PERFORMING MULTITHREADING AND MULTITASKING ON LINUX

MAJOR PROJECT

Submitted in the fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Karthik Muthineni (13011M3421)

Manchikatla Uday Kumar (13011M3453)

Malisetty Reethi (13011M3403)

Under Esteemed Supervision of:

Dr. M. Asha Rani,

Professor

**Department Of Electronics & Communication Engineering
Jawaharlal Nehru Technological University Hyderabad College Of Engineering,
Kukatpally , Hyderabad-500 085,
December 2016**

Department Of Electronics & Communication Engineering,
Jawaharlal Nehru Technological University Hyderabad College Of Engineering,
Kukatpally , Hyderabad-500 085.

DECLARATION BY THE CANDIDATE

We , **M. Karthik** (13011M3453), **M.Uday Kumar** (13011M3453),**M.Reethi**(13011M3403) , here by declare that the report of the B. Tech Major Project work titled “**PERFORMING MULTITHREADING AND MULTITASKING ON LINUX**” under the esteemed guidance of **Dr. M . Asha Rani**, Professor , Department of ECE, JNTUH , College of Engineering Hyderabad is a bonafied work carried out by us. This is being submitted to **JNTUH-CEH**, in fulfilment of the requirements for the award of the degree of **Bachelor of Technology (IIDDMP) in Electronics And Communication Engineering** during the academic years 2013-18. The material contained in this report has not been submitted to any University or Institution for the award of any degree or diploma.

M. Karthik (13011M3421)

M.Uday kumar (13011M3453)

M.Reethi (13011M3403)

**Department Of Electronics & Communication Engineering,
Jawaharlal Nehru Technological University Hyderabad College Of Engineering,
Kukatpally , Hyderabad-500 085.**

CERTIFICATE

This is to certify that the Project Report entitled "**PERFORMING MULTITHREADING AND MULTITASKING ON LINUX**" that is being submitted by **M. Karthik** (13011M3421), **M. Uday Kumar**(13011M3453) and **M. Reethi**(13011M3403) in fulfillment for the award of the Degree of **Bachelor of Technology in Electronics & Communication Engineering** from Jawaharlal Nehru Technological University hyderabad, college of engineering (Autonomous) HYDERABAD is a bonafide record of project work carried out during the academic year 2016-2017.

Dr. T. SATYA SAVITRI
PROFESSOR AND HEAD OF ECE DEPARTMENT
JNTUH-COLLEGE OF ENGINEERING
HYDERABAD-500 085

**Department Of Electronics & Communication Engineering,
Jawaharlal Nehru Technological University Hyderabad College Of Engineering,
Kukatpally , Hyderabad-500 085.**

CERTIFICATE

This is to certify that the Project Report entitled "**PERFORMING MULTITHREADING AND MULTITASKING ON LINUX**" that is being submitted by **M. Karthik** (13011M3421), **M. Uday Kumar**(13011M3453) and **M. Reethi** (13011M3403) in fulfillment for the award of the Degree of **Bachelor of Technology in Electronics & Communication Engineering** from Jawaharlal Nehru Technological University hyderabad, college of engineering (Autonomous) HYDERABAD is a bonafide record of project work carried out during the academic year 2016-2017.

PROJECT GUIDE
Dr. M. ASHA RANI
PROFESSOR AND HEAD OF ECE DEPARTMENT
JNTUH-COLLEGE OF ENGINEERING
HYDERABAD-500 085

ACKNOWLEDGEMENT

We wish to acknowledge our profound sense of gratitude to our project guide **Dr. M. ASHA RANI, Professor, DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING** for her remarkable guidance and continued encouragement during the preparation of this project. Indeed, it was a matter of great felicity and privilege for us to work under her aegis. We express our thankfulness for her dedicated inspiration , lively interest and patience through our errors, without which it would have been impossible to bring the project to near completion.

We would like to take this opportunity to thank **Mrs. R. Dhruva, Research Scholar** who inspired us by being an example of high vision and for pushing us towards greater limits of achievement and has provided constant support and helped us rectify our mistakes.

We are immensely grateful to all the faculty, staff members and administration of Department for their kind co-operation and help.

Lastly, we would like to express our deep appreciation towards our classmates and family for providing the moral support and encouragement for the successful completion of the project.

M. KARTHIK (13011M3421)

M. UDAY KUMAR (13011M3453)

M. REETHI (13011M3403)

ABSTRACT

In a modern computing system, there are usually several concurrent application processes which compete for CPU. The Operating System (OS), amongst other duties, is responsible for the effective and efficient allocation of those resources. The OS module which handles resource allocation is called scheduler. On the basis of the type of OS to be realized, different scheduling policies may be implemented.

All modern computers run have capability of Multitasking and Multithreading. Multitasking, refers to having multiple programs, process, tasks, threads running at the same time. Multithreading is an execution model that allows a single process to have multiple code segments (i.e., *threads*) run concurrently within the “context” of that process or task.

The main objective of this project is to work with Multithreading and Multitasking. This project focuses on developing an single process like Matrix Multiplication of size $1000 * 1000$ having multiple code segments (threads) run concurrently on 2.33GHz quad-core i3 Processor and 900MHz quad-core ARM Cortex-A7 and to make comparison between them.

This project also develops an method for running multiple tasks at a time on particular isolated cpu core on 2.33GHz quad-core i3 Processor and 900MHz quad-core ARM Cortex-A7 and to make comparison between them. To achieve this software like QT CREATER will be used.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1-INTRODUCTION

1.1-UNIX

1.2-LINUX

1.2.1-Linux VS Windows

1.3-MULTITHREADING

1.3.1-Types of Threads

1.4-MULTIPROCESSING

CHAPTER 2-SOFTWARE REQUIREMENT

2.1-BASIC C LANGUAGE

2.2-LINUX CONSOLE

2.3-QT CREATOR

2.4-KERNEL PARAMETER

CHAPTER 3-PROGRAMMING THROUGH QT

3.1-CREATING GRAPHICAL USER INTERFACE

3.2--TYPE OF WIDGETS

3.3-SIMULATION OBSERVATIONS

CHAPTER 4-MATRIX MULTIPLICATION OF 1000 * 1000

4.1-LIBRARIES

4.1.1-Libraries Used

4.2-CREATING 1000 * 1000 MATRIX

4.3-CREATING THREADS

4.3.1-Assigning Thread to Core

4.4-RUNNING ON SINGLE CORE

4.4.1-Graphical Representation

4.5-RUNNING ON MULTICORE

4.5.1-Program Code on Single core And Multicore

4.5.2-Graphical Representation

4.6-TEST RESULTS

CHAPTER 5-MULTITHREADING ON RASPBERRY PI 2

5.1-HARDWARE

5.1.1-Specifications

5.2-INSTALLING RASPBIAN IMAGES

5.3-INSTALLING HTOP AND GCC

5.4-ISOLATING CPU CORES IN RASPBERRY PI 2

5.5-RUNNING MATRIX MULTIPLICATION ON SINGLE CORE ON PI 2

5.5.1-Graphical Representation

5.6-RUNNING MATRIX MULTIPLICATION ON DUAL CORE ON PI 2

5.6.1 -Graphical Representation

5.7-TEST RESULTS

CHAPTER 6-CONCLUSION AND FUTURE SCOPE

CHAPTER 7-REFERENCES

LIST OF FIGURES

Figure 1.2 – Linux Operating System

Figure 1.3 – Processing of threads

Figure 1.3.1 - User level Threads

Figure 2.2 - Linux Console

Figure 2.5 - Displaying list of process using top

Figure 2.5 - Displaying usage of cpu using htop

Figure 3.1 - QT Creator GUI

Figure 3.2.1 - Selection of Filters to Debug Output

Figure 3.2.2 - Types of Filters

Figure 3.2.3 - Application Output

Figure 3.2.4 - Compiler Output

Figure 3.2.5 - Debugger Console

Figure 3.2.6 - Widget GUI

Figure 4.1 – Matrix creation of size 1000x1000

Figure 4.2 – Thread process

Figure 4.2.1 - Running on Single Core

Figure 4.2.2 - Running on Dual Core

Figure 4.3 - Graph of matrix multiplication on Single Core

Figure 4.4 - Graph of matrix multiplication on Dual Core

Figure 5.1 - Raspberry pi 2

Figure 5.2 - Raspbian OS

Figure 5.3 - Htop in Raspberry pi 2

Figure 5.4 - GCC

Figure 5.5 - Isolating cpu cores in pi 2

Figure 5.6 - Running matrix multiplication on Single Core

Figure 5.7 - Graphical Representation

Figure 5.8 - Running matrix multiplication Dual Core

Figure 5.9 - Graphical Representation

LIST OF TABLES

Table 1.2.1 LINUX VS WINDOWS

Table 1.3 PROCESS AND THREAD

Table 1.3.1 USER AND KERNEL LEVEL THREADS

Table 5.7 TEST RESULTS OF MULTITHREADING ON PI 2

CHAPTER 1 - INTRODUCTION

1.1 UNIX

UNIX is a computer Operating System which is capable of handling activities from multiple users at the same time. The UNIX operating system is a set of programs that act as a link between the computer and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the operating system or kernel.

Users communicate with the kernel through a program known as the shell. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

1.2 LINUX

Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

- Kernel – Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- System Library – System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.

- System Utility – System Utility programs are responsible to do specialized, individual level tasks.

Figure 1.2 – Linux Operating System

Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called kernel mode with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in User Mode which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

Following are some of the important features of Linux Operating System.

- Portable – Portability means software can work on different types of hardware in the same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- Open Source – Linux source code is freely available and it is a community-based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- Multi-User – Linux is a multiuser system means multiple users can access system resources like memory/ RAM/ application programs at the same time.
- Multiprogramming – Linux is a multiprogramming system means multiple applications can run at the same time.
- Hierarchical File System – Linux provides a standard file structure in which system files/ user files are arranged.
- Shell – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs, etc.
- Security – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

1.2.1 LINUX VS WINDOWS

Topic	Linux	Windows
Price	The Linux Kernel, and the GNU utilities and libraries which accompany it in most distributions, are entirely free and opensource. You can download and install GNU/Linux distributions without purchase. Some companies offer paid support for their Linux distributions, but the underlying software is still free to download and install.	Microsoft Windows usually costs between \$99.00 and \$199.00 USD for each licensed copy. However, Windows 10 is being offered as a free upgrade to current owners of Windows 7 or Windows 8.1 if they upgrade before July 29, 2016.
Ease Of Use	GNU/Linux operating systems have a steeper learning curve for the average user. They frequently require a deeper understanding of the underlying system to perform day-to-day functions, and troubleshooting technical issues can be a more intimidating and complicated process than on Windows. However, some distributions such as Ubuntu and Linux Mint are designed specifically to ease the transition from Windows to a Linux environment.	Windows is one of the easiest desktop operating systems to use. One of its primary design characteristics is user-friendliness and simplicity of basic system tasks. This is considered a positive by users who want their system to "just work", but more proficient users may be frustrated by oversimplification of system tasks at the expense of fine-grained control over the system itself.
Reliability	Linux is notoriously reliable and secure. It has a strong focus on process management, system security, and uptime.	Although Microsoft Windows has made great improvements in reliability in recent years, it's considered less reliable than Linux. Many of the sacrifices it makes in the name of user-friendliness can lead to security vulnerabilities and system instability. Software There thousands of programs available

		for Linux, and many are available as easy-to-install software packages — all for free. Also, many Windows programs can be run on Linux using compatibility layers such as WINE. Linux supports a wider array of free software than Windows.
Software	There thousands of programs available for Linux, and many are available as easy-to-install software packages — all for free. Also, many Windows programs can be run on Linux using compatibility layers such as WINE. Linux supports a wider array of free software than Windows.	Windows commands the highest number of desktop users, and therefore the largest selection of commercial software. It also has the largest selection of video games, although Linux has made significant gains on this front — especially with the continued development of Linux-compatible gaming platforms such as steam.
Software Cost	Many of the available software programs, utilities, and games available on Linux are free and open source. Even complex applications such as GIMP, Open office, and Star office are available for free.	Although there are many free Windows programs, utilities, and games, the majority of the programs are commercial
Security	Linux is a highly secure operating system. Although attack vectors are still discovered, its source code is open and available for any user to review, which makes it easier to identify and repair vulnerabilities.	Microsoft has made great security improvements over the years. But as the operating system with the largest user base, especially among novice computer users, it is the primary target for malicious coders. As a result, of all major operating systems, Microsoft Windows is the most likely to be the victim of viruses and malware.
Support	There is a massive amount of online	Microsoft Windows offers

	support available for Linux, including here on Computer hope.	integrated and online help systems, and there are thousands of informative books about Windows available for every skill level.
Use Cases	Linux is used by corporate, scientific, and academic organizations of every size. It's used to power the development machines and servers at Google, Facebook, Twitter, NASA It's also used to give new life to older hardware, enable low-budget computing projects to succeed, and serve as the operating system on single-board computers such as the Raspberry pi.	Microsoft Windows is usually the operating system of choice for gamers, novice users, and business users who rely on Microsoft software. It doesn't run well on older hardware, however. Many Windows users are thrilled with the changes that Microsoft has introduced with Windows 10, so if you've never used it, now is a great time to try it out.

Table 1.2.1 Linux VS Windows

1.3 MULTI THREADING

THREAD

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a lightweight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.

Figure 1.3 – Processing of threads

PROCESS AND THREAD

S.N.O	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.

4	If one process is blocked, then no other process can execute until the first process is unblocked	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Table 1.3 Process and Thread

1.3 TYPES OF THREADS

User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

Figure 1.3.1 - User level Threads

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

User-Level & Kernel-Level Thread

S.N.O	User-Level Threads	Kernel-Level Threads
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Table 1.3.1 User and Kernel level Threads

1.4 Multi processing

In computing, a process is an instance of a computer program that is being executed. Or simply running program is also called as process.

Multiprocessing means “ Having Two or more CPU’s in a single computer system”.

Multiprocessing is defined as multiple process can execute at same time rather than one after another. In Multiprocessing, each process have different address space and resources.

Types of Multiprocessing

Symmetric multiprocessing(SMP)

Symmetric multiprocessing (SMP) involves a multiprocessor computer hardware architecture where two or more identical processors are connected to a single shared main memory and are controlled by a single OS instance. Most common multiprocessor systems today use an SMP architecture. In the case of multi-core processors, the SMP architecture applies to the cores, treating them as separate processors. Processors may be interconnected using buses, crossbar switches or on-chip mesh networks.

Asymmetric multiprocessing(AMP)

It was the only method for handling multiple CPUs before [symmetric multiprocessing](#) (SMP) was available. It has also been used to provide less expensive options on systems where SMP was available. Additionally, AMP is used in applications that are dedicated, such as embedded systems, when individual processors can be dedicated to specific tasks at design time. In an asymmetric multiprocessing system, not all CPUs are treated equally; for example, a system might only allow (either at the hardware or operating system level) one CPU to execute operating system code or might only allow one CPU to perform I/O operations. Other AMP systems would allow any CPU to execute operating system code and perform I/O operations, so that they were symmetric with regard to processor roles, but attached some or all peripherals to particular CPUs, so that they were asymmetric with regard to peripheral attachment.

Asymmetric Multiprocessing:

Symmetric Multiprocessing:

Figure 1.4 - Types of Multiprocessing

CHAPTER 2 - SOFTWARE REQUIREMENT

2.1 Basic C Language

C is a general-purpose high level language that was originally developed by Dennis Ritchie for the Unix operating system. It was first implemented on the Digital Equipment Corporation PDP-11 computer in 1972. The Unix operating system and virtually all Unix applications are written in the C language.

2.2 Linux Console

The Linux console is a system console internal to the Linux Kernel (a system console is the device which receives all kernel messages and warnings and which allows logins in single user mode). The Linux console provides a way for the kernel and other processes to send text output to the user, and to receive text input from the user. The user typically enters text with a computer keyboard and reads the output text on a computer monitor. The Linux kernel supports virtual consoles - consoles that are logically separate, but which access the same physical keyboard and display. The Linux console (and Linux virtual consoles) are implemented by the VT subsystem of the Linux kernel, and do not rely on any user space software.

```

top - 16:03:06 up 3:19, 3 users, load average: 0.61, 0.24, 0.18
Tasks: 229 total, 1 running, 228 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4.7 us, 1.0 sy, 0.0 ni, 93.7 id, 0.5 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem: 8105460 total, 4015224 used, 4090236 free, 190704 buffers
KiB Swap: 8318972 total, 0 used, 8318972 free, 1614160 cached Mem

```

PID	USER	PR	NI	VSZ	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2795	uday	20	0	2408128	13816	132268	S	8.6	17.9	34:48.88	firefox
2272	uday	20	0	1535632	204388	67388	S	6.7	2.5	2:16.61	compiz
1424	root	20	0	420200	88516	77108	S	3.3	1.1	2:37.48	Xorg
8686	uday	20	0	883660	14860	9352	S	1.3	0.2	0:00.12	gvfsd-http
8711	uday	20	0	608790	27156	20444	S	0.7	0.3	0:00.15	gnome-scre+
2145	uday	20	0	624344	30864	21688	S	0.3	0.4	0:07.25	unity-pane+
6389	uday	20	0	629316	31892	21988	S	0.3	0.4	0:00.58	gnome-term+
8035	root	20	0	0	0	0	S	0.3	0.0	0:00.19	kworker/1:1
8545	root	20	0	0	0	0	S	0.3	0.0	0:00.17	kworker/u1+
8591	root	20	0	0	0	0	S	0.3	0.0	0:00.07	kworker/u1+
1	root	20	0	33772	4412	2756	S	0.0	0.1	0:01.24	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.04	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:+
7	root	20	0	0	0	0	S	0.0	0.0	0:30.86	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	rt	0	0	0	0	S	0.0	0.0	0:00.01	migration/0

Figure 2.2 - Linux Console

2.3 Qt Creator

Qt Creator provides a cross-platform, complete integrated development environment (IDE) for application developers to create applications for multiple desktop, embedded, and mobile device platforms, such as Android and IOS. It is available for Linux, macOS and Windows operating systems.

Figure 2.3 - QT Creater

Developing Widget Based Applications

Code Editor Integration

To switch between forms (Design mode) and code (Edit mode), press Shift+F4.

You can use Qt Creator to create stub implementations of slot functions. In the Design mode, right-click a widget to open a context menu, and then select Go to Slot. Select a signal in the list to go to an existing slot function or to create a new slot function.

Managing Image Resources

In standalone Qt Designer, image resources are created using the built-in Resource Editor. In Qt Creator, .ui files are usually part of a project, which may contain several resource files (.qrc). They are created and maintained by using the Qt Creator Resource Editor. The Qt Designer Resource Editor is de-activated and the image resources are displayed in the Qt Designer Resource Browser.

Specifying Settings for Qt Designer

To specify settings for Qt Designer:

- Select Tools > Options > Designer.
- Specify settings for generating classes and code in Class Generation.
- Specify embedded device profiles, that determine style, font, and screen resolution, for example, in Embedded Design.
- Specify settings for the grid and previewing forms in Forms.
- Specify an additional folder for saving templates in Template Paths.

To preview the settings, select Tools > Form Editor > Preview, or press Alt+Shift+R.

Previewing Forms Using Device Skins

A *device skin* is a set of configuration files that describe a mobile device. It includes a border image that surrounds the form and depicts a mobile device with its buttons.

To preview your form using device skins:

1. Select Tools > Options > Designer.
2. Select the Print/Preview Configuration check box.
3. In the Device skin field, select a device skin.
4. When the form is open in Design mode, press Alt+Shift+R.
5. To end the preview, right-click the skin and select Close in the context menu

Isolcpus

`Isolcpus` argument is a cpu list. This option can be used to specify one or more CPUs to isolate from the general SMP balancing and scheduling algorithms. You can move a process onto or off an "isolated" CPU via the CPU affinity syscalls or `cpuset`. `<cpu number>` begins at 0 and the maximum value is "number of CPUs in system - 1".

This option is the preferred way to isolate CPUs. The alternative -- manually setting the CPU mask of all tasks in the system -- can cause problems and suboptimal load balancer performance.

Top

The Linux `top` command is used to show all the running processes within your Linux environment. The `top` program provides a dynamic real-time view of a running system.

Keep Linux Top Running All The Time In The Background

You can keep the `top` command easily available without having to type the word `top` each time into your terminal window.

To pause `top` so that you can continue using the terminal, press `CTRL` and `Z` on the keyboard. To bring `top` back to the foreground, type `fg`.

Key Switches For The Top Command

- `-h` - Show the current version
- `-c` - This toggles the command column between showing command and program name
- `-d` - Specify the delay time between refreshing the screen

- -o - Sorts by the named field
- -p - Only show processes with specified process IDs
- -u - Show only processes by the specified user

Adding Extra Columns To The Top Display

Whilst running top you can press the 'F' key which shows the list of fields that can be displayed in the table: Use the arrow keys to move up and down the list of fields.

To set a field so that it is displayed on the screen press the 'D' key. To remove the field press "D" on it again. An asterisk (*) will appear next to displayed fields.

You can set the field to sort the table by simply by pressing the "S" key on the field you wish to sort by. Press the enter key to commit your changes and press "Q" to quit.

Changing Colors

Press the "Z" key to change the colors of the values within top.

There are three stages required to change the colors:

1. Press either S for summary data, M for messages, H for column headings or T for task information to target that area for a color change
2. Choose a color for that target, 0 for black, 1 for red, 2 for green, 3 for yellow, 4 for blue, 5 for magenta, 6 for cyan and 7 for white
3. Enter to commit

```
top - 16:03:06 up 3:19, 3 users, load average: 0.61, 0.24, 0.18
Tasks: 229 total, 1 running, 228 sleeping, 0 stopped, 0 zombie
%Cpu(s): 4.7 us, 1.0 sy, 0.0 ni, 95.7 id, 0.5 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem: 8105460 total, 4015224 used, 4090236 free, 190704 buffers
KiB Swap: 8318972 total, 0 used, 8318972 free. 1614160 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 2795 uday      20   0 2400128 1.381g 132268 S   8.6 17.9 34:48.88 firefox
 2272 uday      20   0 1535632 204388 67308 S   6.7 2.5 2:16.01 compiz
 1424 root       0   0 420280 88516 77108 S   3.3 1.1 2:37.48 Xorg
 8686 uday      20   0 883660 14860 9352 S   1.3 0.2 0:00.12 gvfsd-http
 8711 uday      20   0 508790 27150 20444 S   0.7 0.3 0:00.15 gnome-scre+
 2145 uday      20   0 624344 38864 21688 S   0.3 0.4 0:07.25 unity-pane+
 6309 uday      20   0 629316 31692 21908 S   0.3 0.4 0:00.58 gnome-term+
 8035 root       0   0 0 0 0 S   0.3 0.0 0:00.19 kworker/1:1
 8545 root      20   0 0 0 0 S   0.3 0.0 0:00.17 kworker/ui+
 8591 root      20   0 0 0 0 S   0.3 0.0 0:00.07 kworker/ui+
    1 root      20   0 33772 4412 2756 S   0.0 0.1 0:01.24 init
    2 root      20   0 0 0 0 S   0.0 0.0 0:00.00 kthreadd
    3 root      20   0 0 0 0 S   0.0 0.0 0:00.04 ksoftirqd/0
    5 root      0 -20 0 0 0 S   0.0 0.0 0:00.00 kworker/0:1
    7 root      20   0 0 0 0 S   0.0 0.0 0:30.06 rcu_sched
    8 root      20   0 0 0 0 S   0.0 0.0 0:00.00 rcu_bh
    9 root      rt   0 0 0 0 S   0.0 0.0 0:00.01 migration/0
```

Figure 2.3 - Top

Htop

Htop will show you your usage per CPU, as well as a meaningful text graph of your memory and swap usage right at the top.

Installing htop on Ubuntu

```
sudo apt-get install htop
```

Once installed, just type htop at a terminal to launch it, and notice the great text-mode graph at the top of the display We can use our Up/Down arrow keys to select a process, and then you can kill it with the F9 key if you'd like, or you can change the priority by using the F7 and F8 keys. (note that you'll have to be root to give anything really high priority)

Figure 2.4 - Htop

CHAPTER 3 - PROGRAMMING THROUGH QT

3.1 Creating GUI

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt widgets. We can compose and customize our windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and test them using different styles and resolutions.

Widgets and forms created with Qt Designer integrate seamlessly with programmed code, using Qt's signals and slots mechanism, so that you can easily assign behavior to graphical elements. All properties set in Qt Designer can be changed dynamically within the code. Furthermore, features like widget promotion and custom plugins allow you to use your own components with Qt Designer.

GRAPHICAL USER INTERFACE

When you start Qt Creator, it opens to the Welcome mode, where you can:

- Open recent sessions and projects
- Create and open projects
- Open tutorials and example projects
- Read news from the online community and Qt blogs
- Create or manage a Qt Account

Figure 3.1 - QT Creator GUI

Modes

The mode selector allows you to quickly switch between tasks such as editing project and source files, designing application UIs, configuring how projects are built and executed, and debugging your applications. To change modes, click the icons, or use the corresponding keyboard shortcuts.

To hide the mode selector and to save space on the display, select **Window > Show Mode Selector**.

The following image displays an example application in Edit mode (1) and Design mode (2).

Viewing Project Files

The sidebar displays projects in a project tree. The project tree contains a list of all projects open in the current session. The files for each project are grouped according to their file type.

If the project is under version control, information from the version control system might be displayed in brackets after the project name. This is currently implemented for Git (the branch name or a tag is displayed) and ClearCase (the view name is displayed).

You can use the project tree in the following ways:

- To open files that belong to a project, double-click them in the project tree. Files open in the appropriate editor, according to the file type. For example, code source files open in the code editor and image files in the [image viewer](#).
- To bring up a context menu containing the actions most commonly needed right-click an item in the project tree. For example, through the menu of the project root directory you can, among other actions, build, re-build, clean and run the project.
- To hide the categories and sort project files alphabetically, click (Filter Tree) and select Simplify Tree.
- To hide source files which are automatically generated by the build system, select Filter Tree > Hide Generated Files.
- To keep the position in the project tree synchronized with the file currently opened in the editor, click (Synchronize with Editor).
- To see the absolute path of a file, move the mouse pointer over the file name.

Files that are not sources or data can be still included into a project's distribution tarball by adding their paths to the DISTFILES variable in the .pro file. This way they also become known to Qt Creator, so that they are visible in the Projects view and are known to the locator and search.

The Projects view contains context menus for managing projects, subprojects, folders, and files.

The following functions are available for managing projects and subprojects:

- Set a project as the active project.

- Execute the Build menu commands.
- Add subprojects, libraries, directories, and files.
- Remove subprojects.
- Search from the selected directory.
- Close projects.

For managing files and directories, the same functions are available as in the File System view.

In addition, you can remove and rename files.

Viewing the File System

If you cannot see a file in the Projects view, switch to the File System view, which shows all the files in the file system.

To also show hidden files, select Filter Files > Show Hidden Files.

To keep the position in the tree synchronized with the file opened in the editor, select Synchronize with Editor.

Use the context menu functions to:

- Open files with the default editor or some other editor.
- Show the file or directory in the file explorer.
- Open a terminal window in the selected directory or in the directory that contains the file.
- Search from the selected directory.

Viewing Tests

The Tests view shows the autotests and Qt Quick tests in the current project. Select the test cases to run. If a Qt Quick test case does not have a name, it is marked Unnamed in the list. Unnamed test cases are always executed when the application that contains them is run, and therefore you cannot select or deselect them.

To show or hide init and cleanup or data functions, select Filter Test Tree > Show Init and Cleanup Functions or Show Data Functions. Double-click a function in the list to open its source code in the code editor.

The test cases are listed in alphabetic order. To list them in the order in which they are defined in the source code, select (Sort Naturally).

To run tests, select Run All Tests or Run Selected Tests in the context menu.

Viewing Output

The task pane in Qt Creator can display one of the following panes:

- Issues
- Application Output
- Compile Output
- Debugger Console
- Test Results

Output panes are available in all modes. Click the name of an output pane to open the pane. To maximize an open output pane, click the Maximize Output Pane button or press Alt+9.

To search within the Application Output and Compile Output panes, press Ctrl+F when the pane is active. Enter search criteria in the Find field and click the left and right arrows to search down and up in the pane. In these panes, you can also use the zoom buttons to increase and decrease the text size of the output.

Issues

The Issues pane provides lists of following types of issues:

- Build System - Errors and warnings encountered during a build.
- Compile - Selected output from the compiler. Open the Compile Output pane for more detailed information.
- Debug Information - Lists debug information packages that might be missing.
- Debugger - Errors encountered while running the [Valgrid code analysis tools](#).
- Debugger Runtime - Errors encountered when starting Qt Creator. For example, information about missing DLLs.
- Deployment - Errors encountered between building an application successfully and starting it on a device.

- My Tasks - Entries from a task list file (.tasks) generated by code scanning and analysis tools.

The pane filters out irrelevant output from the build tools and presents the issues in an organized way. To further filter the output by type, select Filter Tree and then select a filter.

Figure 3.2.2 - Type of Filters

Application Output

The Application Output pane displays the status of a program when it is executed, and the debug output.

Figure 3.2.2 - Application Output

Select toolbar buttons to run applications, to attach the debugger to the running application, and to stop running or debugging. To specify settings for displaying application output, select Tools > Options > Build & Run > General. You can select whether to open the Application Output pane on output when running or debugging applications, to clear old output on a new run, to word-wrap output, and to limit output to the specified number of lines.

Compile Output

The Compile Output pane provides all output from the compiler. The Compile Output is a more detailed version of information displayed in the Issues pane.

```
Compile Output
17:05:50: Running steps for project clocks...
17:05:50: Starting: "C:\dev\qt5-build\qtbase\bin\qmake.exe" C:
\dev\qt5\qtdeclarative\examples\quick\demos\clocks\clocks.pro
-r -spec win32-msvc2013 "CONFIG+=qml_debug"
17:05:50: The process "C:\dev\qt5-build\qtbase\bin\qmake.exe"
exited normally.
17:05:50: Starting: "C:\Qt\qtcreator-4.0.0-rc1-04-06-16\bin
\jom.exe"
C:\Qt\qtcreator-4.0.0-rc1-04-06-16\bin\jom.exe -f
Makefile.Debug
cl -c -nologo -Zc:wchar_t -FS -Zi -MDd -
D_HAS_EXCEPTIONS=0 -GR -W3 -w34100 -w34189 -w44996 /Fddebug
\clocks.pdb -DUNICODE -DWIN32 -DWIN64 -DQT_QML_DEBUG -
```

Figure 3.2.3 - Compiler Output

Double-click on a file name in an error message to open the file in the code editor.

Select the Cancel Build button to cancel the build.

To specify whether to open the Compile Output pane on output when building applications, select Tools > Options > Build & Run > General, and then select the Open Compile Output pane when building check box.

Debugger Console

In the Debugger Console, you can type JavaScript expressions and use them to get information about the state of your application during debugging. You can change property values temporarily, without editing the source, and view the results in the running application.

Figure 3.2.4 - Debugger Console

Widgets

Widgets are the primary elements for creating user interfaces in Qt. Widgets can display data and status information, receive user input, and provide a container for other widgets that should be grouped together. A widget that is not embedded in a parent widget is called a window.

Figure 3.2.6 - Widget GUI

3.1.1 Types of Widgets used

1. QComboBox

The [QComboBox](#) widget is a combined button and popup list. A [QComboBox](#) provides a means of presenting a list of options to the user in a way that takes up the minimum amount of screen space.

A combobox is a selection widget that displays the current item, and can pop up a list of selectable items. A combobox may be editable, allowing the user to modify each item in the list. Comboboxes can contain pixmaps as well as strings; the [insertItem\(\)](#) and [setItemText\(\)](#) functions are suitably overloaded. For editable comboboxes, the function [clearEditText\(\)](#) is provided, to clear the displayed string without changing the combobox's contents.

Member Function

`void QComboBox::activated(const QString & text)`

This signal is sent when the user chooses an item in the combobox. The item's *text* is passed. Note that this signal is sent even when the choice is not changed. If we need to know when the choice actually changes, use signal [currentIndexChanged\(\)](#).

Example - `void widget::on_comboBox_activated(const QString &arg1)`

This is created in the edit mode when the combobox is clicked to slot in user interface mode.

This explains as the signal is sent when we choose an item in combobox.

2. QPushButton

The [QPushButton](#) widget provides a command button. The push button, or command button, is perhaps the most commonly used widget in any graphical user interface. Push (click) a button to command the computer to perform some action, or to answer a question. Typical buttons are OK, Apply, Cancel, Close, Yes, No and Help.

Example - `QPushButton *button = new QPushButton("&Download", this);`

CHAPTER 4 - MATRIX MULTIPLICATION OF 1000*1000

4.1 Library

A library in C is a group of functions and declarations, exposed for use by other programs. The library therefore consists of an interface expressed in a .h file (named the "header") and an implementation expressed in a .c file. This .c file might be precompiled or otherwise inaccessible, or it might be available to the programmer.

4.1.1 Libraries Used

Pthread.h

POSIX Threads, usually referred to as Pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time. Each flow of work is referred to as a thread, and creation and control over these flows is achieved by making calls to the POSIX Threads API.

Sched.h

The <sched.h> header defines the sched_param structure, which contains the scheduling parameters required for implementation of each supported scheduling policy. Each process is controlled by an associated scheduling policy and priority. Associated with each policy is a priority range. Each policy definition specifies the minimum priority range for that policy. The priority ranges for each policy may overlap the priority ranges of other policies.

Stdio.h

The C programming language provides many standard library functions for file input and output. These functions make up the bulk of the C standard library header <stdio.h>. The functionality descends from a "portable I/O package" written by Mike Lesk at Bell Labs in the early 1970s. and officially became part of the Unix operating system in Version 7.

Stdlib.h

stdlib.h is the header of the general purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others.

It is compatible with C++ and is known as cstdlib in C++. The name "stdlib" stands for "standard library".

4.2 Creating matrix of 1000 * 1000

Let us consider matrix A with dimension $\text{dim } M \times \text{dim } N$ and matrix B with dimensions $\text{dim } N \times \text{dim } K$. The matrix A and B should be compatible with each other, that is, the number of columns in A should be equal to the number of rows in B. This fact is reflected by using the same name $\text{dim } N$ to this end. The result of a product $A B$ is the matrix C with dimensions $\text{dim } M \times \text{dim } K$. Each row and column of matrix A and matrix B are multiplied and new matrix C is formed.

$$C = A * B$$

Figure 4.1 – Matrix creation of size 1000x1000

4.3 Creating threads

Define thread reference variables

The variable type `pthread_t` is a means of referencing threads. There needs to be a `pthread_t` variable in existence for every thread being created. Something like `pthread_t thread 0;` will do the trick.

Create an entry point for the thread

When creating a thread using pthreads, you need to point it to a function for it to start execution. The function must return void * and take a single void * argument. For example, if you want the function to take an integer argument, you will need to pass the address of the integer and dereference it later. This may sound complicated but, as is shown below, it's pretty simple. An example function signature would be void *my_entry_function(void *param);

Create the thread

Once the pthread_t variable has been defined and the entry point function created, we can create the thread using pthread_create. This method takes four arguments: a pointer to the pthread_t variable, any extra attributes (don't worry about this for now – just set it to NULL), a pointer to the function to call (ie: the name of the entry point) and the pointer being passed as the argument to the function. Now there's a lot of pointers in that call, but don't stress – it's not as tricky as it sounds. This call will look something like pthread_create(& thread 0, NULL, my_entry_function, parameter).

Join everything back up

When the newly-created thread has finished doing its bits, we need to join everything back up. This is done by the pthread_join function which takes two parameters: the pthread_t variable used when pthread_create was called (not a pointer this time) and a pointer to the return value pointer (don't worry about this for now – just set it to NULL). This call will look something like pthread_join(thread 0, NULL);

And that's all there is to it. The function used as the thread entry point can call other functions, create variables or do anything any other function can do. It can also use the variables set by the other thread.

Compilation

When compiling the program, you will also need to add -lpthread to the compile command. ie: gcc program.c -o program -lpthread

Figure 4.2 – Thread process

4.3.1 Assigning thread to core

To run a process only on a particular CPU core you need to specify the affinity of that process to that particular core. This is called data affinity. You can use `sched_setaffinity()` to do this.

Steps for isolating cpu cores

- Before loading the Bios PRESS and HOLD SHIFT to enter into GRUB
 - After entering into GRUB goto advanced option and press 'e' to edit config file it will open a window with set of parameters.
 - Scroll Down to the line `linux /boot/vmlinuz-4.2.0-27-generic` at the end of line after `handoff` give space then type `isolcpus=no` of cpu core you want to isolate
- Ex : 1) `isolcpus=3` //to isolate core 3
- 2) `isolcpus=2,3` // to isolate core 2 and 3
- Then press `ctrl+x` to reboot.this will temporary isolate the cores.

4.4 Running on Single core

For running process on particular core we can use “cpu_set_t”

```
1  cpu_set_t my_set;      /* Define your cpu_set bit mask. */
2  CPU_ZERO(&my_set);    /* Initialize it all to 0 */
3  CPU_SET(2, &my_set);  /* set the bit that represents core 2 */
4  sched_setaffinity(0, sizeof(cpu_set_t), &my_set);
```

For compiling our C code following command need to be executed.

- Gcc filename.c -o filename

Above command will create an executable file which need to be compiled using

- Time ./filename

Above command will display time taken by program to run .

Figure 4.2.1 - Running on Single Core

Running on multicore

For running on multicore the process is similar to that of singlecore.

For compiling our C code following command need to be executed.

- `Gcc filename.c -o filename -lpthread`

Above command will create an executable file which need to be compiled using

- `Time ./filename 2 3 1000000000`

Where 2 is parentcpunumber and 3 is childcpunumber and 1000000000 are number of loops.

Figure 4.2.2 - Running on Dual Core

4.5 Program code

source code (Matrix Multiplication on single core)

```
#define _GNU_SOURCE
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sched.h>
#define M 1000
#define K 30
#define N 1000
#define NUM_THREADS 2

int A [M][K] = { { 1,4}, { 2,5}, { 3,6} };
int B [K][N] = { { 8,7,6}, { 5,4,3} };
int C [M][N];

struct v {
    int i; /* row */
    int j; /* column */
};

void *runner(void *param); /* the thread */
int main(int argc, char *argv[]) {

    int i,j,v,s, count = 0;
    int num_threads=2;
    pthread_t p[num_threads];
    cpu_set_t my_set;          /* Define your cpu_set bit mask. */
    CPU_ZERO(&my_set);        /* Initialize it all to 0 */
```

```

CPU_SET(3, &my_set);
sched_setaffinity(0, sizeof(cpu_set_t), &my_set);
for(i = 0; i < M; i++) {
for(j = 0; j < N; j++) {
//Assign a row and column for each thread
struct v *data = (struct v *) malloc(sizeof(struct v));
data->i = i;
data->j = j;
/* Now create the thread passing it data as a parameter */
pthread_t tid;      //Thread ID
pthread_attr_t attr; //Set of thread attributes
//Get the default attributes
pthread_attr_init(&attr);
//Create the thread
pthread_create(&tid,&attr,runner,data);
//Make sure the parent waits for all thread to complete
pthread_join(tid, NULL);
count++;
}
}

//Print out the resulting matrix
for(i = 0; i < M; i++) {
for(j = 0; j < N; j++) {
printf("%d ", C[i][j]);

}
printf("\n");
}

```

```

}

//The thread will begin control in this function
void *runner(void *param) {

    //sched_setaffinity(0,sizeof(cpuset),&cpuset);
    struct v *data = param; // the structure that holds our data
    int n, sum = 0; //the counter and sum

    //Row multiplied by column
    for(n = 0; n < K; n++){
        sum += A[data->i][n] * B[n][data->j];
    }
    //assign the sum to its coordinate
    C[data->i][data->j] = sum;

    //Exit the thread

    pthread_exit(0);
}

```

Source Code (Matrix Multiplication on Multi core)

```
#define _GNU_SOURCE
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sched.h>
#include <unistd.h>
#include <sys/wait.h>

#define M 1000
#define K 30
#define N 1000
#define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE); \
                        } while (0)

int A [M][K] = { { 1,4}, {2,5}, {3,6} };
int B [K][N] = { { 8,7,6}, {5,4,3} };
int C [M][N];

struct v {
    int i; /* row */
    int j; /* column */
};

void *runner(void *param); /* the thread */
int main(int argc, char *argv[]) {

    int i,j,v,s, count = 0;
    int num_threads=6;
```

```

pthread_t p[num_threads];
cpu_set_t my_set;          /* Define your cpu_set bit mask. */
CPU_ZERO(&my_set);        /* Initialize it all to 0 */
    CPU_SET(3 , &my_set);
sched_setaffinity(0, sizeof(cpu_set_t), &my_set);
    cpu_set_t set;
    int parentCPU, childCPU;
    int nloops, k;

    if (argc != 4) {
        fprintf(stderr, "Usage: %s parent-cpu child-cpu num-loops\n",
            argv[0]);
        exit(EXIT_FAILURE);
    }

    parentCPU = atoi(argv[1]);
    childCPU = atoi(argv[2]);
    nloops = atoi(argv[3]);

    CPU_ZERO(&set);

    switch (fork()) {
    case -1:      /* Error */
        errExit("fork");

    case 0:      /* Child */
        CPU_SET(childCPU, &set);

        if (sched_setaffinity(getpid(), sizeof(set), &set) == -1)
            errExit("sched_setaffinity");
    }

```

```

        for (k = 0; k < nloops; k++)
            getppid();

        exit(EXIT_SUCCESS);

default:    /* Parent */
            CPU_SET(parentCPU, &set);

            if (sched_setaffinity(getpid(), sizeof(set), &set) == -1)
                errExit("sched_setaffinity");

            for (k = 0; k < nloops; k++)
                getppid();

            wait(NULL);    /* Wait for child to terminate */

    }

wait;
for(i = 0; i < M; i++) {
    for(j = 0; j < N; j++) {
        //Assign a row and column for each thread
        struct v *data = (struct v *) malloc(sizeof(struct v));
        data->i = i;
        data->j = j;
        /* Now create the thread passing it data as a parameter */
        pthread_t tid;    //Thread ID
        pthread_attr_t attr; //Set of thread attributes
        //Get the default attributes

```

```

        pthread_attr_init(&attr);
        //Create the thread
while (i<4)
    {
        pthread_create(&tid,&attr,runner,data);
        i++;
    }
    //Make sure the parent waits for all thread to complete
    pthread_join(tid, NULL);
    count++;
}

//Print out the resulting matrix
for(i = 0; i < M; i++) {
    for(j = 0; j < N; j++) {
        printf("%d ", C[i][j]);

    }
    printf("\n");
}

//The thread will begin control in this function
void *runner(void *param) {
    struct v *data = param; // the structure that holds our data
    int n, sum = 0; //the counter and sum

    //Row multiplied by column
    for(n = 0; n< K; n++){

```



```
        sum += A[data->i][n] * B[n][data->j];  
    }  
    //assign the sum to its coordinate  
    C[data->i][data->j] = sum;  
  
    //Exit the thread;  
    pthread_exit(0); }
```

Graphical Representation

System monitor is an in build tool in linux is used for monitoring load on cpu cores

SINGLE CORE

Figure 4.3 - Graph of matrix multiplication on Single Core

MULTICORE

Figure 4.4 - Graph of matrix multiplication running on dual core

CHAPTER 5 - MULTITHREADING ON RASPBERRY PI 2

5.1 Hardware

Raspberry pie is a portable, powerful , and mini computer. The board length is only 85mm and width is only 56mm.Its size only as big as a credit card but it is a capable little PC. It can be used for many of the things that your desktop PC does, like high-definition video, spreadsheets, word-processing, games and more. Raspberry Pi also has more wide application range, such as music machines, parent detectors to weather stations, tweeting birdhouses with infra-red cameras, lightweight web server, home automation server, etc

Figure 5.1 - Raspberry pi 2

5.1.1 Specifications

- SOC: Broadcom BCM2836 (CPU, GPU, DSP, SDRAM)
- CPU: 900 MHz quad-core ARM Cortex A7 (ARMv7 instruction set)
- GPU: Broadcom VideoCore IV @ 250 MHz
- More GPU info: OpenGL ES 2.0 (24 GFLOPS); 1080p30 MPEG-2 and VC-1 decoder (with license); 1080p30 h.264/MPEG-4 AVC high-profile decoder and encoder
- Memory: 1 GB (shared with GPU)

- USB ports: 4
- Video input: 15-pin MIPI camera interface (CSI) connector
- Video outputs: HDMI, composite video (PAL and NTSC) via 3.5 mm jack
- Audio input: I²S
- Audio outputs: Analog via 3.5 mm jack; digital via HDMI and I²S
- Storage: MicroSD
- Network: 10/100Mbps Ethernet
- Peripherals: 17 GPIO plus specific functions, and HAT ID bus
- Power rating: 800 mA (4.0 W)
- Power source: 5 V via MicroUSB or GPIO header
- Size: 85.60mm × 56.5mm
- Weight: 45g (1.6 oz)

5.2 Installing operating system images on Linux

The Raspbian “Wheezy” operating system is the standard operating system to run on the Raspberry Pi. It has very good integration with the hardware and comes pre-loaded with a graphical user interface and development tools. This will come in very handy if you are not too experienced in a linux environment.

- Run `df -h` to see what devices are currently mounted.
- The left column gives the device name of your SD card; it will be listed as something like `/dev/mmcblk0p1` or `/dev/sdd1`. The last part (p1 or 1 respectively) is the partition number but you want to write to the whole SD card, not just one partition.
- Run `umount /dev/sdd1`, replacing `sdd1` with whatever your SD card's device name is (including the partition number).
- In the terminal, write the image to the card with the command below, making sure you replace the input file `if=` argument with the path to your `.img` file, and the `/dev/sdd` in the output file `of=` argument with the right device name.

```
dd bs=4M if=2016-09-23-raspbian-jessie.img of=/dev/sdd
```

- Also note that if you are not logged in as root you will need to prefix this with `sudo`.
- Run `sync`; this will ensure the write cache is flushed and that it is safe to unmount your SD card.

Figure 5.2 - Raspbian OS

5.3 Installing htop and gcc

htop

On top of the information that top provides, htop additionally shows your usage per CPU, as well as a meaningful text graph of your memory and swap usage right at the top.

To install htop simply use

```
sudo apt-get install htop
```

Once installed you can start it using

```
htop
```

Figure 5.3 - Htop in Raspberry pi 2

GCC

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. Originally named the GNU C Compiler,

when it only handled the C programming language, GCC 1.0 was released in 1987. It was extended to compile C++.

Figure 5.4 - GCC

5.4 Isolating Cpu cores in Raspberry pi 2

To isolate cpu cores in pi 2 we have to edit /boot/cmdline.txt as

```
root=/dev/mmcblk0p2 rw rootwait console=ttyAMA0,115200 console=tty1  
selinux=0 plymouth.enable=0 smsc95xx.turbo_mode=N dwc_otg.lpm_enable=0  
kgdboc=ttyAMA0,115200 elevator=noop isolcpus=2,3
```

After Rebooting you can check the result with “top” command.


```
Gcc dynamic.c -o dynamic -lpthread
```

- This will create an executable file , run that file using.
Time ./dynamic
- This will give the output and time taken to run the code.
- Open another terminal and run htop to view the cpu load.

Figure 5.6 - Running matrix multiplication on Single Core

5.5.1 Graphical Representation

rCpu is a minimal web server for remote CPU monitoring (on Linux). Everything is embedded in the executable, so you just need to run the program. The program will automatically adjust the display according to the number of CPU cores.

How to build and run

On the Raspberry Pi (and other Debian based systems probably)

You should be able to do this:

```
git clone https://github.com/davidsblog/rCPU
```

```
cd rCPU/rCPU/
```

```
sudo make install
```

which will build everything and install it as a service (it will run at system start-up). The server will run on port 8111. That means you can view the graphs by visiting <http://192.168.7.2:8111/> (you need to substitute your Raspberry Pi's IP address).

Figure 5.7 - Single Core Graphical Representation

5.6 Running Matrix multiplication on dual core on pi 2

For running matrix multiplication on single core open terminal and goto the directory where .c file is there and follow the steps.

- Open Terminal.
- Go to directory where .c file is located.

Cd /home/jntuh/Downloads

- Compile .c file using GCC compiler.

```
Gcc dynamicnew.c -o dynamicnew -lpthread
```

- This will create an executable file , run that file using.

Time ./dynamic 2 3 100000000

(Where 2 is Parent Cpu , 3 is Child Cpu and number of loops)

- This will give the output and time taken to run the code.
- Open another terminal and run `htop` to view the cpu load.

Figure 5.7 - Running matrix multiplication on Dual Core

5.6.1 Graphical Representation

Figure 5.7 - Dual Core Graphical Representation

5.7) Test Results

2.33GHz quad-core i3 Processor

Matrix Size	Single Core	Dual Core	Speed Up By
1000 * 1000	31 seconds	15 seconds	2X
500 * 500	18 seconds	9.7 seconds	2X

900MHz quad-core ARM CORTEX (Raspberry pi 2)

Matrix Size	Single Core	Dual Core	Speed Up By
1000 * 1000	3min 28sec	2min 05sec	2X
500 * 500	1min 15sec	31sec	2X

Table 5.7 Test results of multi threading on pi 2

CHAPTER 6-CONCLUSION AND FUTURE SCOPE

CONCLUSION

The following are the conclusions:

- The process of multiplying 1000*1000 matrix by creating threads which is a concept of MULTITHREADING was implemented.
- The Matrix multiplication of size 1000*1000 with threads was implemented on Acer quad-core i3 processor and ARM CORTEX 7 quad-core processor of Raspberry pi 2.
- In i3 processor and ARM CORTEX 7 processor of Raspberry pi 2,two cpu cores out of four cpu cores are isolated and matrix multiplication of 1000*1000 is made to run on this isolated cores.
- Multitasking is implemented by running four different tasks like VLC , Firefox , Cheese and Rhythmbox on particular cpu core of Acer i3 processor and observing the latency time taken by applications.

FUTURE SCOPE

Implementing Matrix multiplication in pipeline method can be considered as future work. In matrix multiplication the two operations addition and multiplication are implemented.The operation of multiplication is implemented on one core and the value of it is stored in a pipeline register Then addition operation can be implemented on the other core with stored values.

CHAPTER 6-REFERENCES

- [1] <https://www.cs.princeton.edu/courses/archive/fall10/cos597C/docs/threads1.pf>
- [2] <https://wiki.ubuntu.com/Kernel/KernelBootParameters>
- [3] https://www.quora.com/How-I-can-run-C-files-on-specific-CPU-core-in-Linux/answer/Nayan-Deshmukh-3?_snid3=435464801&_nsrc=2&_filter
- [4] <https://computing.llnl.gov/tutorials/pthreads/#Exercise1>.
- [5] <http://www.runeaudio.com/forum/the-new-4-core-raspberry-pi-t862-40.html>
- [6] <https://www.raspberrypi.org/blog/visualising-core-load-on-the-pi-2/>