A LINUX MICROKERNEL BASED ARCHITECTURE FOR OPENCV IN THE RASPBERRY PI DEVICE

Stevan O. N. Silva, Luciano Silva
Graphics Processing and Digital Media Laboratory – College of Computing and Informatics
Universidade Presbiteriana Mackenzie, São Paulo – SP – Brasil
stevan.ons@gmail.com, luciano.silva@mackenzie.br

Abstract.
This work presents the development of an embedded architecture based on the Linux system and the OpenCV library, focused on the effective use of the hardware, through the selection of specific features required by this library and aiming for performance improvement. The approach goes from Linux kernel configuration to the inclusion of required libraries, the root filesystem structure, system utilities and startup scripts. The concept is applied to a Raspberry PI single board computer and some tests were run comparing the modified Linux kernel with the original one.

Keywords: Embedded Vision, Embedded Image Processing, Embedded OpenCV, Embedded Linux, Raspberry PI.

1. INTRODUCTION

Embedded systems are computer systems used for specific functions, usually being part of a larger system and making control by means of an embedded software. They are present in consumer products and industrial and military sectors such as smartphones, digital cameras, routers, security systems and access controls, automotive onboard computers, healthcare monitoring systems, avionics among others. Time constraints are common in these systems - concept known as Real-Time Computing - in which certain input stimuli should produce reactions within a pre-defined time limit, so that the system operates correctly.

Advances in Computer Vision have introduced a set of useful features such as pattern recognition, localization and counting of objects and people, face recognition and tracking. The OpenCV library allows these features be implemented on computers with relative ease, but general purpose hardware has been preventing a large number of applications, making them too expensive, with large physical size, high power consumption and low tolerance to vibration and changes in environmental temperature.

Thus, the use of embedded systems becomes essential to enable numerous applications, sometimes reducing costs of production of products for the end consumer through the use of specialized hardware and low cost, sometimes in increasing portability, employing a low energy hardware makes possible the use of batteries as a form of power. In addition, to increasing the possibility of commercial applications portability allows researchers to use devices in field experiments or medical analysis images be embedded in portable equipment or systems of small sizes. Other applications such as intelligent factories, power plants or offshore platforms, which require a higher tolerance to the environment and low maintenance equipment, can be made possible with the use of hardware for industrial embedded systems.

Within this context, this paper proposes the development of an embedded architecture based on the Linux operating system and the OpenCV library, under the Raspberry PI hardware, stressing this hardware by selecting specific features of the system that provide support to applications based on this library in order to minimize the processing overhead and reduce the use of computing resources such as main and storage memories.

The work aims to demonstrate that a system specialized for applications using OpenCV can be created without major challenges. The identification of the components needed to simplify this architecture enables the development of other embedded Linux based systems that use this library. Testing data can serve as metrics for performance of certain algorithms in embedded
systems and show the limitations imposed by these types of hardware on certain computationally expensive applications.

This work is organized in the following sections: Section 2 reviews the Linux Microkernel architecture; Section 3 brings the state-of-art in running OpenCV under embedded systems; Section 4 describes our proposed architecture; Section 5 presents and discusses tests and results; Section 6 shows a robotic prototype with the proposed architecture and, finally, Section 7 finishes the work presenting the final conclusions and further work.

2. LINUX MICROKERNEL ARCHITECTURE

Linux microkernel is part of a group of Unix operating systems (Unix-like) style, in accordance with the IEEE POSIX standards, open source, distributed under the GPL license, and free for personal or commercial use. Initially launched in 1991 by Linus Torvalds, is now rapidly developing with the support of large companies such as Fujitsu, Hitachi, HP, IBM, Intel, NEC, Novell and Oracle.

It is a very flexible system used in mobile devices, embedded systems, desktop computers, servers, mainframes and supercomputers. Provides support multiple hardware architectures and is the system that controls many of the everyday products like Android smartphones, routers, smart TVs, GPS devices, e-book readers, autonomous cars and refrigerators.

2.1. System Overview

The operating system is the most basic piece of software, running in core mode. In this sense, it has full access to all hardware and can execute any instruction that the machine is capable of performing. The rest of the software runs in user mode, in which only a subset of machine instructions is available [Tanenbaum, 2007].

The user mode software is the set of programs and utilities that work in user mode and are also found in this layer system libraries such as glibc. The kernel mode software is part of the responsibility for the control of the hardware and software for managing the user-mode operating system.

2.2. Kernel Structure

Linux uses a monolithic kernel, with a large, complex and self-manageable program consisting of several logically different components. This is conventional - most commercial systems derivatives from UNIX are monolithic. [Bovet 2005] decomposes the Linux System into seven subsystems based on their different functionalities as shown in Figure 1, which are described in more detail in subsequent sections.

![Figure 1: Linux kernel subsystems [Bovet, 2005].](image)

2.2.1 Linux System Call Interface

The System Call Interface (SCI) is a layer whose function is to allow user applications to make system calls to the kernel functionality provided in a standardized way, while maintaining control and system integrity. It is a thin layer that provides means to perform function calls from user space to kernel. As mentioned earlier, this interface can be architecture dependent, even within the same family of processors [Jones, 2007].

2.2.2 Process Management

The process manager is the subsystem that gives Linux the multiprocessing capability. It is responsible for the distribution of hardware resources to each of the running processes.

With the arrival of version 2.6, the Linux replaced its old O(n) scheduler by a new O (1) one, a new type of constant-time algorithm, which requires the same number of operations, regardless of how many tasks are active in system. This solved a long-standing problem of performance degradation when a large number of simultaneous tasks disputed the CPU. Since version 2.6.18, the Linux Kernel has been using the Completely Fair Scheduler scheduler (CFQ), making an enhancement of the scheduler (1) through the implementation of the Rotating Staircase Deadline Scheduler algorithm.

2.2.3 Memory Management

In most architectures, Linux works with 4KB pages, including both free memory management and support for mapping of real and virtual memory done by hardware through the Memory
Management Unit (MMU). One can also get Linux on systems with processors without MMU by compiling the kernel using uClibc library.

Linux also provides a resource or virtual memory swapping, which uses a hard disk partition to exchange portions of main memory, allowing occupied regions that are not being frequently used become available for allocation.

2.2.4 Virtual File System

The goal of the virtual file system layer is to allow different file systems to be unified into a single directory structure. Thus, both local disks and shared network drive, and even removable devices can be accessed transparently by applications.

Linux still uses the virtual file system as a channel of information and a way of changing kernel settings at runtime through the / proc and / sys.

2.2.5 Network Stack

The Linux network stack, is derived from BSD system, forming a set of layers, ranging from network device driver to the application layer. This modular approach allows drivers support different communication protocols, and applications can make use of different protocols transparently.

2.2.6 Device Drivers

The kernel interacts with input and output devices through device drivers. Drivers are included in the kernel and consist of data structures and functions that control one or more devices such as hard disks, keyboards, mice, monitors, network interfaces and devices connected to a SCSI bus. Each driver interacts with the remaining part of the kernel through a specific interface [Cesati and Bovet, 2005].

Linux implements the concept of loadable kernel modules (loadable kernel module), feature that allows one to extend the functionality of the kernel at runtime, by loading files of type KO (Kernel Object).

2.2.7 Architecture Dependent Code

Although Linux supports different types of architectures, certain parts of the core need to deal with features that are specific to each CPU as interrupts, Direct Memory Access (DMA), communication with the MMU, among others. To facilitate portability and maintenance of the system, this layer provides an abstraction of each specific type of hardware details, allowing a uniform implementation of the system on supported architectures. Therefore, It is the layer of the lowest level, providing to the kernel a direct communication with hardware.

2.2.8 Filesystem Hierarchy Standard

Linux, and other Unix systems, follows a directory structure that has undergone modifications in recent decades, and has been standardized by the Filesystem Hierarchy Standard Group, a group that defines a model of the structure of the root filesystem and rules for their use. The standards document directory structure can be found on the FHS website at http://www.pathname.com/fhs.

3. OPENCV ON EMBEDDED SYSTEMS

3.1. Embedded Systems

An embedded system is a computer system designed to perform specific functions within a larger system, often subject to constraints of real-time computing. It makes a contrast when compared to a personal computer and it is designed to be more flexible and to meet a wide range of needs of end users [Barr & Pasta, 2006]. In general, they are small computer systems or part of a bigger system. In most cases, they are designed to be minimal in one or more aspects of the production cost, physical size and power consumption.

Due to the huge variety of applications, it is difficult to define a standard model that describes the structure of hardware of embedded systems. However, some essential components can be found in most of these systems, especially in control systems [Marwedel, 2003].

3.2. The OpenCV Library

The OpenCV (Open Source Computer Vision Library) is a Computer Vision library written in C and C++ compatible with Linux, Windows and Mac OS X systems. It has been designed to be computationally efficient, with a strong focus on real-time applications and additionally it may take
advantage of CPUs with multiple cores [Bradski, & Kaehler, 2008].

The purpose of this library is to provide a simple computer vision infrastructure to prototype quickly sophisticated applications. It has over 2500 optimized algorithms, including both a set of classical algorithms and the state of the art algorithms in Computer Vision, which can be used for image processing, detection and face recognition, object identification, classification actions, traces, and other functions. The library is used extensively by companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, and startups area as Applied Minds, VideoSurf and Zeitera, and research groups and government.

3.3. OpenCV Code Portability

OpenCV is a library designed to be highly portable. Though born of a project from Intel, supporting early compilers Intel C++ and Microsoft Visual C++ on x86, the code is always written in a standardized way, allowing for possible use on other platforms. The current version (2.4.5), supports IA32, EM64T, IA64, PowerPC, ARM, MIPS architectures and further processing platform for CUDA GPUs.

3.4. Embedded Vision: OpenCV on Embedded Systems

The term embedded vision refers to the use of Computer Vision technology in embedded systems [Gregori 2012]. Computer Vision algorithms originally could only be implemented in expensive, bulky computer systems and high power consumption, limiting the use to a few applications such as factory automation and military equipments.

From the first decade of this century, however, there was a change in this direction. Due to the high level of integration of components into powerful processors, low cost and energy efficient, along with improvements in imaging sensors, it was possible to incorporate the capabilities of computer vision in a wide range of embedded system applications.

4. OpenCV Embedded in the Raspberry PI

This work identified and collected all components and settings required to build a Linux system, primarily geared to the implementation of algorithms based on the OpenCV library, with a focus on embedded systems.

The Linux kernel has been reconfigured, keeping only necessary components built into the kernel image. Other components that might be used during the execution were compiled as modules, so that they could be loaded and unloaded on demand. A reduced structure of system was created following the standards of the FHS, containing the necessary system utilities, supporting file handling, device configuration, network settings and support for runtime configuration. The structure still takes the required libraries for applications that use OpenCV. Startup scripts were rewritten to set up an environment for implementation of OpenCV and to start strictly necessary services.

For proof of concept, the system was deployed on a Raspberry Pi board, depicted in the Figure 2:

![Figure 2: The Raspberry Pi board. Source: www.raspberrypi.org](image)

Raspberry Pi is a Single Board Computer (computer on a single board) based on ARM, the same used by much of the embedded system architecture.

4.1. OpenCV in the Raspberry Pi Requirements

The main feature of this architecture is to provide a framework for the implementation of applications based on OpenCV library, focused on embedded systems using a Raspberry Pi board. To use the processing power of the BCM2835 SoC, the kernel might be optimized in order to reduce the overhead caused by the large systems required functionality.
To support image and video devices, the architecture must provide compatibility with the USB bus drivers and include the most common cameras. Memory usage by the operating system and services should be minimized, leaving the most amount of free memory for applications. The architecture should also make use of energy saving features available on hardware, such as using dynamic CPU clock, extending the time of use of systems by making use of batteries as a power source.

4.2 Deployment Diagram

The developed system is based on a custom build of Linux 3.6.11, providing support to the essential functions of an operating system. The core includes a wide hardware support such as I/O controllers, USB, GPIO and video cameras devices, as shown in the Figure 3 (top of this page).

On this core there are the Core Libs, a set of libraries that provide basic support for downstream applications of C and C++. In addition to this support, these libraries provide the primary communication pathways among core and application, through system calls.

Providing support for OpenCV applications, the version 2.3.1 is integrated into the system as the user library. Applications can use low-level features such as DMA access of devices to capture images through the core libs and functions of Image Processing and Computer Vision OpenCV library. They can also send signals to actuators through the General Port for I/O (GPIO), part of Raspberry PI board.

4.3 Environment Configuration

For the developed system, it was compiled a version of the Linux kernel targeted at Broadcom BCM2835 SoC. As the specific drivers of this component are not included in the source code distributed officially by the Linux Foundation, it was necessary to obtain the kernel source code with a set of patches in order to get this support. A system of reduced root file was also created, containing a standard Unix-like based on the FHS, with the inclusion of essential system libraries and utilities systems.

A startup script is designed to make the mounting of / proc and / sys and load the drivers related to the Raspberry Pi board. This script also
loads the modules related to video capture and video camera drivers.

5. TESTS AND RESULTS

The system was built and compared with the Linux distribution for the Raspberry Pi (Raspbian), created and maintained by the Raspberry Pi Foundation. Both systems use the same basic set of libraries based on glibc 4.6.3. A version of OpenCV used was 2.3.1, compiled the source code with binary hardware support for floating-point (hardfp) operations. Table 1 shows the characteristics of the two systems:

Table 1: Architectures of Raspbian and Our System.

<table>
<thead>
<tr>
<th>RASPBIAN</th>
<th>OUR SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel: Linux-3.6.11</td>
<td>Linux-3.6.11-custom</td>
</tr>
<tr>
<td>Kernel Size: 2.7 Mb (gzip image)</td>
<td>1.6 Mb (gzip image)</td>
</tr>
<tr>
<td>ABIs: hardfp</td>
<td>hardfp</td>
</tr>
<tr>
<td>LIBCs: GLIBC 4.6.3</td>
<td>GLIBC 4.6.3</td>
</tr>
<tr>
<td>OpenCV: 2.3.1</td>
<td>2.3.1</td>
</tr>
</tbody>
</table>

In order to compare the performances of these two systems, several tests involving boot time, memory usage and running time of OpenCV algorithms were done. For each test, 5 data outlets were taken to calculate an arithmetic average for comparison.

Boot Time

The first test was a measure of the boot time. The test was performed including the line cat / proc / uptime as the last command of the startup scripts for each system. The command displays the time that the system is active, based on the start time of the kernel execution, thereby measuring the time taken until the system becomes available to the user.

Table 2 shows the mean time (5 outlets) to boot the original system (Raspbian) and our system:

Table 2: Boot time comparison.

<table>
<thead>
<tr>
<th>RASPBIAN</th>
<th>OUR SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.65 s</td>
<td>2.38 s</td>
</tr>
</tbody>
</table>

Data show that the use of a reduced kernel image, the elimination of unnecessary services to OpenCV and simplified startup scripts could reduce by more than 90% time needed by operating system startup.

Memory Usage

As a second test, it was measured the amount of main memory available to the user. As the Raspberry Pi board uses shared video memory, in both systems were allocated to 32MB video memory, leaving 224MB of main memory, a total of 256MB (221 378 kB user available). Table 3 compares the mean values of free memory for the two implementations:

Table 3: Free memory comparison.

<table>
<thead>
<tr>
<th>RASPBIAN</th>
<th>OUR SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>171.303</td>
<td>213.975</td>
</tr>
</tbody>
</table>

Data were obtained by running the cat/proc/meminfo command soon after the boot process from a cold boot. The command displays information on memory usage via the virtual kernel files in / proc filesystem. Table 3 shows that the specialized kernel and the elimination of unnecessary services could increase the user free memory in 24.9%.

OpenCV Algorithms Running Time

As a third test measures the execution time of two programs based on OpenCV were performed: one of face recognition based on Cascade Classifier functions and other for object recognition based on SURF (Speeded Up Robust Features) algorithm.

For the first program, it does the face recognition initially training with sample images provided as an example for OpenCV. In the end, the program generates an output image based on the input image with a circle made about each region where a face was detected.

It was used a high resolution image (1680x1050 pixels), depicted below in reduced size, which contains many face objects:
Table 4 shows the execution times of the two systems. The average shows a small gain in performance in favor of our system (4.2% of reduction).

\textbf{Table 4:} OpenCV Face Recognition Running Time.

<table>
<thead>
<tr>
<th>RASPBAN</th>
<th>OUR SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.65s</td>
<td>40.86s</td>
</tr>
</tbody>
</table>

It may be noted however, that the system presented a more homogeneous response time, especially from the second run. It is possible this fact is due to the reduction in the number of services and other processes that may have concurrent execution with the test application.

The second performance test program receives two input images: a sample image of the object, and an image of the scene where the object will be tracked. The images are provided as an example in the OpenCV source code and Figure 5 show the sample image (box on the left) and the scene image (boxes on the right). Lines indicate the corresponding points in the tracking.

\textbf{Figure 5:} Image tracking in OpenCV.

After opening the input images, the program performs a pre-processing, converting to grayscale and performing a histogram equalization. As a result, regions of interest in the sample image are identified, which are eventually screened in the scene and the key points identified are linked by a set of lines. Table 5 summarizes the execution times of the second algorithm in both systems:

\textbf{Table 5:} OpenCV Face Recognition Running Time.

<table>
<thead>
<tr>
<th>RASPBAN</th>
<th>OUR SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.56 s</td>
<td>9.32 s</td>
</tr>
</tbody>
</table>

The average shows a subtle gain in performance (2.1% of reduction), but again greater homogeneity can be observed in the time of application running in our system.

6. A ROBOT PROTOTYPE

As a way of statement of work, the system was applied to built a robot using the Raspberry Pi board processing for computer vision and motion control, shown in Figure 6:

\textbf{Figure 6:} The robot prototype.

The prototype works by running the detection algorithm on the input of the webcam. The robot remains spinning slowly on its own axis, capturing images in real time and analyzing them. When a face is detected, the position of this scene is evaluated, allowing the robot to align with the viewer. After alignment, the robot moves forward to the identified person.
6. CONCLUSIONS AND FURTHER WORK

This paper summarized the essential components to an embedded Linux system, supporting Image Processing and Computer Vision in OpenCV applications and porting them to the Raspberry Pi, a computational system based on ARM architecture built on a plate with dimensions of a credit card. In the Linux kernel, convenient components for both the hardware architecture as used for Computer Vision applications have been identified. Besides the kernel configuration were identified and gathered the necessary libraries, resulting in a compact system supporting the OpenCV library.

The compilation of OpenCV 2.3.1 on ARM architecture and implementation of complex algorithms in Computer Vision Raspberry Pi board showed that it is possible to have OpenCV in embedded systems, even with a limited processing capacity. Building a custom application for this type of system allowed a drastic reduction in boot time and memory usage, making better use of resources in hardware. In embedded systems, it means cost savings and lower power consumption. Tests have shown greater consistency in execution times, making it more predictable and responsive system.

Image Processing and Computer Vision algorithms are computationally expensive, require powerful hardware and big enough memory when working with high resolution images. However, the Raspberry Pi architecture has a very poor performance by limiting the number of practices of this work the simplest algorithms applications, especially when the application involves video processing with multiple frames per second.

Although embedded systems typically use hardware with reduced processing power when compared to general purpose computers, the ARM architecture, due to its popularity, has undergone considerable advances in performance. New chips have faster cores and possess relatively powerful integrated graphics processors.

The work may be as a basis for developing a more powerful architecture, using hardware that provide parallelism via SIMD instructions and making use of the graphics processor to assist the computation of the OpenCV algorithms.

The proposed system can also be useful for the development of other embedded applications. By joining the Linux system libraries such as OpenGL and OpenAL for example, one could have a device capable of synthesizing images and sounds, producing interactive multimedia applications. With a library of physics for games such as Havok, it would be possible to have a portable device capable of running video games.

References


