



UMiMAP

**Embedded Operating System Optimization
for Face Recognition System**

by

Shuhaizar Bin Daud

A thesis submitted
In fulfilment of the requirements for the degree of
Master of Science (Computer Engineering)

**School of Computer & Communication Engineering
Universiti Malaysia Perlis**

2010

UNIVERSITI MALAYSIA PERLIS


DECLARATION OF THESIS

Author's full name : SHUHAIZAR BIN DAUD
Date of birth : 10.03.1983
Title : EMBEDDED OPERATING SYSTEM OPTIMIZATION
FOR FACE RECOGNITION SYSTEM
Academic Session : 2010/2011

I hereby declare that the thesis becomes the property of Universiti Malaysia Perlis (UniMAP) and to be placed at the library of UniMAP. This thesis is classified as :

- CONFIDENTIAL** (Contains confidential information under the Official Secret Act 1972)*
- RESTRICTED** (Contains restricted information as specified by the organization where research was done)
- OPEN ACCESS** I agree that my thesis is to be made immediately available as hard copy or on-line open access (full text)

I, the author, give permission to the UniMAP to reproduce this thesis in whole or in part for the purpose of research or academic exchange only (except during a period of ____ years, if so requested above).



SIGNATURE

830310-02-5303
(NEW IC NO. / PASSPORT NO.)

Date : 31.03.2010

Certified by: 

SIGNATURE OF SUPERVISOR

P.M. DR. R. BADLISHAH AHMAD
NAME OF SUPERVISOR

Date : 31.03.2010

NOTES : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organization with period and reasons for confidentiality or restriction.

ACKNOWLEDGEMENT

I would like to extend my highest gratitude to my supervisor Assoc. Prof. Dr. R. Badlishah Ahmad for all his guidance over the years I spent in this research. I would also like to thank my co-supervisor Assoc. Prof. Dr. Rizon Muhamed Juhari for his support and help. I am thankful to all the researchers & friends in Embedded Computing Cluster that in more than one way helped me pull it off together. I would also like to thank my family and my wife and child for their kind support, patience and encouragement throughout these years.

The time I've spent on this research has been a great opportunity for me to learn and apply the knowledge I have acquired during my studies. I am also indebted to all the lecturers and staffs of the School of Computer Engineering and Communications for their help and friendship.

THANK YOU!

TABLE OF CONTENTS

	Page
Table of contents	ii
List of table	vii
List of figures	viii
Abstrak	xi
Abstract	xii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement.....	2
1.2 Project Aim.....	4
1.3 Project Motivation.....	4
1.4 Project Objectives.....	5
1.5 Project Scope	5
1.6 Thesis Outline.....	6
CHAPTER 2 LITERATURE REVIEW	8
2.1 Introduction	8
2.2 Linux	8
2.2.1 The Linux Kernel	9
A. Linux 2.4 Kernel.....	11
B. Linux 2.6 Kernel.....	11
C. Comparison of the 2.4 Kernel and 2.6 Kernel	13

2.3	Linux GNU C Compiler Collection (GCC)	18
2.4	Embedded System	19
2.4.1	Single Board Computers	20
	A. Advantech PCM-9375 SBC.....	20
	B. Technologic Systems TS-5500 SBC	22
2.4.2	Embedded Operating System.....	23
2.4.3	Embedded Linux	24
2.4.4	Embedded Linux Booting Sequence.....	25
2.4.5	Examples of Embedded Linux Operating System	28
	A. BlueCat Embedded Linux.....	29
	B. PeeWeeLinux.....	30
	C. Technologic Systems TS-Linux	31
2.5	Face Recognition System	31
2.5.1	Face Recognition Through Iris Detection.....	34
	A. Face Database Modeling.....	35
	B. Face Detection Stages.....	36
2.5.2	Histogram Equalization.....	40
	CHAPTER 3 METHODOLOGY	42
3.1	Introduction	42
3.2	Available Development Paths	43
3.3	Host System and Target Development Setup.....	45
3.4	Host System Setup.....	48
3.5	Linux Main System Toolchain	51
3.6	Linux GNU Compiler Collection (GCC)	52

3.6.1	GCC Compiler Optimization	53
3.6.2	GCC General Optimization Flags	54
A.	1 st Level Optimization	57
B.	2 nd Level Optimization	60
C.	Code Size Optimization (Level 2.5)	61
D.	3rd Level Optimization	62
3.6.3	GCC Architectural Optimization Flags.....	63
3.7	Bootstrapped Compilation Process.....	65
3.8	Development Directory Setup	68
3.9	Prototyping & Evolution Development.....	70
3.10	Prototype and Disk Image Storage	72

CHAPTER 4 DESIGN, TESTING & VALIDATION..... 73

4.1	Introduction	73
4.2	Target Operating System Design.....	74
4.2.1	System Components Selection.....	74
4.2.2	Kernel Configuration and Selection.....	76
4.2.3	Bootloader Selection & Setup.....	78
4.2.4	Filesystem	80
4.2.5	Root Filesystem Structure	83
4.3	Hardware Testing Platform	85
4.4	Performance Evaluation	86
4.5	Power Measurement Platform	89
4.6	Memory Usage	91

CHAPTER 5 RESULT & DISCUSSION	93
5.1 Introduction	93
5.2 Developed Prototypes.....	93
5.3 Software Based Test Results	96
5.3.1 Booting Performance	96
5.3.2 Code Execution Performance.....	98
A. Binary compiled with no compiler optimization (Level 0)	99
B. Binary compiled with 1 st level compiler optimization.....	100
C. Binary compiled with 2 nd level compiler optimization.....	101
D. Binary compiled with 3 rd level compiler optimization	102
E. Summary of Code Execution Performance Test.....	103
5.3.3 Binary Efficiency	103
5.4 Hardware Based Test Results.....	106
5.4.1 System Memory Footprint	107
5.4.2 Power Measurement.....	107
A. Voltage Drop across Shunt Resistor	108
B. Current Draw	111
C. Power Consumption.....	114
D. Energy Usage.....	118
 CHAPTER 6 CONCLUSION	 122
6.1 Introduction	122
6.2 Future Works	123
6.3 Contribution.....	124

REFERENCES	125
-------------------------	-----

APPENDICES

Appendix A: Raw data of algorithm execution performance	130
Appendix B: Raw data of power measurement for Prototype 1	134
Appendix C: Raw data of power measurement for Prototype 2	136
Appendix D: Raw data of power measurement for Prototype 3	138
Appendix E: Raw data of power measurement for Prototype 4	140
Appendix F: Published papers & awards received	142

© This item is protected by original copyright

LIST OF TABLES

Table No.		Page
2.1	Kernel repository for different versions	10
2.2	Available commercial face recognition systems	33
3.1	Development Architecture Test Results	48
3.2	Operating System Evaluation Result	51
3.3	Specific optimization routine invoked by GCC general optimization flag	57
3.4	First Level Optimization Description	58
3.5	Second Level Optimization Description	60
3.6	Third Level Optimization Description	63
4.1	Linux Bootloader Comparison	79
4.2	Filesystem Characteristics	81
4.3	Root Filesystem Directories According to FHS	84
4.4	Test condition for measurement process	90
5.1	Prototype details	94
5.2	Binary size for test algorithm compiled without optimization	104
5.3	Binary size for test algorithm compiled with 1st level compiler optimization	104
5.4	Binary size for test algorithm compiled with 2nd level compiler optimization	104
5.5	Binary size for test algorithm compiled with 3rd level compiler optimization	105
5.6	System footprint size in idle mode after boot up	107

LIST OF FIGURES

Figure No.		Page
2.1	Linux kernel numbering scheme	10
2.2	Average response time for 2.4 and 2.6 kernel under load	15
2.3	System worst case response time between 2.4 and 2.6 kernel under load	16
2.4	Advantech PCM-9875 Single Board Computer	21
2.5	Linux booting sequence and stages	26
2.6	Extended face template	35
2.7	Coordinate system of the extended face template, T_E	36
2.8	Face recognition using iris detection method	37
2.9	Positions of irises detected by the iris detection algorithm	38
3.1	The host and target in a linked development setup	45
3.2	The host and target in a removable development setup	46
3.3	The host/target setup in a standalone development setup	47
3.4	Bootstrapped compilation process	66
3.5	Development host partition setup separating the development partitions in prototype1 and prototype2 partition	69
3.6	QEMU emulating hardware boot during the prototype testing stage	70
3.7	Boot failure on the test hardware	71
4.1	Default bootloader setup on a normal system	80
4.2	Test platform partition configuration	80
4.3	ext2 filesystem creation using mkfs	82
4.4	ext3 journaling filesystem creation using the mkfs.ext3 command	83
4.5	Root filesystem setup on the prototype following FHS ruling	85
4.6	Picture of the embedded systems test platform	86
4.7	Portion of the histogram equalization testing script	87
4.8	Portion of the measurement result time.txt showing execution measurement of test algorithms	88

4.9	Portion of the validation file output.txt showing the algorithm processing results	89
4.10	Circuit diagram for the power measurement platform	90
4.11	Sample output from the top utility	92
4.12	System CPU and memory utilization information from the top utility	92
5.1	Developed prototypes booting time	97
5.2	Developed prototypes booting performance compared to commercial operating systems	98
5.3	Processing time graph for histogram equalization and face recognition algorithm	99
5.4	Processing time graph for histogram equalization and face recognition algorithm compiled with 1st level compiler optimization	100
5.5	Processing time graph for histogram equalization and face recognition algorithm compiled with 2nd level compiler optimization	101
5.6	Processing time graph for histogram equalization and face recognition algorithm compiled with 3rd level compiler optimization	102
5.7	Voltage drop across shunt resistor during testing for Prototype 1	109
5.8	Voltage drop across shunt resistor during testing for Prototype 2	109
5.9	Voltage drop across shunt resistor during testing for Prototype 3	110
5.10	Voltage drop across shunt resistor during testing for Prototype 4	110
5.11	Current draw of the system during testing for Prototype 1	111
5.12	Current draw of the system during testing for Prototype 2	112
5.13	Current draw of the system during testing for Prototype 3	112
5.14	Current draw of the system during testing for Prototype 4	113
5.15	Minimum, maximum and average current draw for each prototype during measurement	114
5.16	Power consumption of the system during testing for Prototype 1	115
5.17	Power consumption of the system during testing for Prototype 2	115
5.18	Power consumption of the system during testing for Prototype 3	116
5.19	Power consumption of the system during testing for Prototype 4	116
5.20	Minimum, maximum and average power consumption for each prototype during measurement	117

5.21	Energy usage of the system during testing for Prototype 1	118
5.22	Energy usage of the system during testing for Prototype 2	119
5.23	Energy usage of the system during testing for Prototype 3	120
5.24	Energy usage of the system during testing for Prototype 4	120
5.25	Total energy usage of the prototypes after completing measurement process	121

© This item is protected by original copyright

PENGOPTIMUMAN SISTEM OPERASI TERBENAM UNTUK SISTEM PENGECAMAN WAJAH

ABSTRAK

Pengoptimuman oleh pengkompil telah dibuktikan berupaya untuk meningkatkan prestasi dan memperbaiki kecekapan kod-kod binari bagi aturcara. Kebaikan pengoptimuman terhadap kod aturcara semasa proses pengkompilan adalah jelas terhadap prestasi sistem, prestasi memori, akses cakera, kecekapan penggunaan tenaga dan dalam hampir semua aspek lain sistem. Ini secara khususnya menarik bagi sistem terbenam di mana prestasi dan kecekapan begitu ditekankan. Penyelidikan ini bertujuan bagi mengimplementasikan teknik pengoptimuman pengkompil terhadap proses pembangunan sistem operasi dan mengkaji kesan implementasi tersebut terhadap prestasi dan kecekapan sistem. Thesis ini menggariskan teknik-teknik untuk membangunkan sistem operasi khas berasaskan teras Linux dan cara untuk mengimplementasikan teknik pengoptimuman semasa proses pembangunan dan kompilasi. Memfokuskan sistem pengecaman wajah yang diimplemen di atas sistem terbenam, kesan pengoptimuman dikaji dari segi kesannya terhadap prestasi perisian dan juga perkakasan. Dari segi perisian; kajian dilakukan terhadap prestasi proses *booting*, kelajuan pelaksanaan kod dan kecekapan kod binari oleh prototaip yang dibangunkan dengan tahap pengoptimuman yang berbeza. Dari perspektif perkakasan pula, kajian dilakukan terhadap saiz penggunaan memori, penggunaan arus elektrik dan kecekapan tenaga. Keputusan yang diperolehi menunjukkan bahawa pengoptimuman pengkompil bukan hanya bermanfaat apabila diterapkan pada kod program, tapi juga menunjukkan kesan baik apabila diterapkan pada sistem operasi.

EMBEDDED OPERATING SYSTEM OPTIMIZATION FOR FACE RECOGNITION SYSTEMS

ABSTRACT

Compiler optimizations have been proven to be beneficial in improving performance and efficiency of binary codes across different type of operating systems. Improvement from optimization of program code during compilation are obvious to the system performance, memory performance, disk access, energy efficiency and nearly all aspect of the system. This is particularly attractive to an embedded system where performance and efficiency are seriously considered. This research focuses on implementing those proven compiler optimizations to the development of an embedded operating system and studying the effect of such implementation to the performance and efficiency of the system. This thesis outlines the methods available to develop a custom operating based on the Linux kernel and ways to implement such optimization during the compilation and development process. Focusing on face recognition systems implemented on embedded Single Board Computers, optimization effects to an embedded operating system are studied through software and hardware perspective. From the software perspective; booting performance, code execution performance and binary efficiency of different prototypes developed with varying level of compiler optimization are tested and examined. From hardware side; system memory footprint, current draw, power consumption and energy usage of the different prototypes are tested and measured. Results obtained in this thesis shows that compiler optimization are beneficial not only when applied to the program code, but also have significant effects when applied to the operating system.

CHAPTER 1

INTRODUCTION

An embedded system is a special-purpose computer system that is designed to perform very small sets of designated activities. Embedded systems date back as early as the late 1960s where they used to control electromechanical telephone switches. The first recognizable embedded system was the Apollo guidance computer developed by Charles Draper and his team (Raghavan et.al, 2006). Later they found their way into the military, medical sciences, and the aerospace and automobile industries. Today they are widely used to serve various purposes; some examples are the following:

- i. Network equipment such as firewall, router, and switches.
- ii. Consumer equipment such as MP3 players, cell phones, PDAs, digital cameras, camcorders and home entertainment systems like TiVo.
- iii. Household appliances such as microwaves, washing machines, and televisions set top boxes.
- iv. Mission-critical systems such as satellites and flight control.

Embedded systems are playing important roles in our everyday lives, even though they might not necessarily be visible. Some of the embedded systems we use everyday control the menu system on television, the timer in a microwave oven, a cellphone, an MP3 player or any other device with some amount of intelligence built-in. In fact, recent data poll shows that embedded computer systems currently outnumber humans in the US (Wikibooks, n.d.).

Unlike fully featured personal computers, embedded systems require a different kind of operating system different from the larger operating system designed for ordinary computers. Since embedded system are bound with small memory footprint with a fraction of the processing power usually available in desktop personal computers, it requires a different breed of operating system specifically built for embedded system environment.

This research focuses on developing an embedded operating system based on the newer Linux kernel 2.6 for face recognition systems. Compiler optimization sequences are also implemented during the development phases and the effects of such optimizations to the performance and power requirement of the system are studied in this research.

1.1 Problem Statement

Embedded systems require a different breed of operating system different from the ones used by the normal personal computers. Implementing minimal memory size, small storage size and with a fraction of the computing power, an embedded system

would significantly benefit from a streamlined operating system developed with embedded system in mind.

To the best of our knowledge, there is very little research that have been done on the effects of operating system miniaturization on the speed and performance compared to fully featured operating system for an embedded system. A streamlined operating system developed for a specific purpose might be beneficial for the entire system performance and could theoretically lead to a better performing system.

Larger operating system have too much components not needed for embedded system operation. Much of these components can be removed to lower the operating system size. After removal, the size of the operating system will be smaller thus reducing the required storage size and could theoretically help in improving the entire system speed. An operating system specifically built from the ground up to target a particular embedded device could prove to be a better option because the entire system could be kept at a minimal size and extra features only added when required. By controlling the operating system function and operation, the operating system itself could be developed to provide only the specific functions required by the target system and other unnecessary functions and components could be removed.

With the introduction of the newest Linux kernel (version 2.6), major improvements that could not be integrated into the previous kernel (version 2.4) could now be put into practice. Though it has been 4 years since the new generation kernel has been introduced to the community, adoption by the embedded community has been slow with only a handful of embedded operating system that make use of the new kernel.

Compiler optimizations have been proven by various studies to be beneficial for system performance, improving memory performance, whilst reducing power

consumption. It also helps in reducing footprint size of compiled binaries (Lombardo, 2002) thus making the entire system smaller in size. Though have been proven numerous time to be beneficial, little have been studied the effects of such optimizations when applied to an operating system.

1.2 Project Aim

To develop an embedded operating system based on the Linux kernel for improvement of face recognition system process.

1.3 Project Motivation

Compiler optimization have been proven numerous time to be beneficial in improving system performance (Mehis, 2002 & Wolfe, 2004) and memory subsystem performance (Kandemir et.al, 2000, Pan, 2004 & Marwedel, 2002), reducing power consumption (Chakrapani et.al, 2001 & Seng, 2003), reducing memory system subsystem power consumption (Kim et.al, 2000 & Kandemir, 2001), improving I/O performance (Kandemir et.al, 1999) and are beneficial to nearly all aspects of the system.

Though a lot of studies have been done to the hardware and software side, little attention have been given to the middleware part of a working system which is the operating system that handles all the system execution and processing. Little have been studied on how the operating system affects the entire system performance and how

compiler optimizations when applied to an operating system would affect the system and subsystem performance. This study focuses on applying compiler optimizations which are known to be a great benefit to system performance to the operating system during development and studies the results of such optimization to the entire system performance.

Face recognition systems are selected as part of the implementation and evaluation process mainly because it is one of the most resource taxing real time systems available. The implementation of both pre-processing and post-processing algorithms on a face recognition system transformed it to a very resource hungry beast and requires a considerable amount of processing time to complete. This allows any improvement made by optimization routines to reflect in the final processing time. A properly optimized face recognition system should trim down the processing time required to obtain results thus making the system speedier with better response time.

1.4 Project Objectives

- i. To develop an optimized Linux variation for embedded system.
- ii. To evaluate the embedded operating system for face recognition system.

1.5 Project Scope

- i. Development of the operating system is built around the general Linux Kernel from the Kernel Source Tree.

- ii. Implementation of the system is done on a selected Single Board Computer consisting of an Advantech PCM9375F and a Technologic Systems TS5500.
- iii. Performance evaluation and benchmark are done using image processing algorithm consisting of histogram equalization algorithm and face recognition through iris detection algorithm.

1.6 Thesis Outline

This thesis are composed of 6 chapters and is organized as follows:

- i. Chapter 1 presents the overview and problem statement that clarifies the driving force and motivating aspect, together with objectives, scope and thesis layout.
- ii. Chapter 2 presents the literature review for the project consisting of brief introduction towards the Linux and the Linux kernel, embedded systems and Single Board Computers (SBC), and face recognition systems.
- iii. Chapter 3 explains the methodology process used to develop the embedded operating system prototype and discusses the compiler optimizations implemented within.
- iv. Chapter 4 describes the design of the embedded operating system together with critical system components selected for implementation and the testing and evaluation platform used.
- v. Chapter 5 outlines the result obtained from the experimental prototype and the efficiency of the developed system.

- vi. Chapter 6 concludes the thesis by summarizing the most important ideas and conclusions. In the end the contribution and possible directions for future work are discussed.

© This item is protected by original copyright

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter summarizes the research and reading done in developing the upcoming prototype. In this chapter, basic operating system concepts and development methods are presented in order to provide a basic understanding on the development phases. Vital areas and problem stages are identified at the end of the review thus ensuring a successful build of the target operating system.

2.2 Linux

Linux is an operating system firstly created by Linus Benedict Torvalds in 1991 while he was still in the University of Helsinki. The operating system is built entirely from the bottom up by a collaboration of developers from all over the world. The project started to pick up speed when Linus Torvalds posted his idea and his

works in a newsgroup, attracting contribution from thousands of developers worldwide (Raghavan et.al, 2006).

Since the first posting releasing the first Linux version to the world, Linux has matured into a full-fledged operating system capable of delivering the reliability, performance and features of the best commercial operating systems on the market (Hallinan, 2007). The Linux operating system is distributed under an open source agreement which grants users access to the operating system source code. This makes the OS a very viable option to be used as a platform for conducting research on operating system.

With the source code to the Linux kernel available for users and developers, a few variant of the operating system quickly spawned into the market. These variants, called a *distribution*, are built on the same Linux base though sometimes having different focus and employing different software package and configurations. Some of the more popular distributions currently available includes Fedora, SUSE, Debian and Slackware.

2.2.1 The Linux Kernel

The Linux kernel is the single most important component in the Linux kernel. The kernel handles every hardware interfaces that connects the hardware to the software layer. The kernel exists as the core component in all Linux systems.

The main repository for the Linux kernel exists at the Kernel Repository site located at <http://www.kernel.org> and tracks each and every release of the kernel. This repository however, doesn't track the different kernel tree for different architectures

that are supported by Linux. Table 2.1 lists alternative locations for the appropriate kernel for other architectures.

Table 2.1: Kernel repository for different versions.

Processor architecture	Appropriate kernel location	Available download means
x86	http://www.kernel.org/	ftp, http, rsync
ARM	http://www.arm.Linux.org.uk/developer/	ftp, rsync
PowerPC	http://penguinppc.org/	ftp, http, rsync, bitkeeper
MIPS	http://www.Linux-mips.org/	cvcs
SuperH	http://Linuxsh.sourceforge.net/	cvcs
M68k	http://www.Linux-m68k.org/	ftp, http

The Linux kernel has a numbering scheme system that numbers the different kernel releases to differentiate between kernel sources intended for development and experimental work and the versions intended as a stable, production-ready kernels. The numbering scheme consists of a major version number, a minor version number followed by a sequence number. Generally if the minor version number is even, it denotes a production kernel; and if it's odd, it denotes a development kernel (Hallinan, 2007). Figure 2.1 illustrates the kernel numbering scheme used to differentiate between major and minor revisions.

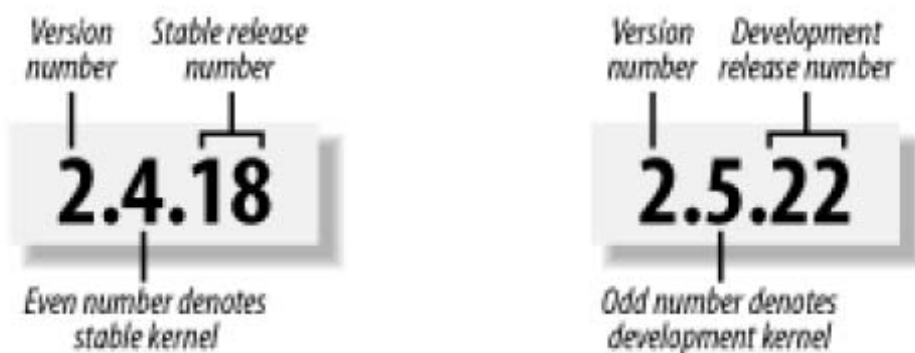


Figure 2.1: Linux kernel numbering scheme.