

Some of I2C terminologies is explained as follow:

- Transmitter: Device which sends data to bus.
- Receiver: device which receives data from bus.
- Master: The device which initiates a transfer, generates clock signals and terminates a transfer.
- Slave: The device addressed by the master.

There are two signals associated with I2C, clock signal SCL and data signal SDL. The latter is bi-directional line used for sending and receiving data to/from the interface.

I2C slave address for accelerometer is 0x19. Every slave on I2C has its own I2C address. LSM303DLHC provides feature called Auto address increment, which allows driver to read multiple bytes in a single I2C transaction. In order to enable this feature one need to logically OR register address with 0x80. It means make MSB of register address 1. This way by reading multiple bytes in a single transaction, performance of driver can be improved because I2C transactions are time consuming.

2.4.1 Initialization of accelerometer LSM303DLHC

In order to initialize accelerometer we need to write default values to its control registers. We need to perform following operations.

- Determine which registers are to be written for initialization.
- Determine default values for each registers.
- Write each registers over I2C.

Following are registers which needs to be written to initialize for accelerometer LSM303DLHC. These registers definitions are taken from datasheet which can be found at link given in a reference.

- CTRL_REG_1A (address = 0x20)

ODR3	ODR2	ODR1	ODR0	LPen	Zen	Yen	Xen

Figure 2.3: Control register 1 structure

ODR 3-0	Output data rate selection. (0000: Power down mode) For other values refer table 3.
Lpen	Low-power mode enables. Default value: 0 (0: normal mode, 1: low-power)
Zen	Z axis enables. Default value: 1 (0: Z axis disabled, 1: Z axis enabled)
Yen	Y axis enables. Default value: 1 (0: Y axis disabled, 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled, 1: X axis enabled)

Figure 2.4: Control register 1 description

ODR3	ODR2	ODR1	ODR0	Power mode
0	0	0	0	Power down mode
0	0	0	1	Normal / low-power mode (1 Hz)
0	0	1	0	Normal / low-power mode (10 Hz)
0	0	1	1	Normal / low-power mode (25 Hz)
0	1	0	0	Normal / low-power mode (50 Hz)
0	1	0	1	Normal / low-power mode (100 Hz)
0	1	1	0	Normal / low-power mode (200 Hz)
0	1	1	1	Normal / low-power mode (400 Hz)
1	0	0	0	Low-power mode (1.620 KHz)
1	0	0	1	Normal (1.344 kHz) / low-power mode (5.376 KHz)

Figure 2.5: output data rate selection

For initialization, low power should be enabled to save the power and sensor should be in normal mode once configuration is applied. Thus for control register 1A value to be written is 0x0F.

Ctrl_reg_1A = 0x0F; //Enable low power mode and all axis.

- CTRL_REG_2A (address = 0x21)

HPM1	HPM0	HPCF2	HPCF1	FDS	HPCLICK	HPIS2	HPIS1

Figure 2.6: Control register 2 structure

HPM1 -HPM0	High pass filter mode selection. Default value: 00
HPCF2 - HPCF1	High pass filter cut-off frequency selection
FDS	Filtered data selection. Default value: 0 (0: internal filter bypassed, 1: data from internal filter sent to output register and FIFO)
HPCLICK	High pass filter enabled for CLICK function. (0: filter bypassed, 1: filter enabled)
HPIS2	High pass filter enabled for AOI function on Interrupt 2, (0: filter bypassed, 1: filter enabled)
HPIS1	High pass filter enabled for AOI function on Interrupt 1, (0: filter bypassed, 1: filter enabled)

Figure 2.7: Control register 2 description

HPM1	HPM0	High pass filter mode
0	0	Normal mode (reset reading HP_RESET_FILTER)
0	1	Reference signal for filtering
1	0	Normal mode
1	1	Autoreset on interrupt event

Figure 2.8: High pass filter mode configuration

For control register 2, High pass filter should be in normal mode only (value: 00) and Filtered data selection should be 1 , other setting should be as default values. Thus control register 2 should have value 0x80.

Ctrl_reg_2 = 0x80.

- CTRL_REG_4A (address = 0x23)

  Control register 4 can be used to enable block data update, setting full scale range, enabling high resolution mode and SPI interface mode selection.

BDU	BLE	FS1	FS0	HR	0 [1]	0[1]	SIM

[1] This bit must be set to '0' for correct working of the device.

Figure 2.9: Control register 4 structure

BDU	Block data update. Default value: 0 (0: continuous update, 1: output registers not updated until MSB and LSB Reading)
BLE	Big/little endian data selection. Default value 0. (0: data LSB @ lower address, 1: data MSB @ lower address)
FS1-FS0	Full-scale selection. Default value: 00 (00: +/- 2G, 01: +/- 4G, 10: +/- 8G, 11: +/- 16G)
HR	High resolution output mode: Default value: 0 (0: high resolution disable, 1: high resolution enable)
SIM	SPI serial interface mode selection. Default value: 0 (0: 4-wire interface, 1: 3-wire interface).

Figure 2.10: Control register 4 description

For control register 4, following settings should be preferable:

- Output registers not updated until MSB and LSB reading.
- Full scale can be selected based on application. Currently keeping default value.
- Enable high resolution mode.

Thus control register 4's value will be 0x88.

Other control registers are described in data sheet given in appendix B. Other initialization configuration can also be done for enabling FIFO mode, enabling interrupt mode and different configurations related to interrupt.FIFO mode can be enabled, if data is needed to be stored in FIFO memory of accelerometer. This can be useful if samples need to be read later but FIFO can store only 32 samples, after this it will overwrite old samples. In order to use FIFO for accelerometer, first FIFO must be enabled by writing FIFO_EN $FIFOenable$ bit in control register 5 and then selecting operating mode for FIFO in FIFO control register.

Code for initialization of accelerometer is given in next page. This code depends

upon kernel and also which firmware is being used. Code just indicates initialization process, driver developers also need to attach code to kernel and for Linux, and probe must be created for code while writing code in kernel mode. In this example code, Linux's smbus I2C APIs are being used and how to initialize I2C for a device is shown. For more details regarding I2C APIs in Linux, please refer appendix A.

```
#include <linux/i2c-dev.h>

#define ctrl_reg1  0x20
#define ctrl_reg2  0x21
#define ctrl_reg4  0x24

void accel_init()
{
    int file;  // File descriptor.

    // 1 is adapter which may vary depending upon system.
    char file_name[20] = "/dev/i2c-1";

    file = open(file_name, O_RDWR);

    if(file < 0){
        printf("failed to open I2C adapter \n");
        return;
    }
  /* The I2C slave address of accelerometer */
     int addr = 0x19;

     if (ioctl(file, I2C_SLAVE, addr) < 0) {
        printf("failed to configure I2C adapter \n");
        return;
      }

    // Write 0x0F at address of control reg. 1.
    i2c_smbus_write_byte(file , ctrl_reg1 , 0x0F );

    // Write 0x80 at address of control reg. 2.
    i2c_smbus_write_byte(file , ctrl_reg2 , 0x80 );

    // Write 0x88 at address of control reg. 4.
    i2c_smbus_write_byte(file , ctrl_reg4 , 0x88 );

}
```

Figure 2.11: LSM303DLHC initialization code

2.4.2 Configuration of accelerometer LSM303DLHC

In configuration we can configure frequency and range (full scale range) for sensors. For accelerometer we need to write control register 1 and control register 4. It depends upon request from user or application.

```
1   void accel_config(int freq, int scale)
2   {
3       int freq_bit = 1, scale_bit = 0;    // Default freq: 1Hz, scale = 2G.
4       int tmp_ctrl_reg1;
5       int tmp_ctrl_reg4;
6
7       switch(freq){
8
9           case 0:
10              freq_bit = 0;
11              break;
12
13          case 1:
14              freq_bit = 1;
15              break;
16
17          case 10:
18              freq_bit = 2;
19              break;
20
21          case 25:
22              freq_bit = 3;
23              break;
24
25          case 50:
26              freq_bit = 4;
27              break;
28
29          case 100:
30              freq_bit = 5;
31              break;
```

Figure 2.12: LSM303DLHC configuration code part-1

```
33          case 200:
34              freq_bit = 6;
35              break;
36
37          case 400:
38              freq_bit = 7;
39              break;
40
41          default:
42              printf("frequency not supported \n");
43              break;
44      }
45
46      switch(scale){
47
48          case 2:
49              scale_bit = 0;
50              break;
51
52          case 4:
53              scale_bit = 1;
54              break;
55
56          case 8:
57              scale_bit = 2;
58              break;
59
60          case 16:
61              scale_bit = 3;
62              break;
63      }
64
65      tmp_ctrl_reg1 = i2c_smbus_read_byte(file, ctrl_reg1);          // Read control register
66      tmp_ctrl_reg1 |= freq_bit;
67      i2c_smbus_write_byte(file, tmp_ctrl_reg1, ctrl_reg1);
68
69      tmp_ctrl_reg4 = i2c_smbus_read_byte(file, ctrl_reg4);          // Read control register
70      tmp_ctrl_reg4 |= scale_bit;
71      i2c_smbus_write_byte(file, tmp_ctrl_reg4, ctrl_reg4);
72
73  }
```

Figure 2.13: LSM303DLHC configuration code part-2

2.4.3 Reading data of accelerometer LSM303DLHC

Once sensor is configured properly, sensor's data can be read from data output registers and status register. First status register needs to be read and if new data is available then and then data can be read from data output registers.

Sensor's status register and data output register's structure and description is given below.

ZYXOR	ZOR	YOR	XOR	ZYXDA	ZDA	YDA	XDA

Figure 2.14: LSM303DLHC status register structure

ZYXOR	X, Y, and Z axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new set of data has overwritten
ZOR	Z axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new data for the Z-axis has overwritten the previous one)
YOR	Y axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new data for the Y-axis has overwritten the previous one)
XOR	Z axis data overrun. Default value: 0 (0: no overrun has occurred, 1: a new data for the X-axis has overwritten the previous one)
ZYXDA	X, Y, and Z axis new data available. Default value: 0 (0: a new set of data is not yet available, 1: a new set of data is available)
ZDA	Z axis new data available. Default value: 0 (0: a new data for the Z-axis is not yet available, 1: a new data for the Z-axis is available)
YDA	Y axis new data available. Default value: 0 (0: a new data for the Y-axis is not yet available, 1: a new data for the Y-axis is available)
XDA	X axis new data available. Default value: 0 (0: a new data for the X-axis is not yet available, 1: a new data for the X-axis is available)

Figure 2.15: LSM303DLHC status register description

OUT_X_L_A (28h), OUT_X_H_A (29h)

X-axis acceleration data. The value is expressed in 2's complement.

OUT_Y_L_A (2Ah), OUT_Y_H_A (2Bh)

Y-axis acceleration data. The value is expressed in 2's complement.

OUT_Z_L_A (2Ch), OUT_Z_H_A (2Dh)

Z-axis acceleration data. The value is expressed in 2's complement.

Figure 2.16: LSM303DLHC output registers

```
#define OUT_X_L     0x28
#define OUT_X_H     0x29
#define OUT_Y_L     0x2A
#define OUT_Y_H     0x2B
#define OUT_Z_L     0x2C
#define OUT_Z_H     0x2D
#define STATUS_REG  0x27

int output[3] = {0};

void accel_read_data()
{
    int status = 0;

    status = i2c_smbus_read_byte(file , STATUS_REG);

    /* If all axis data ready then and then read data*/
    if( status & 0x80){

        Output[0] = i2c_smbus_read_word_data(file, OUT_X_L);
        Output[1] = i2c_smbus_read_word_data(file, OUT_Y_L);
        Output[2] = i2c_smbus_read_word_data(file, OUT_Z_L);
    }
}
```

Figure 2.17: code to read data of LSM303DLHC

These three routines are necessary for driver but additional routines can be added for frequency and range arbitration and interrupt can be enabled. The full code for reference is given in appendix B.

2.5 Results

Once sensor driver is created, certain steps are required to expose it to host containing different operating system. Here example of windows 8 is taken, and results are shown. First every sensor needs to have persistent unique identifier (PUID). Each sensor should create its own identifier and should have access to it. You can also filter the data of sensor using sensor diagnostic tool. Sensor diagnostic tool is a tool from Microsoft which helps developers to test their drivers, firmware as well as hardware. Sensor diagnostic tool provide option called change sensitivity which updates data only when data goes above sensitivity. For example, change sensitivity has been set to 1 then data will be only uploaded to host when difference between last uploaded sample and current sample is 1 G in case of accelerometer. One thing to be noted is developer needs to add support for change sensitivity in his/her code.

After developing driver for accelerometer lsm303dlhc, driver is added to ISH firmware which also has host interface to windows 8. Drivers can be exposed on sensor diagnostic tool. Results are shown as below for all three drivers e.g.,accelerometer, gyro meter and magnetometer.

Figure 2.18: Data of accelerometer on sensor diagnostic tool

Figure 2.19: Data of magnetometer on sensor diagnostic tool

Figure 2.20: Data of gyrometer on sensor diagnostic tool

As shown in figure 2.18 ,accelerometer data can be observed on sensor diagnostic tool. Actual data of accelerometer is data of X, Y and Z axis which is shown as
SENSOR_DATA_TYPE_ACCELERATION_X_G,
SENSOR_DATA_TYPE_ACCELERATION_Y_G &
SENSOR_DATA_TYPE_ACCELERATION_Z_G.

As shown in figure 2.19 ,magnetometer data can be observed on sensor diagnostic tool. Actual data of magnetometer chip is data of X, Y and Z axis which is shown as
SENSOR_DATA_MAGNETIC_FIELD_STRENGTH_X_MILLIGAUSS,
SENSOR_DATA_MAGNETIC_FIELD_STRENGTH_Y_MILLIGAUSS &
SENSOR_DATA_MAGNETIC_FIELD_STRENGTH_Z_MILLIGAUSS.
Field which shows magnetic north heading is actually algorithm which calculates north direction from data of X, Y and Z axes.

As can be seen in figure 2.20, gyroscope data can be shown as velocity per degree per second in all 3 directions shown by fields
SENSOR_DATA_TYPE_ANGULAR_VELOCITY_X_DEGREES_PER_SECOND,
SENSOR_DATA_TYPE_ANGULAR_VELOCITY_Y_DEGREES_PER_SECOND &
SENSOR_DATA_TYPE_ANGULAR_VELOCITY_Z_DEGREES_PER_SECOND.

2.6 summary

In this chapter device drivers which is most important part of sensor hub firmware is explained. In first section Linux device drivers which are also known as kernel mode device drivers is with an example. If there is a mistake in kernel mode device driver then it can cause whole firmware to crash and it's also very difficult to debug. To overcome this disadvantage device drivers can be written in user space and it suffers from poor performance issue because kernel call will take much time. So new concept of micro-drivers of evolved and micro-drivers combine advantages of both kernel mode drivers and user mode drivers.In micro-drivers critical tasks such as input-output are kept in kernel and user mode can just call these function. Micro drivers also has certain advantages. After that example of micro-driver for accelerometer LSM303DLHC is explained in detail. In next chapter performance analysis is explained in detail.

Chapter 3

Performance analysis

Performance of code can be described as a time required to execute specific section of code. For example, Driver code contains different code for events like one section for initializing device, configuration of device and another for reading data from device. Thus we may want to measure time required just for initialization of device. One may want to measure time for configuration or to read and process data. Thus one cam measure performance in context of different event.

There are two ways to measure timings in c.
1.Use in-built ́time.hĺibrary.
2.Measuring number of cycles taken by processor.

When we use time.h library we can directly use clock() function which is in-built function in library to get the time. Thus measuring time difference across any section we can measure time taken by any function or any section of the code. Main disadvantage of this method is that it provides time resolution in milliseconds and it's not accurate in many systems.

To overcome this disadvantage we can use assembly instruction **rdtsc** which gives number of ticks of processor. Note that value returned by rdtsc instruction is 64 bit long and thus we should use 64 bit variable to store value and once we get number of processor cycle we can easily convert it to any time resolution we want. Thus using assembly instruction we can measure time resolution in microseconds and Nano-seconds. How to use both methods in code is explained in following sections.

3.1 Performance measurement using clock function

Clock function is in-built function in time.h header file and it must be included in code. This function will return result which is of type clock_t type so variables which stores time must be of type clock_t.

In order to measure performance we need to insert some code which is shown by black boxes in following figure.Note that whole code is given for reference in appendix.

```
 9  # include<stdio.h>
10  #include<conio.h>
11  #include<stdlib.h>
12  #include <string.h>
13  #include <time.h>
14  //#include <intrin.h>
15
16  #define MAXLINE 10
17
18  char *lineptr[MAXLINE];
19
20  void readline(char *lineptr[], int nline);
21  void writeline(char *lineptr[], int nline);
22  void swap(char *v[],int i,int j);
23  void sort(char *lineptr[],int nline);
24
25  //#pragma intrinsic(__rdtsc)
26
27  int main()
28  {
29
30      int nline;
31      nline = 6;
32      clock_t start, stop;
33
34
35      readline(lineptr,nline);
36      start = clock();
37      sort(lineptr,nline);
38      writeline(lineptr,nline);
39      stop = clock();
40
41      printf("\n time %d \n",stop - start);
42      getch();
43
44      return 0;
45  }
```

Figure 3.1: Use of clock library in example code

When we measure time required to sort the given strings using above code, It

will give output in milliseconds as shown in following figures 3.2. and 3.3.

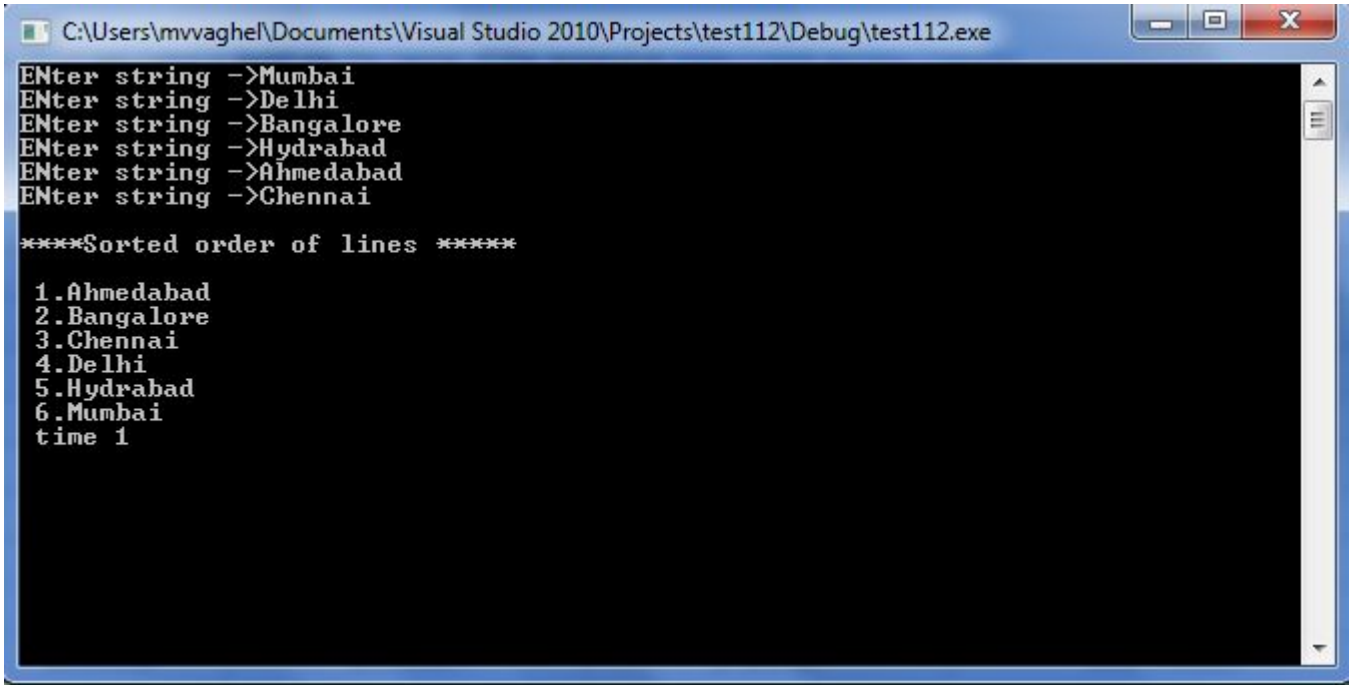


Figure 3.2: 1ms time required to sort 6 strings

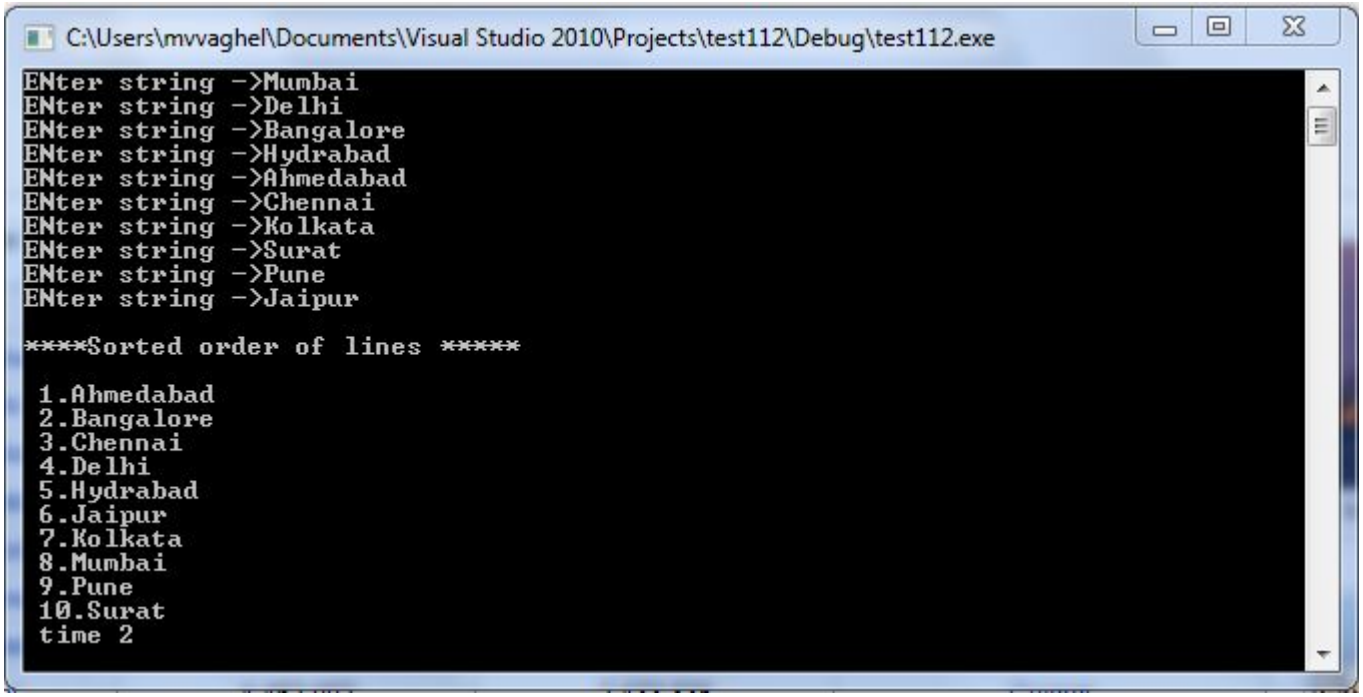


Figure 3.3: 2ms time required to sort 10 strings

3.2 Performance measurement using rdtsc instruction

Rdtsc is an assembly instruction which gives accurate count of cycle on processor. Starting from Pentium processors Intel processors allow programmers to access rdtsc register. Rdtsc register keeps an accurate count of every clock cycle on processor and its value is set to zero when processor resets. Rdtsc is 64 - bit register which is incremented on every clock cycle. To access this register, programmers need to use rdtsc instruction. Once number of cycles is available, it can be converted into time unit using following equation 4.1.

seconds = (number of cycle) / (frequency of processor)

(3.1)

In visual studio rdtsc instruction is accessed through intrin.h header file which will define assembly instruction as an intrinsic function. Intrinsic function will be treated as inline function and thus it will reduce overhead of function call resulting in better performance. If we don't use rdtsc as an intrinsic function it will also include overhead of function call.

This example code is same code which we used to illustrate clock function. So that we can compare results from both. Black boxes in code show instructions which are used to measure time using rdtsc instruction. One thing to take care is define variable which are 64 bit long, for that purpose one can use "long long int " type or "_int64 " type which is available in visual studio and it will also improve readability of code.

```
106  # include<stdio.h>
107  #include<conio.h>
108  #include<stdlib.h>
109  #include <string.h>
110  //#include <time.h>
111  #include <intrin.h>
112
113  #define MAXLINE 10
114
115  char *lineptr[MAXLINE];
116
117  void readline(char *lineptr[], int nline);
118  void writeline(char *lineptr[], int nline);
119  void swap(char *v[],int i,int j);
120  void sort(char *lineptr[],int nline);
121
122  #pragma intrinsic(__rdtsc)
123
124  int main()
125  {
126
127      int nline;
128      nline = 10;
129      _int64 start, stop;
130      int i = 65536;
131      readline(lineptr,nline);
132      start = __rdtsc();
133      sort(lineptr,nline);
134      while(--i);
135      writeline(lineptr,nline);
136      stop = __rdtsc();
137
138      printf("\n cycle %ld time %f \n",(stop - start));
139      getch();
140
141      return 0;
142  }
```

Figure 3.4: Use of rdtsc instruction in example code

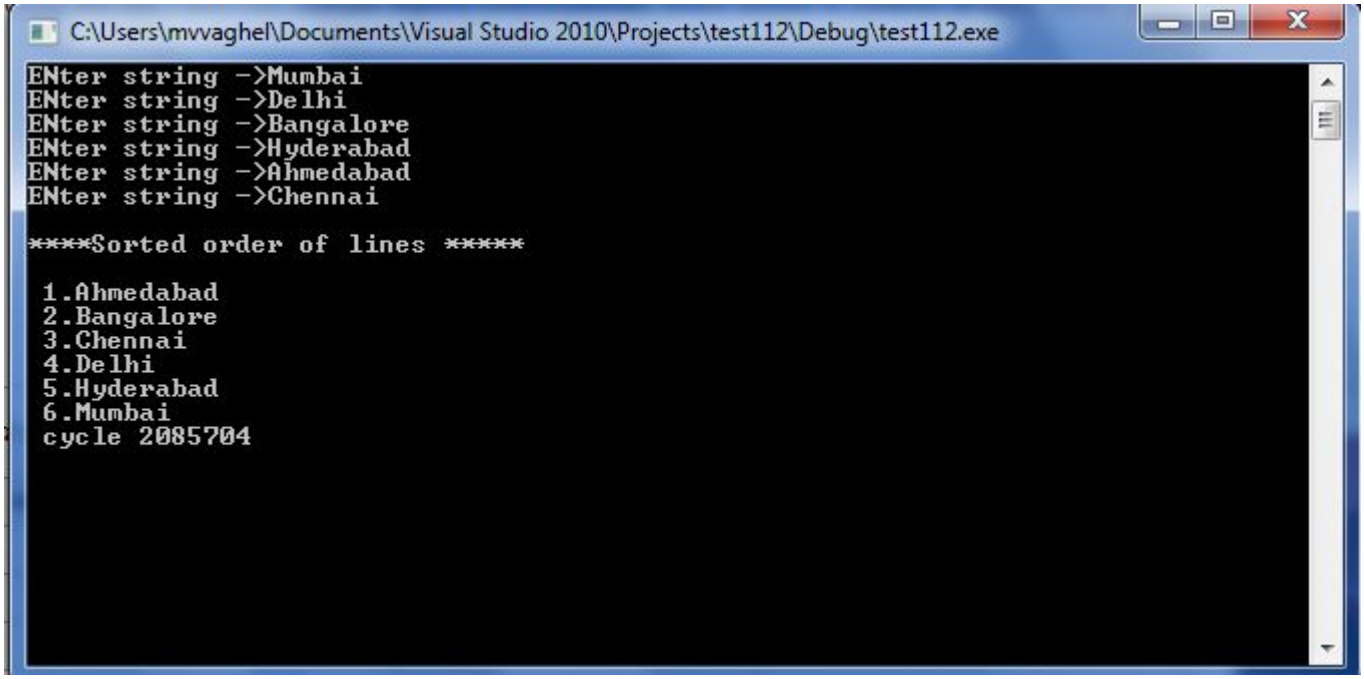


Figure 3.5: ticks required to sort 6 strings

As shown in figure 3.5. to sort 6 strings processor takes 2085704 cycles and processor is running at 2.6 Ghz frequency.

Timings can be calculated from equation 4.1.

$$time = number of cycles/Frequency$$

$$time = 2085704/(2.6 * 109)$$

$$time = 0.802194ms.$$

Thus accurate time is 0.802194 millisecond instead of 1 ms shown by clock function. We can measure same for another case where number of strings to be sorted is 10. It is shown on next page.

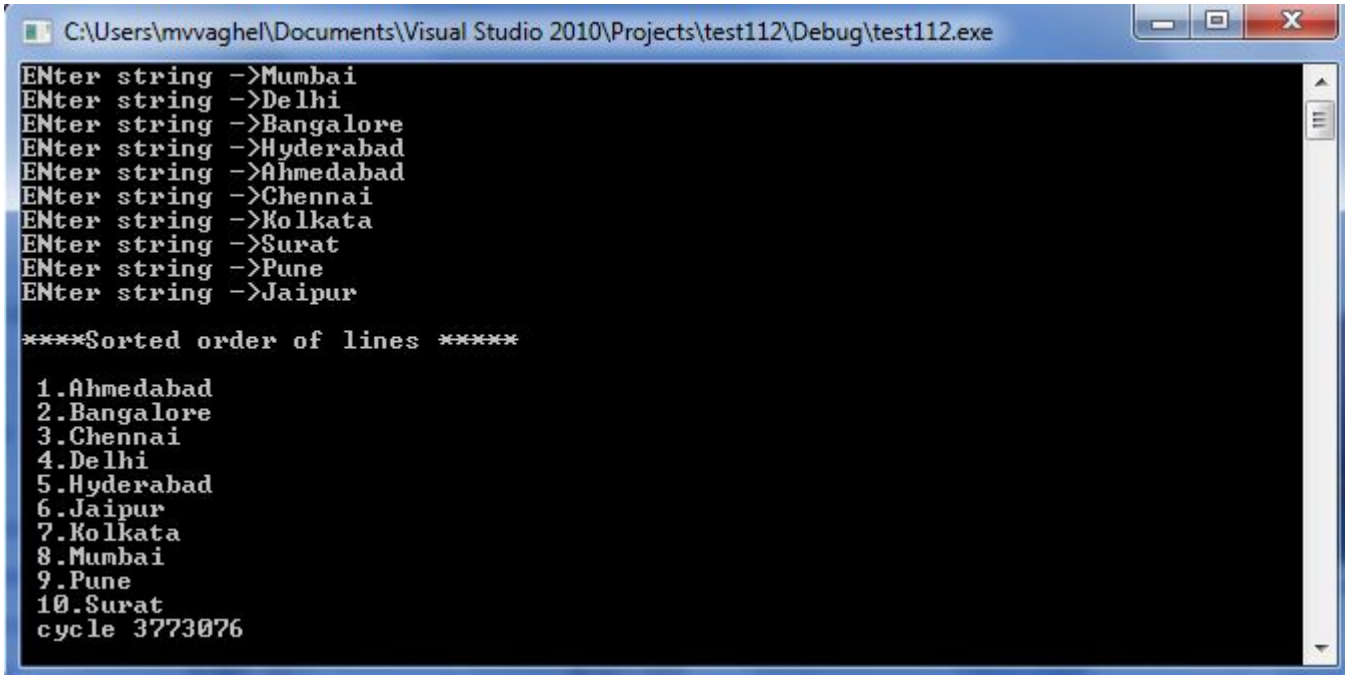


Figure 3.6: ticks required to sort 10 strings

As shown in fig.3.6 to sort 10 strings processor takes 4307844 cycles and processor is running at 2$\dot{6}$ Ghz frequency.

Timings can be calculated from equation 4.1.

$$time = numberofcycles/Frequency$$

$$time = 4307844/(2.6 * 109)$$

$$time = 1.65686307ms.$$

Thus accurate time is 1.65686307 millisecond instead of 2 ms shown by clock function.

3.3 summary

Main disadvantage of this method is that to accurately measure timing we should not allow any other process to run in between and It can also create problem on new processors which has scalable frequency because processor may have maximum frequency of 3$\dot{3}$ GHz and currently might be running at 1$\dot{6}$ GHz due to less number of processes and there is no way to determine processor frequency while code is running. For embedded systems which have processor with fixed frequency and don't have many processes running this method gives accurate resolutions.

Chapter 4

Code debugging using GDB

4.1 Introduction to code debugging

Programmers are prone to mistake and while writing code some mistakes can happen. These mistakes can introduce bug or unexpected behavior in code and while testing the code if any issue come it becomes necessary to debug that code. There are three ways to debug program:

- By putting prints at certain places where programmer feels error might occur. This method works sometime when code is small and programmer is well aware of type of error but for big projects this method would consume large amount of time.

- By looking at code and thinking what code is doing and making educated guess of what problem may be. This method is also time-consuming and programmer should be well aware of code.

- Use of debugger tool such as GDB which allows various functionalities to debug existing code while its running. This is easiest and less time consuming option and debugging of large program is simplified at great extent.

GDB is a GNU project debugger and it is very useful to see what is going on in another program while it executes. GDB can be useful for conditions such as page fault where during development of large project its difficult to find how page fault occurred. GDB also provides support for scripting thus one can automate process of debugging for certain behaviors of code, for example, during page fault someone can run script to dump some variable information to know what is going wrong. Most of the programmers prefer to debug their program by putting print at certain places and it is only suitable when code is small or programmer knows portion of code from where issue is coming but for larger application this method wont work or itll consume much time. Using debugger like GDB makes it very easy to find a problem and it can be done very quickly. GDB can also be useful in multithreading environment and when you have multiple applications.

GDB can be helpful to debug your code or an application in following ways:

- You can start your program after starting GDB, thus you may also specify that might change programs behavior. For example, you can specify some parameters that may change behavior of your code.

- One can put breakpoints at certain places in your code thus making code execution to stop on certain condition and then you can also check values of various variables in your code, making debugging code easier.

- You can examine your code for what happened when your program stops abruptly for example when page fault occurs while running code its difficult to debug code at that time using GDB you can examine code and know how page fault occurred.

- You can also change things (variables value) in your code once code stops at breakpoint thus you can experiment with different values to check code behaviors and sometimes it can also help to solve certain bugs in code.

GDB uses debugging symbols in order to debug your code and GDB can only understand debugging symbols generated by g++ or gcc. Thus while compiling your code you need to give flag to gcc to generate symbol file and you may want to debug your code later. In standard gcc program you just need to give -g flag in order to enable GDB. You should also need to have GDB installed on your system.

For example, if you have file named main.c then you can compile as follow:

```
g++ main.cpp -g -o main
```

Once you have compiled file successfully then you need to start GDB using command gdb in shell of your respective operating system. Once gdb starts it will prompt as follow:

```
(gdb)
```

After this prompt you can load file to gdb using file command for example, file main.c. If you want any help regarding any command then you can write help [command] and gdb will give description about that command.

```
(gdb) file main.c
```

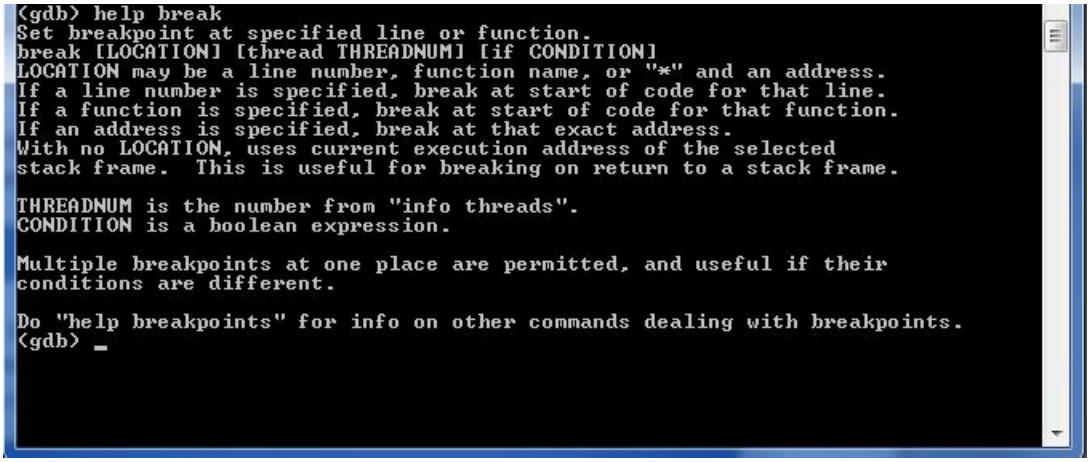


Figure 4.1: help command in GDB

Once gdb start you can just type run command and itll start execution of code and program will execute normally. If you want to debug some bug in the code and want execution of code at certain point (which is known as breakpoint) then you can put breakpoint in the code using following commands:

- Putting breakpoint at specific line
  Programmer can put breakpoint in gdb using break command and if you know the line number in file where you want to put a breakpoint then it can be done as follow:

Figure 4.2: Breakpoint at specific line

Thus above command will put breakpoint at line number 60 in file main.c

- Breakpoint at function
  GDB also support breakpoint at specific function, so if programmer wants to stop execution of code when it enters in specific function it can be done as follow.

Figure 4.3: Breakpoint at specific function

Thus GDB will put break point at function foo.

Once breakpoint is set, programmer can continue to execute program using continue command and if programmer want to execute step by step step command is useful. You can also give number of steps to be executed along with it. For example,

Figure 4.4: stepping through code

As shown in example above program will stop after executing 10 steps, here step means single ASM (assembly language) instruction not 10 C- statements. For single stepping next command can be helpful rather than using step 1.

GDB also provide feature that one can check value of variables at breakpoint. Value of a variable can be print in hex also which is shown below. More commands of GDB can be found in reference 1. Thus one can check value of variable as follow:

```
(gdb) print/x var
```

Figure 4.5: checking value of variable

GDB is most useful in scenario when page fault occurs because its never predictable and page fault is difficult to trace through prints. GDB can also give information after page fault occurs. Once page fault is encountered, just start GDB and run program without any breakpoints, once page fault occurs GDB will give exception information along with address of exception and how program reached at that point. It can be checked with backtrace command which will give output as follow:

```
(gdb) backtrace
#0 Node< int >::next (this=0x0) at main.c:28
#1 0x2a16c in LinkedList< int >::remove (this=0x40160,
 item_to_remove=@0xffbef014) at main.c:77
#2 0x1ad10 in main (argc=1, argv=0xffbef0a4) at main.c:111
(gdb)
```

Thus as can be seen from above this is a null pointer and at line number 28 we tried to dereference null pointer thats why page-fault occurs and line after that indicates that indicates that from which function it was called so programmer can analyze the values at function calling this line.

4.2 summary

GDB is very useful open source debugging tool which provides various functionalities to make debugging of large and complex code easier. GDB can reduce debugging time to great extent and can provide accurate insight of code. However there is some limitation of GDB, which comes due to compiler optimization. When code is compiled, compiler will try to optimize the code in a way that arguments which are never used will be taken away, sometimes if a looping variable uses same value then itll be kept in CPU register instead of stack. In these cases GDB will show variable as an optimized output and thus wont be able to show value of variable. In this situation its better to turn off compiler optimization to debug the code.

Chapter 5

Secure coding

Code written by a programmer can be vulnerable to hackers and it is possible that due to this vulnerabilities code will be expose to vulnerable entities. Goal of secure coding is to reduce amount of vulnerabilities in code to the level that can be mitigated to operational environment. This can be only accomplished by discovering and improving security flaws into code while developing code or testing your code. Most of the time programmers make common mistakes in the code such as not checking NULL pointer while allocating memory, not checking array boundaries while accessing array, not freeing memory once its use is completed etc. For this purpose some companies have developed a tool for static analysis of source code which can detect these common mistakes but these tools are not much reliable, they work in most of cases but in some cases programmer needs to think about security while development code and have to put appropriate checks to handle exceptions. In this chapter well take a look at what are common mistakes programmers make and how that can make code vulnerable. In second part use of one of the static analysis tool Klocwork is explained.

5.1 Common vulnerabilities in code

Following are common mistakes made by software developers while developing a code. These are explained in this chapter with detail.

1. Validation of array index.
2. Improper null termination of strings.
3. Temporary files.
4. Releasing memory improperly.
5. Using uninitialized variables.
6. Pointer dereferences.
7. Exposure of system data to control sphere.

1. **Validation of array index.**

In most of the C or C++ code there is no automatic boundary check performed while accessing an array. This problem occurs most commonly while using loop index in order to access an array. If the end condition of for loop has some flaws then index may go out of boundary of arrays and can cause out of bound array access. Index out of bound error can also occur if no index checking is done and when request comes for array index which is out of bound. Even sometimes validation is performed it can also be wrong sometimes for example if we only check index for upper bound then index can be violated on lower bound. Improper validation of array index leads out of bound access which can cause:

- Diminished availability

  In most of the cases application will crash while accessing outside of protected memory area.

- Loss of data integrity

  It is possible to write data out of an array boundary. This can be harmful to data which is maintained by an application. For example, an application maintains two array for student ID and mark of a student. Both array are contiguous i.e. both are placed together, if student IDs are written out of bound then it may change data of their marks.

- Sensitivity information leakage

  It is possible to write data using array index which is out of boundary. It can expose sensitive data to unauthorized entity.

- Alteration in programming logic

  Somewhere memory can be accessed and controlled an attacker may be able to execute arbitrary code. This may cause change in programming logic.

```
1  int main()
2  {
3      int x, a[20];
4      a[12] = 3;       // Writing within bound.
5      a[22] = 5;       // Writing out of bound , No error.
6      a[35] = 10;
7
8      a[10000] = 100;     // Error.
9  }
```

Figure 5.1: Example code showing validation of an array index

Here indices 22 and 35 are out of bound but it wont give any error while accessing it because memory has been pre-allocated by an application on stack and we are still in memory boundary which is much larger than array size. However when one tries to write index 10000, it may give segmentation fault since application tries to write out of allocated memory. Thus this out of range behavior is undefined.

Out of bound scenario occurs because sometimes no validation is performed. For example, when programmer wants to access lookup table based on users input and he has not put index checks then it may cause it to read out of bound or sometimes application may crash. Another common mistake is improper validation and most common mistake is that programmer checks for array size less than or equal as shown below.

```
11  int len = 10;
12  int a[len];
13
14  int get_value(int index)
15  {
16      int value = 0;
17
18      if(value <= len)
19          value = a[len];
20
21      else{
22          printf("Array index out of bound \n");
23          value = -1;             // Error
24      }
25
26      return value;
27  }
```

Figure 5.2: Improper validation of an array index

There are 2 mistakes in above code. Caller of function may send negative index, so programmer needs to check for lower bound as well and another mistake is to check for array bound and array bound is less than maximum length. For example, when length is 10, array indices are 0 to 9 so array index 10 will be out of bound. Thus it should be $<$ instead $<=$.

Thus programmer needs to take care of following things:

- Understand range of loop variable.
- Check both upper and lower boundary of array.
- Be careful around array boundaries.

2. **Improper null termination of strings**

Problem with null termination will occur when string or an array of strings is not terminated properly. Improper null termination may occur due to forgetting null termination, one may place null termination at wrong place by mistake or null termination may be overwritten by mistake. Problem with null termination is that these kinds of mistakes are difficult to spot manually and they can also cause loss of data integrity, diminished availability, and leakage of sensitive information. These mistakes can be easily identified by static analysis tool.

In C language string related functions depends heavily on null termination for various operation for example, in order to find length of string function will start from first character and will count characters till it encounters null character. One

basic rule is to have a null termination before or at null termination of a string. Sometimes when copying data from source from destination, if we try to write data with size of function and if null termination is not proper then it could lead to vulnerabilities.

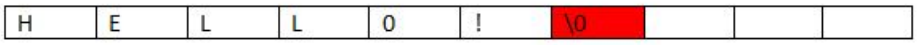


Figure 5.3: Null termination in strings

As shown an array above, if null termination is omitted and then strlen function is called to calculate length of a string then it will return wring string length because function will count until it encounters null character in memory. If function does not encounter null termination and if tries to read out of memory bound of an application then it can cause segmentation fault to occur. Sometime if code has loop which only breaks when it encounters null terminating character then improper termination can cause program to go into infinite loop.

One another example, lets say programmer develops application which reads from file and copies file into buffer. Buffer is allocated for lets say 2KB memory. If someone can alter content of an input file and remove null termination then strcpy function will continue to write until it finds arbitrary null character, thus attacker can control data after buffer and program is vulnerable to buffer overflow attack.

Care to be taken while manipulating strings:

- Check boundaries if the code is accessing the buffer in a loop
- Truncate input data to a maximum allowable size
- Be wary of off-by-one errors

3. **Temporary files**

Temporary files are files which have been allocated outside memory of a process. Temporary files are easy mechanism which can be used to provide communication between processes or can be used to transform data within single process. Programmer needs to realize that temporary files are entity which are outside process but contains important data for process, so they need to protect data in temporary storage. Problem occurs because programmer creates temporary files in predictable location with predictable names without proper privileges and this makes it vulnerable to attackers.

Use of temporary file in an insecure manner can result in one of following.

- Data modification
  If one application writes data in temporary storage which will be read

by another application later, attacker may be able to change the data and can risk data integrity and may be vulnerable for second application.

- Application crash
  Once attacker modifies data from temporary storage it can cause application to behave in undesirable way and sometime it may crash.

- Leakage of sensitive information
  If one application writes sensitive information to temporary storage file then attacker can get hold of sensitive information.

Temporary files are common entry point for an attacker and application which maintains large data base benefits heavily from temporary files. Sometimes source code analysis tool can help quickly to uncover problems.

Care needs to be taken by programmer while using temporary files:

- Ensure file name is randomly given.
- Set sufficient permissions on file and location
- Ensure file does not already exist
- Do not store sensitive data into temporary files.

4. **Releasing memory improperly**

Releasing allocated memory improperly leads to term known as memory leak. Depending upon relative size of memory that is never released leak is said to be large or small. Larger leaks are easier to detect because they can adversely affect system easily and thus it can be easily identified but smaller leaks are difficult to track and when it gets add up it becomes larger leak and system dont have any memory to be allocated. At the end application will crash because it doesnt have sufficient memory.

Memory leaks can occur due to memory is allocated but never freed. This is most common cause of memory leak. For example, code frees memory at the end of a set operation but error condition and exception can cause DE allocation part to be skipped and it leads to memory leak. When allocated memory is passed among functions which are written by different developers, there may be confusion about which part of the program is responsible for freeing the memory and complex data branching elements, such as switch and if-else statements, can cause the variables storing the memory allocations to go out of scope, making them irrecoverable.

```
29  void foo(int a)
30  {
31  int *ptr1 = malloc(30);
32  int *ptr2 = malloc(40);
33  int *data_ptr = NULL;
34
35      if(a <30)
36          data_ptr = ptr1;
37      else if(a < 40)
38          data_ptr = ptr2;
39      else{
40          printf("not suffiecient memory \n");
41          return;
42      }
43
44      free(data_ptr);
45
46  }
```

Figure 5.4: Improper memory release

In above code we have 2 buffers and each will be used according to incoming data size. There are 2 mistakes in code. First is if data size is more than maximum buffer size then we return without freeing memory and another is even if we free one buffer we are not freeing another buffer. This can lead to possible memory leak.

Care needs to be taken by programmer:

- Avoid freeing the same memory across different functions.
- Ensure that exception handlers do not cause leakage.
- Error checking code should always de-allocate before returning.
- Always free all memory allocated by code.

5. **Using uninitialized variables**

The use of uninitialized variable is vulnerable because variables may contain random value which may be invalid, may contain value of previously allocated memory. This problem has more impact when programmer depends upon external agent to initialize variable for you. This can happen when initialization code is called only when invoked by an external agent.

Uninitialized variables can cause application to enter in an unexpected state because it may lead to miscalculated output or logical error. In many cases attacker may have been able to execute code that uses uninitialized memory segments then random data can lead to program crash. These can also affect confidentiality if attacker manages to read certain segments of memory which is used previously. Sometimes compiler also gives warning for uninitialized memory and static analysis tools are also helpful for that. One rule of thumb is to always initialize variables with zero and pointers will null value until allocated some meaningful value. It is better to use calloc instead of malloc to allocate memory because calloc also initialize memory contents with all zeros.

6. **Pointer dereferences**

   In C null pointer doesnt point to any memory and null pointer dereference can cause program to stop abruptly and also cause runtime errors. It may be due to ineffective error handling, race conditions and not proper checking before using pointers. One rule is to always check for null pointer before dereferencing pointer. In most of the cases null pointer error affects availability of an application.

   Care needs to be taken by programmers:

   - Check pointers for null before using them.
   - Check the return results of all functions because some function may return null.
   - Beware of race conditions.
   - Use source code analysis tools to find null pointer issues.

7. **Exposure of system data to control sphere**

   System data is exposed to untrusted entities due to one of the following reasons:

   - No filtration of error information sent to the user
   - Non-generic and over-informative error messages
   - Unhandled exception cases

   It can mainly affect confidentiality of organization, sometimes it can also lead to more serious attacks and these errors are very difficult to spot manually but static analysis tool can be of more help.

   Error information such as stack traces contain function call hierarchy information as well as the number and type of parameters those functions requires. In some cases, actual parameter values may also be included in stack trace information. This reveals application implementation information to the attacker, which can be leveraged to conduct other attacks against the system.

   Care needs to be taken by a programmer:

   - Ensure that all error cases are handled
   - Suppress backend error messages
   - Use generic error messages

5.2 Introduction to klocwork

Klocwork is company that provides tool for static code analysis which can detect this common programming mistakes and can help organizations to develop secure and reliable code thus by extending life cycle of complex software
There are two tools from klocwork.

- **Klocwork Insights**

  Klocwork insight is static analysis tool for analysis of c, java and c++ code. It also provides different plug-ins for code analysis. It also features on the fly analysis of code in visual studio. It is same as spelling checking in Microsoft word.

- **Klocwork Cahoots**

  Klocwork Cahoots is a tool for code review. It makes process of reviewing code easy. After improving code one can put it for review online using this tool and reviewer can review code, insert comments and author can add reviewer.

Klocwork Insight has two versions. One is for server and another is desktop client. Thus with desktop client developer can analyze code locally and can prevent new errors from being introduced to project.Server version can be useful to maintain code and run analysis periodically to know that whether new bugs have been introduced or not.

5.3 Example of use of klocwork

To give simple example of how Klocwork works, we will take example of same program that takes strings from user and will sort in ascending order. Code is given in chapter 4 but for reference it is also given below.

```c
# include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include <string.h>


#define MAXLINE 10

char *lineptr[MAXLINE];

void readline(char *lineptr[], int nline);
void writeline(char *lineptr[], int nline);
void swap(char *v[],int i,int j);
void sort(char *lineptr[],int nline);

int main()
{

    int nline;
    nline =3;

    readline(lineptr,nline);
    sort(lineptr,nline);
    writeline(lineptr,nline);
    getch();

    return 0;
}

void readline(char *lineptr[], int nline)
{
  int i;

  for(i=0;i<nline;i++)
  {
       char *p = (char *)malloc(50 *sizeof(char));
      printf("ENter string ->");
       gets(p);
      lineptr[i]=p;

  }
}
```

Figure 5.5: Example code(part1)

```
51
52  void writeline(char *lineptr[],int nline)
53  {
54    int i;
55    printf("\n****Sorted order of lines *****\n");
56
57    for(i=0;i<nline;i++)
58    {
59       printf("\n %d.%s ",i+1,lineptr[i]);
60
61    }
62  }
63
64  void sort(char *lineptr[], int nline)
65  {
66    int i,j;
67
68
69    for(i=0;i<nline;i++)
70    {   for(j=0;j<nline-1;j++)
71        {
72            if(strcmp(lineptr[j],lineptr[j+1])>0)
73                swap(lineptr,j,j+1);
74
75        }
76    }
77
78  }
79
80  void swap(char *s[] ,int i,int j)
81  {
82       char *temp;
83       temp = s[i];
84       s[i]=s[j];
85       s[j]=temp;
86
87  }
88
```

Figure 5.6: Example code(part2)

Steps :

It is assumed that Klocwork Insight desktop client is installed in your system.

- Once code is saved as a c file open command prompt and goto directory of saved code.
- Now create a new project using Klocwork desktop client. Just click new project .

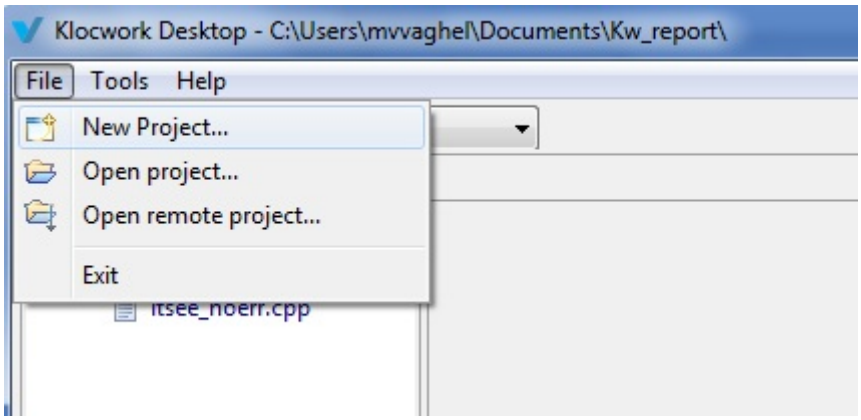


Figure 5.7: Creating new project in klocwork

- Once new project is selected following window will open. select stand-alone project and give location of saved c file at project location.

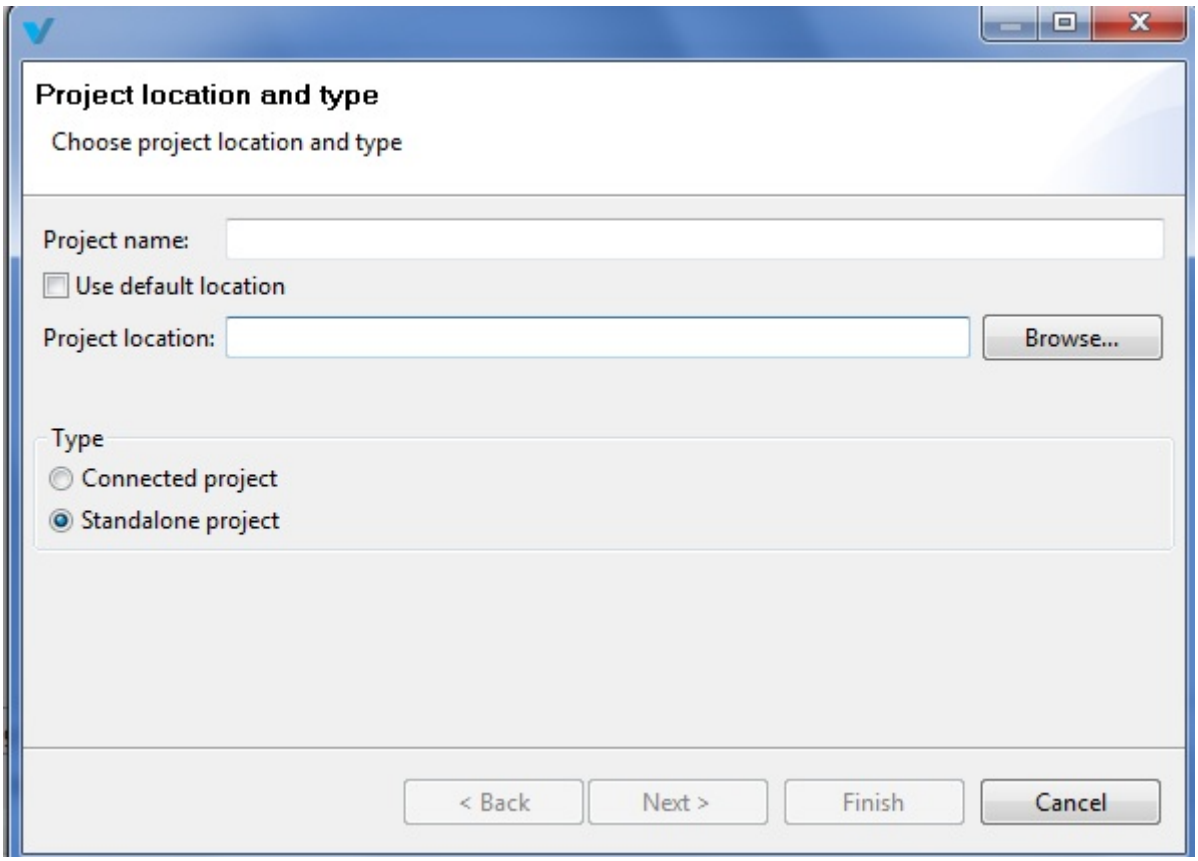


Figure 5.8: Selecting location ad project type

- Once project is created from command prompt write command “kwshell ” as shown below in PowerShell

Figure 5.9: Opening kwshell

- After entering to Klocwork shell compile your code with command “gcc filename.c ”

- After this step Klocwork desktop client will automatically show different issues in code. If detailed report in text file is required then use following command
  kwcheck run F detailed report report_nametxt

- It will compile the code , analyze the code and will generate detailed report in same directory.

Figure 5.10: Generating detailed report

- For our example code report generated is as below:

- Thus there are total 2 issues at line 46 and detailed report also explains how that issue affects the code. One can choose other report style which can generate report in Excel format.

---

1. (Local)C:\Users\mvvaghel\Documents\c_files\ltsee.cpp:46
SV.UNBOUND_STRING_INPUT.FUNC (1:Critical) Analyze
Do not use 'gets', this function does not check buffer boundaries and is very likely to cause buffer overflow vulnerabilities
Current status 'Analyze'

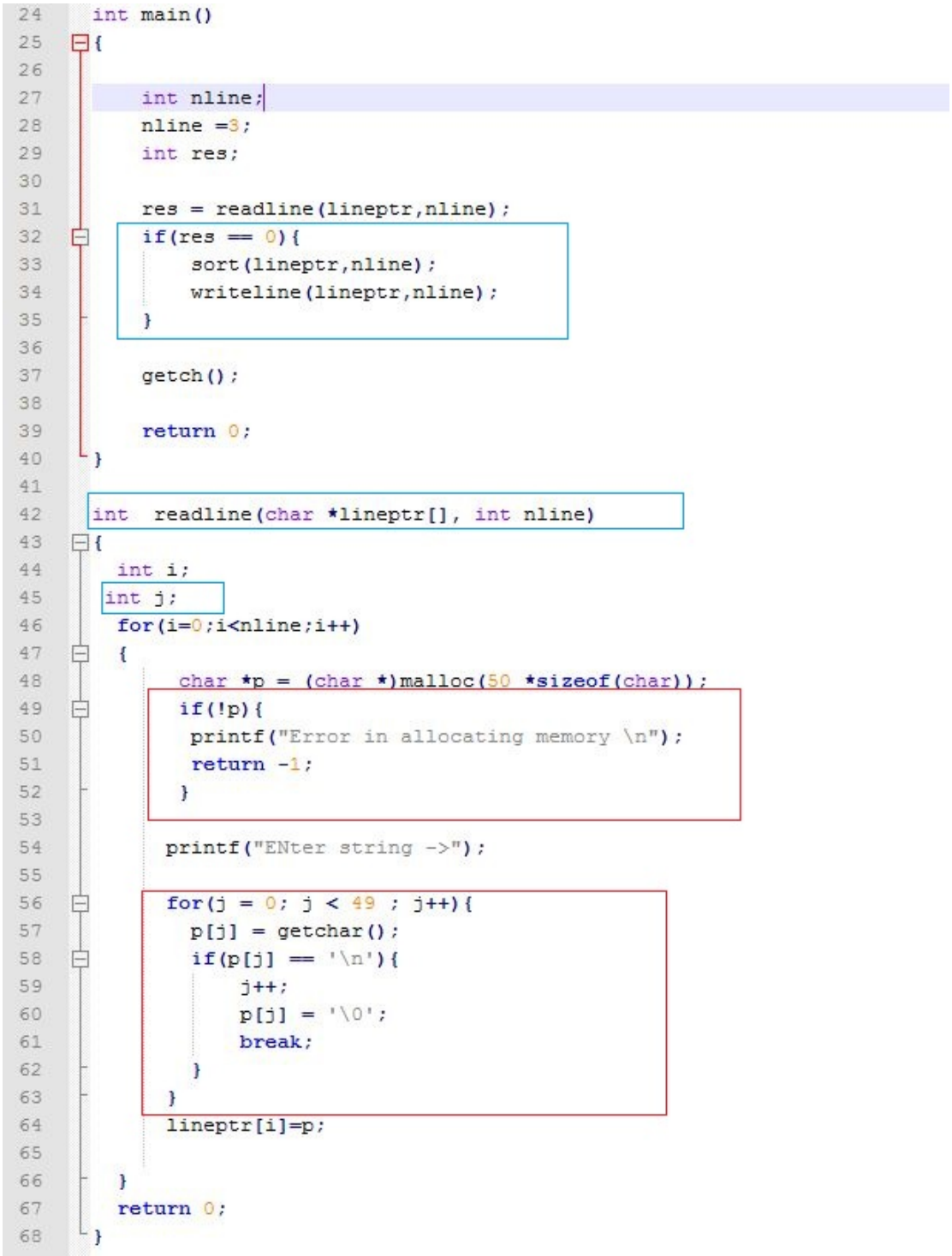
---

2 (Local) C:\Users\mvvaghel\Documents\c_files\ltsee.cpp:46 NPD.FUNC.MUST (1:Critical) Analyze
Pointer 'p' returned from call to function 'malloc' at line 44 may be NULL and will be dereferenced at line 46.
** ltsee.cpp:42: Continue loop iteration, while i<nline is true*
** ltsee.cpp:44: 'p' is assigned the return value from function 'malloc'.*
** ltsee.cpp:46: 'p' is dereferenced by passing argument 1 to function 'gets'.*

Current status 'Analyze'
Summary: 2 Local
2 Total Issue(s)

Figure 5.11: Detailed report

- Thus there is error at line 44 that we are not checking for null pointer, which is most common programming error and at line 46 we should not use 'gets' function because input can be more than 50 characters and 'gets' function will not check boundary. Thus we need to modified code to resolve two issues.
- Modified code is shown on next page .Red box shows code added/modified to resolve two errors and code with blue box is additional code for error checking when malloc returns null pointer.



```
24  int main()
25  {
26
27      int nline;
28      nline =3;
29      int res;
30
31      res = readline(lineptr,nline);
32      if(res == 0){
33          sort(lineptr,nline);
34          writeline(lineptr,nline);
35      }
36
37      getch();
38
39      return 0;
40  }
41
42  int  readline(char *lineptr[], int nline)
43  {
44    int i;
45    int j;
46    for(i=0;i<nline;i++)
47    {
48        char *p = (char *)malloc(50 *sizeof(char));
49        if(!p){
50         printf("Error in allocating memory \n");
51         return -1;
52        }
53
54        printf("ENter string ->");
55
56        for(j = 0; j < 49 ; j++){
57          p[j] = getchar();
58          if(p[j] == '\n'){
59              j++;
60              p[j] = '\0';
61              break;
62          }
63        }
64        lineptr[i]=p;
65
66    }
67    return 0;
68  }
```

Figure 5.12: Modified code to remove KW error

- Now running Klocwork analysis again on modified code using same step gives following report

  Summary: 0 Local
  0 Total Issue(s).

Thus Both issues are resolved.

5.4 summary

While writing code, programmer make some mistakes unknowingly and it makes code vulnerable to hackers.In this chapter reason for vulnerabilities are explained and how it can harm your application is explained. After that static analysis tool Klocwork is introduced.Klocwork is very powerful static analysis tool and it's also widely used in many industries. Klocwork's limitation is that it can do static analysis of tool only. Let's take a simple example of two functions. First function does some calculations and calls another function for getting pointer of data buffer. Second function might return null pointer if it can't find buffer. Now Klocwork will only give error if second function explicitly returns null. Second example would be when programmer pass pointer as input argument and someone sends that pointer as null, Klocwork doesn't give error there. Although Klocwork also provides several options(checkers), which you can enable to analyze different aspect of code. Thus conclusion of this chapter is that programmers can use Klocwork for analysis of their code to remove vulnerabilities upto some extent but care must be taken while writing a code and programmer should also aware of different vulnerabilities that can occur and how to resolve them which will lead to more secure code and applications.

Chapter 6

Conclusion and future scope

Electronics industry is moving towards making device smarter and sensors will play an important role in that. Devices need to continuously sense it?s surrounding environment and makes decision based on that. Devices are also becoming very small in size and thus have less battery power and embedded systems need to have a better battery life. For smart devices this purpose can be achieved by use of a sensor hub which is a low power processor acting as a co-processor for a main processor. It reduces power consumed by main processor and will continuously sense the sensor data while main processor is in sleep mode.

For sensor hub to run independently of main processor, sensor hub needs its own firmware which runs and collects data. Firmware contains various components like sensor drivers, host interface (Interface with an operating system), sensor management, algorithms etc. Sensor driver plays an important role here and it provides interface with sensor hardware. Sensor drivers are same as device drivers in Linux but it?s better to use micro-drivers which is device driver in user space rather than integrating it with kernel. It makes programming, debugging and integration of driver pretty easy because in user space many tools are available for debugging and it?s relatively easy compared to kernel. Host interface provides an interface with an operating system and it is useful to receive commands from host and send data as per requirement. Sensor management manages sensor?s properties like frequency, range, mode etc. and it also provides necessary arbitration. Algorithms are useful to take sensor?s data and make useful decision based on data.

While developing firmware, due to some small mistakes of a programmer firmware may not give desired functionality and in this case one need to debug certain problems. Most powerful and open source debugging tools is GDB. With use of this tool, certain issues have been resolved successfully and quickly. Certain functionalities of GDB like single stepping, breakpoint and stack-trace are very much useful for embedded programmer.GDB can even debug multi-threaded applications.

For real time embedded systems security is also important. Some common mistakes in code can make system vulnerable to hackers. Some mistakes can be identified by programmer while writing a code and some are difficult to catch. Static code analysis tools are very helpful to trace this kind of issues. Klocwork is a tool for static analysis and it provides accurate analysis of source code but if there is mistake in logic, Klocwork doesn?t help much.

Future scope of project includes adding more functionality to firmware such as power management algorithms which takes decision based on data of various sensors and put whole system into sleep mode. Another future scope is to support multiple operating systems as currently it only supports windows.

Chapter 7

References

1. Accelerometer LSM303DLHC datasheet from ST-Microelectronics : http://www.st.com/web/en/resource/technical/document/datasheet/DM00027543.pdf

2. Gyroscope L3GD20 datasheet from ST-Microelectronics : http://www.pololu.com/file/0J563/L3GD20.pdf

3. Magnetometer LSM303D datasheet from ST-Microelectronics : http://www.st.com/web/en/resource/technical/document/datasheet/DM00057547.pdf

4. "Dynamic Configuration of Sensors Using Mobile Sensor Hub in Internet of Things Paradigm "by Charith Perera, Prem Jayaraman, Arkady Zaslavsky,Peter Christen, Dimitrios Georgakopoulos, Intelligent Sensors, Sensor Networks and Information Processing, 2013 IEEE Eighth International Conference

5. Yu, Meng-Chieh, Tong Yu, C. J. Lin, and E. Y. Chang."Low power and low cost sensor hub for transportation-mode detection." Studio Engineering, HTC, Tech. Rep (2013).

6. Ganapathy, Vinod, Matthew J. Renzelmann, Arini Balakrishnan, Michael M. Swift, and Somesh Jha. "The design and implementation of microdrivers." ACM SIGOPS Operating Systems Review 42, no. 2 (2008): 168-178.

7. "Linux Device Drivers" book by Jonathan Corbet and Alessandro Rubini published by OReily.

8. Microsofts guide for developing sensor drivers: http://msdn.microsoft.com/en-us/library/windows/hardware/ff545810(v=vs.85).aspx

9. RDTSC developer reference from Intel : http://software.intel.com/en-us/node/392533?language=ruwapkw=rdtsc

10. GDB documentation by GNU organization.
http://www.gnu.org/software/gdb/documentation/

11. GNU GDB commands cheat sheet
http://www.yolinux.com/TUTORIALS/GDB-Commands.html

12. “Secure coding boot camp from Klocwork”

Appendix

In this appendix some background is given about I2C drivers in Linus kernel. How to find I2C adapter in Linux operating system is explained first and then how to use various methods to write or read over I2C line is explained. Some open source libraries are already available for I2C device in Linux, SMBus is one of the famous library which provides standard APIs(functions) to interact with device over I2C bus.

Generally I2C functions reside in kernel space but its possible to access I2C adapter from user-space also through /dev interface. You need to load module i2c-dev for this. Each registered i2c adapter gets a number from 0. You can examine /sys/class/i2c-dev/ to see what number corresponds to which adapter. Alternatively, you can run "i2cdetect -l" to obtain a formatted list of all i2c adapters present on your system at a given time. i2cdetect is command which is present in I2C tools provided in Linux.

Linux treats every device as a file only and I2C device files are character device files. Linux devices contain major and minor numbers. For I2C device major number is 89 and minor number depends upon adapter number which is assigned from 0. Linux provides provision for 256 I2C devices connected to it, thus minor number varies from 0 to 255.

In order to use standard Linux APIs in your code, you need to include header file which contains definition for these APIs. One needs to include linux/i2c-dev.h file for APIs. Here precaution is needed because there are two i2c-dev.h file, one for kernel mode program which provides direct call to the APIs and another is for code in user space thus providing interaction between user and kernel space. Once you have included file in user space then next step is to determine the adapter for I2C adapter as discussed above. Once you get adapter number, next step is to open a device file:

```
int file;
int adapter_nr = 2;         /* dynamically determined */
char filename[20];
/* snprintf stores formatted string output in filename variable. */
snprintf(filename, 19, "/dev/i2c-%d", adapter_nr);

/*Open a file with name specified in read-write mode*/
file = open(filename, O_RDWR);
if (file < 0) {
/* ERROR HANDLING; you can check errno to see what went wrong */
exit(1);
}
```

Figure 1: Opening a device in Linux

Once device is opened by user you need to specify with which address you want to communicate with device, it means you need to specify slave address of a device. For example, if accelerometer LSM303DLHC chip is connected with I2C and you want to communicate with it then give slave address as 0x19. Ioctl function is used to set I2C slave address.

```
int addr = 0x19; /* The I2C address of accel lsm303dlhc */
if (ioctl(file, I2C_SLAVE, addr) < 0) {
/* ERROR HANDLING; you can check errno to see what went wrong */
exit(1);
}
```

Figure 2: set I2C slave address to accel's I2C address

Once slave address is set one can use SMBus APIs or plain I2C APIs to communicate with device. SMBus is more preferred if device supports them because there is slight difference between SMBus and I2C. Both are illustrated below:

```
__u8 register = 0x0A; /* Device register to access */
__s32 res;
char buf[10];
/* Using SMBus commands */
res = i2c_smbus_read_word_data(file, register);
if (res < 0) {
    /* ERROR HANDLING: i2c transaction failed */
} else {
    /* res contains the read word */
}
/* Using I2C Write, equivalent of
i2c_smbus_write_word_data(file, register, 0x6543) */
buf[0] = register;
buf[1] = 0x43;
buf[2] = 0x65;
if (write(file, buf, 3) ! =3) {
    /* ERROR HANDLING: i2c transaction failed */
}
/* Using I2C Read, equivalent of i2c_smbus_read_byte(file) */
if (read(file, buf, 1) != 1) {
    /* ERROR HANDLING: i2c transaction failed */
} else {
    /* buf[0] contains the read byte */
}
```

Figure 3: set I2C slave address to accel's I2C address

As shown in example above both are almost similar except that write or read API supports how many bytes to be read or written where SMBus reads fixed bytes of data. Following SMBus functions can be used to do read-write over I2C bus:

- s32 i2c_smbus_write_quick(int file, u8 value);
- s32 i2c_smbus_read_byte(int file);
- s32 i2c_smbus_write_byte(int file, u8 value);

- s32 i2c_smbus_read_byte_data(int file, u8 command);
- s32 i2c_smbus_write_byte_data(int file, u8 command, u8 value);
- s32 i2c_smbus_read_word_data(int file, u8 command);
- s32 i2c_smbus_write_word_data(int file, u8 command, u16 value);
- s32 i2c_smbus_process_call(int file, u8 command, u16 value);
- s32 i2c_smbus_read_block_data(int file, u8 command, u8 *values);
- s32 i2c_smbus_write_block_data(int file, u8 command, u8 length,u8 *values);

One advantage of using SMBus over simple read and write is that SMBus can do combined read and write in a single call and stores data into buffer with read or write flag so by checking the flags programmer can know whether data was read or written and if any error occurs during I2C transactions then above all functions returns -1 value which indicates I2C error occurred. Codes for accelerometer driver LSM303DLHC is given on next page.

```
/*
 * main.c
 *
 *  Created on: Apr 12, 2014
 *      Author: mvvaghel
 */

#include <linux/i2c-dev.h>
#include <stdio.h>
#include <conio.h>
#include "accel_header.h"

static   int file;  // File descriptor.
void accel_init()
{
        unsigned char ctrl1 = ACCEL_XYZ_ENABLE | ACCEL_LOW_POWER_ENABLE;
        unsigned char ctrl2 = ACCEL_HPF_NORMAL_MODE;
        unsigned char ctrl4 = ACCEL_BDU_ENABLE | HIGH_RES_OP_ENABLE;

        // 1 is adapter which may vary depending upon system.
        char file_name[20] = "/dev/i2c-1";

        file = open(file_name, O_RDWR);
        if(file < 0){
                printf("failed to open I2C adapter \n");
                return;
        }
        /* The I2C slave address of accelerometer */
         int addr = ACCEL_SLAVE_ADDR;

         if (ioctl(file, I2C_SLAVE, addr) < 0) {
                printf("failed to configure I2C adapter \n");
                return;
          }

        // Write 0x0F at address of control reg. 1.
        i2c_smbus_write_byte(file , ctrl_reg1 , ctrl1 );

        // Write 0x80 at address of control reg. 2.
        i2c_smbus_write_byte(file , ctrl_reg2 , ctrl2 );

        // Write 0x88 at address of control reg. 4.
        i2c_smbus_write_byte(file , ctrl_reg4 , ctrl4 );
}
```

```
int find_freq_bit(int freq)
{
	int freq_bit = 2;				// Default frequency : 10 Hz.
	switch(freq){

		case 0:
			freq_bit = 0;
			break;

		case 1:
			freq_bit = ACCEL_FREQ_1_HZ;
			break;

		case 10:
			freq_bit = ACCEL_FREQ_10_HZ;
			break;

		case 25:
			freq_bit = ACCEL_FREQ_25_HZ;
			break;

		case 50:
			freq_bit = ACCEL_FREQ_50_HZ;
			break;

		case 100:
			freq_bit = ACCEL_FREQ_100_HZ;
			break;

		case 200:
			freq_bit = ACCEL_FREQ_200_HZ;
			break;

		case 400:
			freq_bit = ACCEL_FREQ_400_HZ;
			break;

		default:
			printf("frequency not supported \n");
			break;
	}
	return freq_bit;
}
```

```
int find_scale_bit(int scale)
{
        int scale_bit = 0;                              // default range: 2G
        switch(scale){
                case 2:
                        scale_bit = ACCEL_RANGE_2G;
                        break;
                case 4:
                        scale_bit = ACCEL_RANGE_4G;
                        break;
                case 8:
                        scale_bit = ACCEL_RANGE_8G;
                        break;
                case 16:
                        scale_bit = ACCEL_RANGE_16G;
                        break;
        }
        return scale_bit;
}

void accel_config(int freq, int scale)
{
        int freq_bit = 1, scale_bit = 0;   // Default freq: 1Hz, scale = 2G.
        int tmp_ctrl_reg1 = 0;
        int tmp_ctrl_reg4 = 0;

        freq_bit = find_freq_bit(freq);
        scale_bit = find_scale_bit(scale);

        //Read control register 1.
        tmp_ctrl_reg1 = i2c_smbus_read_byte(file, ctrl_reg1);
        // Flush previous frequency value.
        tmp_ctrl_reg1 &= ~(ACCEL_FREQ_MASK);

        tmp_ctrl_reg1 |= freq_bit;
        i2c_smbus_write_byte(file, tmp_ctrl_reg1, ctrl_reg1);
        // Read control register 4
        tmp_ctrl_reg4 = i2c_smbus_read_byte(file, ctrl_reg4);
        // Flush previous range value
        tmp_ctrl_reg4 &= ~(ACCEL_RANGE_MASK);

        tmp_ctrl_reg4 |= scale_bit;
        i2c_smbus_write_byte(file, tmp_ctrl_reg4, ctrl_reg4);

}
```

```
void accel_read_data()
{
	int status = 0;
	int output[3] = {0};
	status = i2c_smbus_read_byte(file , STATUS_REG);

	/* If all axis data ready then and then read data*/
	if( status & ACCEL_STATUS_ZYXDA){

		output[0] = i2c_smbus_read_word_data(file, OUT_X_L);
		output[1] = i2c_smbus_read_word_data(file, OUT_Y_L);
		output[2] = i2c_smbus_read_word_data(file, OUT_Z_L);
	}
}


int main(void)
{
	accel_init();
	accel_config(100, 4);		// Set 100 Hz freq and 4G range.

	while(1){
		accel_read_data();
	}
}
```

```
/*
 * accel_header.h
 *    Created on: Apr 12, 2014
 *     Author: mvvaghel
 */

/*Header guards*/
#ifndef ACCEL_HEADER_H_
#define ACCEL_HEADER_H_

#define ACCEL_SLAVE_ADDR          0x19
/**
 *       CTRL_REG_1 - Configure power mode, output data rate and axes enables
 *
 *       Bits                Bit_Name                Function
 *       7-4                   ODR <3:0>          Data Rate Selection
 *       3                     LPen               Power Mode Selection
 *       2                     Zen                Z axis enable
 *       1                     Yen                Y axis enable
 *       0                     Xen                X axis enable
 *
 *       Default Value : 0000 0111
 */

#define ctrl_reg1                         0x20

#define ACCEL_MAX_FREQ                    1344
#define ACCEL_POWER_DOWN_MODE             0
#define ACCEL_FREQ_1_HZ                   1
#define ACCEL_FREQ_10_HZ                  2
#define ACCEL_FREQ_25_HZ                  3
#define ACCEL_FREQ_50_HZ                  4
#define ACCEL_FREQ_100_HZ                 5
#define ACCEL_FREQ_200_HZ                 6
#define ACCEL_FREQ_400_HZ                 7
#define ACCEL_FREQ_1344_HZ                9
#define ACCEL_FREQ_OFFSET                 4
#define ACCEL_FREQ_MASK                   (0x0F << ACCEL_FREQ_OFFSET)
```

```
#define ACCEL_LOW_POWER_ENABLE        (0x01 << 3)

/* Accel Axis Enables */
#define ACCEL_Z_ENABLE            (1 << 2)
#define ACCEL_Y_ENABLE            (1 << 1)
#define ACCEL_X_ENABLE            (1 << 0)
#define ACCEL_XYZ_ENABLE          (7 << 0)

/**
 *      CTRL_REG_2 - Configure flags for High Pass Filter
 *
 *      Bits                Bit_Name                Function
 *      7-6                 HPM<1:0>        High Pass Filter mode selection
 *       5-4                HPCF<1:0>       High pass filter cut-off frequency configuration
 *      3                   FDS              Filtered data selection
 *      2                   HPClick          HPF enabled for CLICK Function
 *      1                   HPIS2            High pass filter enabled for interrupt 2 source
 *      0                   HPIS1            High pass filter enabled for interrupt 1 source
 *
 *      Default Value : 0000 0000
 */
#define ctrl_reg2                       0x21

/* HPF mode configuration */
#define ACCEL_HPF_NORMAL_RESET          0
#define ACCEL_HPF_REF_SIGNAL            (1 << 6)
#define ACCEL_HPF_NORMAL_MODE           (2 << 6)
#define ACCEL_HPF_AUTORST_INTR          (3 << 6)

/* HPF filtered data selection */
#define ACCEL_HPF_FDS_ENABLE            (1 << 3)

#define STATUS_REG                      0x27

/* Data Available flags */
#define ACCEL_STATUS_ZYXDA              (1 << 3)
```

```
* CTRL_REG_4 - Configure LE/BE data, full-scale range, high resolution, self test & block data update
*       Bits                    Bit_Name                    Function
*       7                       BDU                 Block Data Update
*       6                       BLE                 Big/Little Endian data Selection
*       5-4                     FS<1:0>             Full-Scale selection
*       3                       HR                  High resolution output mode
*       2                       ----                Reserved (0)
*       1                       ST                  Self Test Enable
*       0                       ----                Reserved (0)
*
*       Default Value : 0000 0000
*/
#define ctrl_reg4                       0x23

/* Accel block data update config */
#define ACCEL_BDU_ENABLE                (0X01 << 7)
#define HIGH_RES_OP_ENABLE              (0X01 << 3)

/* Accel Full Scale Range */
#define ACCEL_RANGE_2G                  0       /* 1 mg/LSB */
#define ACCEL_RANGE_4G                  1       /* 2 mg/LSB */
#define ACCEL_RANGE_8G                  2       /* 4 mg/LSB */
#define ACCEL_RANGE_16G                 3       /* 12 mg/LSB */
#define ACCEL_RANGE_OFFSET              4
#define ACCEL_RANGE_MASK                (3 << ACCEL_RANGE_OFFSET)

/*Output registers definition*/
#define OUT_X_L                         0x28
#define OUT_X_H                         0x29
#define OUT_Y_L                         0x2A
#define OUT_Y_H                         0x2B
#define OUT_Z_L                         0x2C
#define OUT_Z_H                         0x2D

#endif /* ACCEL_HEADER_H_ */
```

Example code for I2C bit-banging

```
/*
 * Bit banging I2C using micro-controller.
 */

#define SDA P1_0
#define SCL P1_1

 //During initialization both clock and data lines are at logic 1.
void I2CInit(){
SDA = 1;
SCL = 1;
}

// I2C transactions start with sending high to low pulse on clock line.
void I2CStart(){
SCL = 1;
SDA = 0;
SCL = 0;
}

// Stop bit for I2C is sending low to high pulse on data line.
void I2CStop(){
SDA = 0;
SCL = 1;
SDA = 1;
}
 /*
  * During sending data over I2C one need to follow steps mentioned below:
  * 1. Pull clock pin to low, because data should remain stable while clk pin is hig
  * 2. After pulling clock to low, change data on pin.
  * 3. pull clk pin to high.
  */

unsigned char I2CSend(unsigned char Data)
{
    unsigned char i, ack_bit;

    for(i = 0;i < 8; i++){
       SCL = 0;
       if ((Data & 0x80) == 0)
           SDA = 0;
       else
           SDA = 1;
       SCL = 1;
       Data <<= 1;
    }
    SCL = 0;
    ack_bit = SDA;
    SCL = 1;
    SCL = 0;
    return (!ack_bit);
```

```
53
54  /*
55  * During Reading data over I2C one need to follow steps mentioned below:
56  * 1. Pull clock pin to high, because data should remain stable while clk pin is
57  * 2. After pulling clock to high, read one bit data from pin.
58  *
59  */
60 unsigned char I2CRead(){
61     unsigned char i, Data = 0;
62
63     for(i = 0;i < 8;i++){
64         SCL = 0;
65         SCL = 1;
66         if(SDA)
67             Data |=1;
68         if(i<7)
69             Data <<=1;
70     }
71     SCL = 0;
72     SDA = 1;
73     return Data;
74 }
```

Example code to measure performance of system

```
 9  # include<stdio.h>
10  #include<conio.h>
11  #include<stdlib.h>
12  #include <string.h>
13
14
15  #define MAXLINE 10
16
17  char *lineptr[MAXLINE];
18
19  void readline(char *lineptr[], int nline);
20  void writeline(char *lineptr[], int nline);
21  void swap(char *v[],int i,int j);
22  void sort(char *lineptr[],int nline);
23
24  int main()
25  {
26
27      int nline;
28      nline =3;
29
30      readline(lineptr,nline);
31      sort(lineptr,nline);
32      writeline(lineptr,nline);
33      getch();
34
35      return 0;
36  }
37
38  void readline(char *lineptr[], int nline)
39  {
40    int i;
41
42    for(i=0;i<nline;i++)
43    {
44          char *p = (char *)malloc(50 *sizeof(char));
45         printf("ENter string ->");
46          gets(p);
47         lineptr[i]=p;
48
49    }
50  }
```

```
51
52  void writeline(char *lineptr[],int nline)
53  {
54    int i;
55    printf("\n****Sorted order of lines *****\n");
56
57    for(i=0;i<nline;i++)
58    {
59       printf("\n %d.%s ",i+1,lineptr[i]);
60
61    }
62  }
63
64  void sort(char *lineptr[], int nline)
65  {
66    int i,j;
67
68
69    for(i=0;i<nline;i++)
70    {   for(j=0;j<nline-1;j++)
71        {
72            if(strcmp(lineptr[j],lineptr[j+1])>0)
73               swap(lineptr,j,j+1);
74
75        }
76    }
77
78  }
79
80  void swap(char *s[] ,int i,int j)
81  {
82       char *temp;
83       temp = s[i];
84       s[i]=s[j];
85       s[j]=temp;
86
87  }
88
```