International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

ARM Based Customizing an Operating System for the Single Board System (Cubie-Truck)

S. Karthik[1], T. S. Murunya[2]

[1]PG Scholar, Prist University – Kumbakonam, India

[2]Assistant Professor, CSE, Prist University – Kumbakonam, India

Abstract: ***In this paper the author going to present , The design and implementation of a CubieBoard Operating System (CBOS) on ARM (Advanced RISC Machine) platform in technical details, including boot loader design - UBOOT, building the Kernel - uImage, design of root file system and init process. The Single Board Computer Operating System (SBC OS) is developed on Linux platform with GNU tool chain. The system is mainly designed for the purpose of technical research and curriculum based teaching and students to learn, study and more readable, of which the source codes can be provided to students, guiding them to design tiny operating system on ARM platform from scratch.***

Keywords: Single board computer, UBOOT, ARM, UImage, Cubieboard, Monolithic Kernel, Init Process

1. Introduction

In our current electronic market there is many single board system computer are available, but in the other side, developing Operating System for that single board system is playing the major role in the electronic market. The author of this paper is going to design the Operating System to the single board computer system (Cubieboard). In the following subsection, we are going to discuss the introduction about the single board computer (CUBIETRUCK), Monolithic kernel structure, advantages, scope, and purpose of the project.

A. Introduction about Cubietruck and Monolithic kernel structure

Cubieboard is a single-board computer, made in china. The Cubieboard team managed to run an Apache Hadoop Computer cluster using the Lubuntu GNU/Linux distribution. It's a new PCB model adopted with Allwinner A20 main chip, just like Cubieboard2. But it is enhanced with some features, such as 2GB memory, VGA display interface on-board, 1000M nic, WIFI+BT on-board, support Li-battery and RTC,SPDIF audio interface [4]. The Cubietruck representation is given in the Figure-1 which is shown below [5].

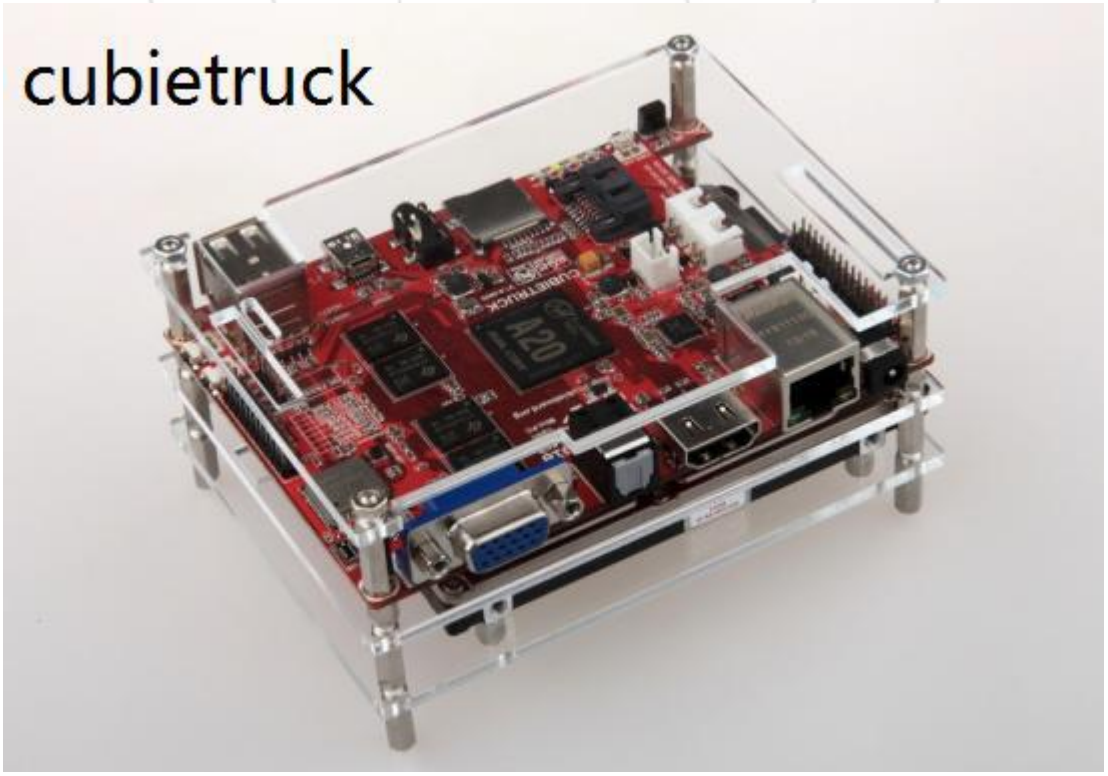

Figure 1: Cubietruck graphical representation

The components of monolithic operating system are organized haphazardly and any module can call any other module without any reservation. Similar to the other operating systems, applications in monolithic OS are

Licensed Under Creative Commons Attribution CC BY

separated from the operating system itself. That is, the operating system code runs in a privileged processor mode (referred to as kernel mode), with access to system data and to the hard-ware; applications run in a non-privileged processor mode (called the user mode), with a limited set of interfaces available and with limited access to system data. The monolithic operating system structure with separate user and kernel processor mode is shown in Figure – 2

When a user-mode program calls a system service, the processor traps the call and then switches the calling thread to kernel mode. Completion of system service switches the thread back to the user mode, by the operating system and allows the caller to continue. The monolithic structure does not enforce data hiding in the operating system. It delivers better application performance, but extending such a system can be difficult work because modifying a procedure can introduce bugs in seemingly unrelated parts of the system.

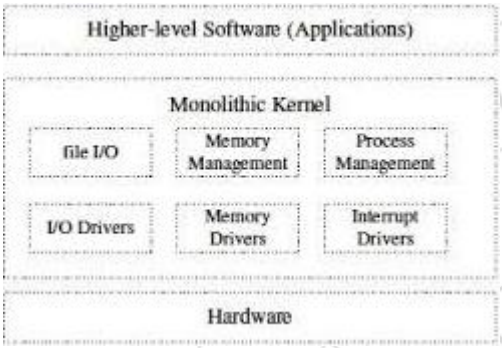

Figure 2: Monolithic kernel operating sysem structure representation

B. Advantage of CBOS

In this paper, author using the Monolithic Kernel structure for developing the cubie board Operating System (CBOS). So the essential OS services are passes from user space to the Kernel space and also increases the modularity and Structure.

2. Structure of Cubie Board Operating System (CBOS)

At the top of our SBC OS contains the user or application space where the user applications are executed. Below the user space is the Kernel space where the SBC OS Kernel exists. Figure 3 represents the architecture of the SBC OS.

Our SBC OS also contains a GNU C Library (glibc) which provides the system call interface that connects to the SBC OS Kernel and provides the mechanism to transition between the user or application space and the SBC OS Kernel.

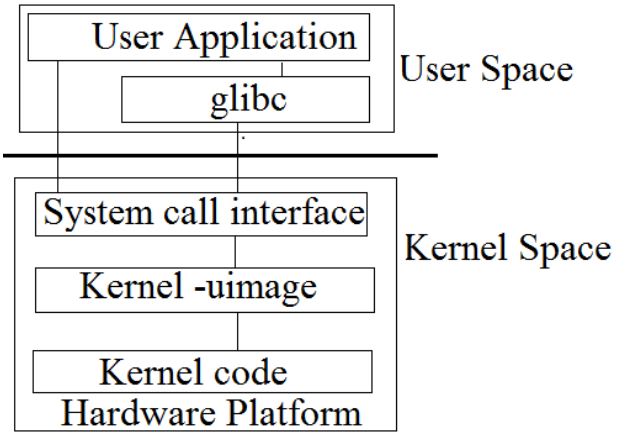

Figure 3: Structure of Cubie board Operating System (CBOS)

This is important because the Kernel and user application occupy different protected address spaces while each user or application space process occupies its own virtual address space, SBC OS Kernel occupies a single address space.

The SBC OS Kernel can be further divided into three gross levels. At the top is the system call interface, which implements the basic functions such as read and write. Below the system call interface is the SBC OS Kernel code, which can be more accurately defined as the architecture-independent Kernel code.

This code is common to all of the processor architectures supported by SBC OS. Below this is the architecture-dependent code, which forms what is more commonly, called a BSP (Board Support Package). This code serves as the processor and platform-specific code for the given architecture.

3. Modules Description

A. Design of Boot Loader (UBOOT):

The boot loader loads a kernel image from user space to the kernel space, at the time of the programmer power on the system. In the personal computer (BIOS) Basic Input Output System performs the system initialization tasks after the boot loader execution. But in the single board systems the boot loader performs the major role, because the single board system doesn't have the BIOS. So the boot loader has the three important functions which are given below.

1) Providing boot parameter to the Operating System.
2) Hardware initialization.
3) Starts up the Operating System.

The command to compilation and building the boot loader (UBOOT) for the single board system cubie board is given below,

$make clean && make cubietruck
CROSS_Compile_arm_linux_gnueabihf

B. Building the Kernel

The Kernel is the important part of the Operating System. In the Kernel space, the entire Device Driver for the Hardware of the system, Process management package, File management package, Memory management package, Network management package are available. The graphical

Licensed Under Creative Commons Attribution CC BY

representation of the CBOS Kernel is shown below in the Figure 4.

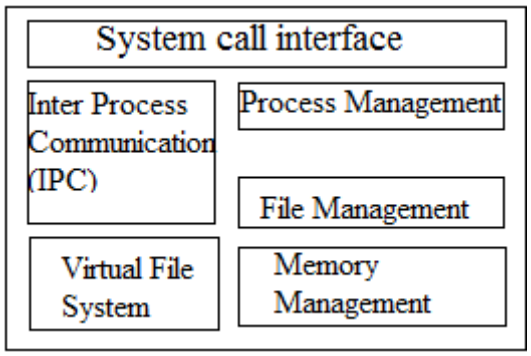

Figure 4: Structure of Kernel

The Command to compilation and building the kernel image for the sing le board system Cubie board is given below,

For Configuration:
$make ARCH=arm CROSS_COMPILE=arm-linuxgnueabi-menuconfig

For Compilation:
1. Kernel image: $make ARCH=arm CROSS_COMPILE=arm-linux-gnueabiuimage
2. Modules: $make ARCH=arm CROSS_COMPILE=arm-linux-gnueabimodules

For Installation:
1. Modules: $make ARCH=arm CROSS_COMPILE=arm-linux-gnueabimodules_ install
2. Kernel image: $make ARCH=arm CROSS_COMPILE=arm-linux-guueabiinstall

C. Root File System

Root file system contains the system files and very critical files.That is, the kernel code, Operating System supportable files are stored in the root file system. Once the root file systems get corrupted the Operating System becomes unbootable. To store the various Operating System supportable files, hardware configurable files, software routines (Kernel) in the created directory (partitions), need to mount the file system. So need to download the basic file systems from the web.

Using the below command, download the basic file systems and mount the needed file system. After the mounting the files can store in the partitions.

$debootstrap –no –check –gig –arch=armhf –foreignwheezy

D. Init Process

The init process is a program just like any other on the Linux system. The main purpose of the init process is to start and stop other programs in a particular sequence [14]. The processes or services are running on the system based on the time or state. This time or state is known as the *Runlevel.* This runlevel are represented using the numbers from 0 to 6.

4. Conclusion

The author of this paper is mainly designed this Operating System (CBOS) for the purpose of technical research and curriculum based teaching and students to learn, study and more readable, of which the source codes can be provided to students, guiding them to design tiny operating system.

References

[1] Reconos: “An Operating System Approach For Reconfigurable Computing”, Published by the IEEE computer society 0272-732/14/$31.00_c 2014 IEEE.
[2] Bo Qu and Zhaozhi Wu, “Design of Mini Multi-Process Micro-Kernel Embedded OS on ARM”, Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation, 2013, pp. 0295.
[3] Operating Systems & Boot loaders for ARM Single Board Computers, “Solutions for ARM,” http://www.embeddedarm.com/software/solutions-arm.php, 2012.
[4] Samuel Ram rajkar, Mehul Shah, Nishant Parekh, “SINGLE BOARD COMPUTER FOR APPLICATION MULTITASKING,” International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 6, December 2012.
[5] http://cubieboard.org/model/
[6] http://docs.cubieboard.org/products/start#a20-cubietruck
[7] http://www.igorpecovnik.com/2013/12/24/cubietruck-de bianwheezy- sd-card-image
[8] http://www.github.com/cubieboard
[9] Brain Ward, How Linux Works What Every super-user should know, 2004.
[10] William stallings, Operating Systems: Internals and Design Principles, 6E, Prentice Hall, inc., 2009.
[11] A.S. Tanenbaum and A.S Wookhull, Operating Systems: Design and Implementation, 3e, Prentice Hall, Inc., 2008.

Licensed Under Creative Commons Attribution CC BY