Embedded Operating Systems Internals

Final Project Report

Puneet Paresh Nipunage
Computer Engineering (CS) Department
Arizona State University
Tempe, United States
puneet.nipunage@asu.edu

Abstract—**The following report gives a brief description of all the projects performed as a part of this course. The principle aim of the project assignments was to develop a Character Device Driver in Linux and get acquainted with various related topics like proc file system, process synchronization, cache memory allocation, thread creation etc. The Device driver was targeted for the ARM architecture on the BeagleBoard.**

Index Terms—**Character Device Drivers, bootloaders, memory management, process synchronization, kernel debugging.**

I. INTRODUCTION

As a part of Embedded Operating Systems Internals course the concepts of kernel initialization, kernel debugging, device driver basics, bootloaders, memory management, process synchronization etc. were studied. The main focus of all the project assignments of the course was to get a hands-on knowledge of all the concepts studied during the coursework. At the same time each project assignment was specifically designed to get knowledge of a sub-portion in detail keeping the bigger picture of developing a character device driver in mind. Hence the entire task was divided into smaller project assignments. The entire project was divided into three major parts: I. Getting the environment prepared for developing the device driver. II. Developing the actual character device driver. III. Extending the functionality of the driver to implement some additional feature.

II. EXPLANATION OF THE SOLUTION

In order to complete the first part of the project the work of building the environment for the driver was divided into five smaller project assignments each dealing with some specific aspect of the environment. To start with, the fundamental step was to install an Operating System (OS) for developing the required driver. Linux Ubuntu version 13.04 was selected as the OS. Two options available to install the OS were: 1. Making the machine dual bootable. 2. Installing Ubuntu using Virtual Machine. For the reasons of simplicity and easy crash recovery, the second alternative was chosen.

Further a bootable MicroSD card had to be prepared as a part of the next assignment. This was necessary in order to have a portable storage for our Linux OS. In order to achieve this the MicroSD card first had to be formatted and partitioned into two areas boot and root. Boot partition is required for booting the BeagleBoard and root partition is necessary for all other user files. A sub part of this assignment dealt with creating a Linux Angstrom distribution for the BeagleBoard and then to copy the created files to the MicroSD card. Some of the files had to be moved to the boot partition which are necessary for booting Angstrom from the BeagleBoard and the others are copied to the root partition.

Next part of the first assignment involved the use of serial capture mechanism. Multiple alternatives for serial communication with the BeagleBoard were explored in this part which were: Minicom, RealTerm, secure shell (ssh) and VNC. To connect with the board the available interfaces were RS232 cable, DVI interface and the LAN cable. All these alternatives were tried out and evaluated. Next part of the assignment was the most crucial part in building the environment for the driver which involved installing software development kit (SDK) toolchain for the BeagleBoard. The toolchain actually includes various libraries and programs which are necessary for compiling the kernel for the BeagleBoard. The SDK files were downloaded from the Narcissus website which was the source even for the Angstrom distribution. SDK initialization and compilation of a test program were the initial steps which were handled. In this assignment it was also expected to understand the use of the gdb debugger. Use of gdbserver was also essential in this case since the debugging session was remote. The package installation and management program of BeagleBoard i.e. opkg was used for this purpose. Importance of compiling a program along with the debugging information embedded in the object file was understood while working on this assignment. Then with the help of the gdbserver a remote debugging session was setup with the BeagleBoard and the test program was accessed though the GUI of the gdb i.e. the ddd or the data display debugger. Process of setting a breakpoint in the program to be debugged and tracing it was exercised and understood. The last part in setting up of the environment was the project assignment part E. The goal of this part was to install the kernel build on the BeagleBoard. The main build engine required to accomplish this task was OpenEmbedded. In order to use it, some additional programs had to be installed as a pre-requisite. These were some programs like git, subversion, gawk, texinfo, python-dev scripts etc. The git program clone was used to download all the necessary setup scripts required to prepare the environment setup required for building the kernel. The kernel build also required a good memory footprint of about 12 GB. The kernel to be built had to be built with the cross complier and other necessary facilities. Bitbake was the build engine which was used to actually build the kernel for BeagleBoard as the target machine. The program menuconfig was used for configuring some options for the kernel which included building it with the debugging capabilities enabled. Once done the kernel could be installed with bitbake's install command. The importance of the kernel top level source directory and the kernel file structure was understood. After this the newly updated library files and also the newly build kernel image file were transferred to the MicroSD card. This marked the completion of the first part of the three major parts i.e. the setting up of the environment for the device driver.

The second part of the project assignments included developing the actual character device driver for ARM architecture. The driver had to be developed as a loadable kernel module. The part A of the second assignment focused on getting to work a very basic functional kernel module with the only required bare minimum functionality. In order to compile the kernel module and to generate the kernel object (.ko) file, again the build engine Bitbake was used. Before compiling the driver source code file there were a few changes which had to be made to some of the files inside the kernel top level source directory. These changes were like creating a new directory to store the source files to be complied, changing the kernel configuration file (kconfig) and also modifying the Makefile so that every time a new source file is added to the newly created directory it will be automatically compiled by just running one Bitbake command and modifying its corresponding Makefile. The earlier installed program menuconfig was instrumental in setting up the environment for Bitbake and for adjusting the required configuration parameters. These configuration changes also included setting the option for making the kernel module as a loadable module. The usage of various commands like for creating a node file (mknod), inserting a module into kernel (insmod) and removing or unloading the module (rmmod) was exercised. The functionality of the driver was tested by creating and compiling a user application program.

In the second part of project assignment two the use of proc file system was investigated and implemented in the driver created in the earlier part of the assignment. Function and importance of the proc file system in exporting critical information was studied and applied in this part of the assignment. In doing this even the kernel debugger kgdb was used. This assignment basically exposed the procedure of debugging a kernel module with the help of the kernel debugger. As a part of this assignment a read function was implemented to read from the proc file inside the driver module which basically displayed the value of a global variable.

Further as the next part of the assignment different storage allocation methods were compared and based on their pros and cons one method : kmem_cache_alloc was used to allocate storage space for the driver. The kmem_cache_alloc was selected over the other alternatives kmalloc and vmalloc because it saves a lot of time and hence makes memory allocation much more efficient. A simple linked list based character read and write functionality was implemented in the driver using read and write subroutines. An user application program was also written to exercise the functionality of the driver.

The last part of the driver implementation dealt with incorporating synchronization into the driver. Different synchronization methodologies were first studied and compared like the spinlocks and semaphores. Considering the requirement of this project, semaphores were a better alternative as compared to spinlocks. The critical sections of code: i.e. the read and write functions were protected against simultaneous access with the use of semaphore functions up() and down(). The thread creation was handled inside a user application program which referenced the implemented driver. This assignment marked the end of the part two of the projects i.e. implementing the driver.

In the part three of the project assignment it was required to add some additional functionality to the existing driver. The feature which was chosen as an add on was: glowing of BeagleBoard LED on the occurrence of a special event: reading of linked list by user when the list is empty. In addition to this even interrupt handling was included inside the code. The LED on the board also lit when the User Button on the board was pressed. Understanding of interrupt service mechanism and some knowledge of BeagleBoard's general purpose IO pins (gpio) was required to successfully implement this part.

III. RESULT AND LEARNING

As an outcome of the three project assignments a character device driver for ARM architecture was successfully implemented. The important learning from the project was getting a hands-on knowledge of majority of concepts studied in the coursework. Especially the concepts of memory allocation using slab allocator, thread creation and synchronization with the help of semaphores, use of debugging for kernel modules, importance and working of the proc file system were the once which were applied practically to get the project working. In addition the significance and working of different toolchains related to the BeagleBoard was understood. So as a whole, the learning experience out of the project was really immense.

References

1. Understanding the Linux Kernel, 3rd Edition by Daniel P. Bovet, Marco Cesati

2. Embedded Linux Primer: A Practical, Real-World Approach by Christopher Hallinan

3. Linux Device Drivers, Third Edition by Jonathan Corbet, Rubini, and Greg Kroah-Hartman