Real Time Kernel Based Hot Spot Communication Using Raspberry Pi

By K.Tamilsevan, Dr. A. Satheesh “Scientist” & Dr.S.Natarajan
Nandha Engineering College University, India

Abstract- The Real time application of an embedded Linux is essential in the area of device driver platform. Device driver plays a vital role of both hardware and software. Configuration of raspberry Pi Processor in various commands sets in Embedded Linux by enabling of Wi-Fi Device by scratch Process of various units in hardware. More number of devices can be accessed without any problem enabling N number of connections. The development of a kernel is finally changed into an image. That Backup structure will enabled by the Core-image-minimal process.

GJCST-E Classification : D.4.7
Real Time Kernel Based Hot Spot Communication Using Raspberry Pi

K. Tamilsevan, Dr. A. Satheesh “Scientist” & Dr. S. Natarajan

Abstract: The real-time application of an embedded Linux is essential in the area of device driver platform. Device driver plays a vital role of both hardware and software. Configuration of raspberry Pi Processor in various commands sets in Embedded Linux by enabling of Wi-Fi Device by scratch. Process of various units in hardware. More number of devices can be accessed without any problem enabling N number of connections. The development of a kernel is finally changed into an image. That backup structure will enabled by the Core-image-minimal process. Implementations of the bit bake execution to form an image configuration. Finally a pure kernel with a device driver bridge module is done. Here efficient to create a new hotspot communication by Raspberry Pi board.

I. Introduction

The kernel development for Raspberry Pi was essential to execute reduced time consuming methodologies. The description is systematic developments of kernel development and various control strategy proposed techniques are given below. The need for highly reliable time efficient system real-time operating systems are useful for measurement and control applications, and how they differ from standard general-purpose operating systems like Windows.

II. Problem Identification

GUIs take up a much larger amount of hard disk space than other interfaces. They need significant more memory RAM to run than other interface types. They can slow for experienced programmers to use. These people often find CLI interfaces faster than to use. More time is required for allocate individual application. Not able to execute multitasking sections. Flexibility is more.

III. Existing System

Existing system microcontroller will be configured RTOS code. There will not have a sufficient memory for a large code. Microcontroller not able to support for multitasking and scheduling process.

IV. Proposed System

- To make and configure they image data and beagle bone setup in terminal window unless the hardware being control

a) Algorithm for Empty kernel

In Linux operating system will able to execute the instructions in the terminal window. Here various parameter and command sets will run in the terminal window. Creating a directory setup updating the essential packages. Then install Yocto project simulator tool is prospective manner from the company website.

Step 1  - go to terminal and connect to internet
Step 2  - sudo apt-get update
Step 3  - sudo apt-get install build-essential
Step 4  - git clone -b dylan git://git.yoctoproject.org/poky.git
Step 5  - cd poky (getting into the folder of yocto)
Step 6  - source oe-init-build-env build-tamil-arm-simulation (creating a build directory in the name of yours)
Step 7  - bitbake -k core-image-minimal (compiling ---- it will take more time to download and compile)
Step 8  - runqemuqemuarm (running the simulation)

V. Block Diagram

These patches usually do only one thing to the source code they are built on top of each other, modifying the source code by changing, adding, or removing lines of code. Each patch should, when applied, yield a kernel which still builds and works properly. This discipline forces kernel developers to break their changes down into small, of the traditional embedded bootloaders (uBoot, RedBoot, etc.), delivering high flexibility and total system control in a 100% Linux-based small-footprint embedded solution. Version. On embedded systems, devices are often not connected through a bus allowing enumeration, hot plugging, and providing unique identifiers for devices.
VI. **BOOT LOADER**

Boot loader is a piece of code that runs before any operating system is running.

Table 1.1: Comparisons of Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Existing System</th>
<th>Proposed System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot loader size</td>
<td>40 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>Kernel size</td>
<td>2MB</td>
<td>1.5MB</td>
</tr>
<tr>
<td>Boot time</td>
<td>30 Sec</td>
<td>25 Sec</td>
</tr>
<tr>
<td>Threading</td>
<td>Single Thread</td>
<td>Multi thread</td>
</tr>
<tr>
<td>No of Devices</td>
<td>Limited to 5</td>
<td>N number of Device</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Connectivity</td>
<td>Connectivity</td>
</tr>
</tbody>
</table>

VIII. **CONCLUSION**

Embedded Linux is an essential platform for advanced real world interfaces. Here kernel development will be executed in the idea of image formations. Various command sets are used to develop a kernel in the research idea of bit bake executions. Here poky setup will identify directory setup respective progress. Here setup of a core images are configured in poky configuration of a tool. YOCTO project are used to make a simulate and analyse the hardware bridge module as a device driver section. Finally creation of an empty kernel in a reduced boot time execution. Finally hot spot communication are achieved.

**REFERENCES**

3. Dumitru TODOROI “Creativity’s Kernel Development for Conscience Society” Informatica Economică vol. 16, no. 1/2012
4. Diiva Sharma “Porting the Linux Kernel to Arm System-On-Chip And Implementation of RFID
Based Security System Using ARM” Volume 3, Issue 5, May 2013
This page is intentionally left blank