Real Time Kernel Based Hot Spot Communication using Raspberry PI
K.Tamilselvan¹ Dr.ASatheesh² Dr.S.Natarajan³
¹P.G Scholar, ²³Professor & Dean
¹²³Department of Electronics & Communication Engineering
¹²Nandha Engineering College, Erode ³United Institute of Technology

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. For making these terms in bridge connectivity for the purpose of reliable data transfer Multi-tasking is most important to run the task in perfect scheduling process with a minimized time. Kernel development is necessary criteria to implement perfect scheduling process. Perfect scheduler will raise total gain of a system Boot loader in an initial loading Period of hardware. For any critical application in minimum time period, Total size of a kernel will be reduced. Parallel operation of multithreading can run sequentially in a hardware module. The aim about this project is to implement Real Time strategy of kernel development. In order to the reduced kernel size, Boot loader time, Boot loader Size with execution of multi-threading are the important execution terms. 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 bride module is done.

Key words: Algorithm for Empty kernel, YOCTO Project

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. Real-time operating systems were designed for two general classes of applications: event response and closed-loop control. Event response applications, such as automated visual inspection of assembly line parts, require a response to a stimulus in a certain amount of time. In this visual inspection system, for example, each part must be photographed and analyzed before the assembly line moves.

II. PROBLEM IDENTIFICATION
In the Modern world development real world application should have an excellent features is necessary. Regarding that multitasking, more efficiency, Time requirement, memory requirement and effective kernel development is necessary to overcome a drawback.
- 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.

A. Drawbacks:
- Boot loader size is previous size of 35 kb above this size It having unwanted tools like i2c tools, Memtest, etc..
- Kernel size was 2 MB it is a default allocation in beagle bone black.
- Boot time is more than 30 seconds. Here unwanted drivers run in the time of execution.
- Real time kernel patch have not the option of multi-threading.
- More drivers are required to execute.
- Limited Number of users can access the Wi-Fi

IV. PROPOSED SYSTEM
The main objective of the system,
- To implement a pure kernel system in an Empty manner for creates an efficient platform for device driver.
- To make and configure they image data and beagle bone setup in terminal window.
- Unless the hardware being controlled is really misbehaving.
- User memory is swappable, unlike kernel memory. An infrequently used device with a huge driver won't occupy RAM that other programs could be using, except when it is actually in use.
- A well-designed driver program can still allow concurrent access to a device.

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. Formation of an image in the minimal execution method. Device driver bridge module
will be created by run qemux module. More executed parameter run on the executed terminal window.

1) STEP 1 - go to terminal and connect to internet
2) STEP 2 - sudo apt-get update
3) STEP 3 - sudo apt-get install build-essential
4) STEP 4 - git clone -b dylan git://git.yoctoproject.org/poky.git
5) STEP 5 - cd poky (getting into the folder of yocto)
6) STEP 6 - source oe-init-build-env build-tamil-arm-simulation (creating a build directory in the name of yours)
7) STEP 7 - cd build-tamil-arm-simulation (getting into that directory)
8) STEP 8 - cd conf (getting into conf directory)
9) STEP 9 - geditlocal.conf (editing the file)
10) STEP 10 - change the line following
11) STEP 11 – MACHINE ??= "qemu86" to MACHINE ??= "qemuarm"
12) STEP 12 - save and quit gedit
13) STEP 13 - cd.. (coming out of the conf directory)
14) STEP 14 - bitbake -k core-image-minimal (compiling --- it will take more time to download and compile)
15) STEP 15- 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, Figure 4.1 shows the Logical pieces; as a result, each change can be reviewed for code quality and correctness in a single Boot loader solution, all the facilities of the full-featured Linux OS and the simplicity and speed of the traditional embedded boot loaders (uBoot, RedBoot, etc..), delivering high flexibility and total system control in a 100% Linux-based small-footprint embedded solution.

VI. BOOT LOADER

Boot loader is a piece of code that runs before any operating system is running. Boot loader are used to boot other operating systems, usually each operating system has a set of boot loaders specific for it. Boot loaders usually contain several ways to boot the OS kernel and also contain commands for debugging and/or modifying the kernel environment. In this talk we will concentrate on Linux boot loaders. Since it is usually the first software to run after power up or reset, it is highly processor and board specific, the Linux fast boot solution and the Linux “boot load” program provided.

A. YOCTO Project:

The YOCTO to Project will take ease-of-design to the next level. The alignment with Open Embedded will drastically increase open source collaboration efforts and make it easier than ever to promote Linux in embedded systems as well as deliver a positive and improved end developer experience.

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. It was founded in 2010 as collaboration among many hardware manufacturers, open-source operating systems vendors, and electronics companies to bring some order to the chaos of embedded Linux development. As an open source project, the Yocto Project operates with a hierarchical governance structure based on meritocracy and managed by its chief architect, Richard Purdie, a
Embedded Linux and Real time kernel are essential for a real-time embedded system. Here real-time embedded codes are fetched in the terminal window. Configurations each and every set will play a bigger role in kernel development. Finally with the use of minimum driver Empty kernel will produce. That set will simulate in Yocto project window. Qemu panel will create Device driver bridge module. Window panel is full of pure kernel size of that kernel is reduced.

VII. RASPBERRY PI TERMINAL WINDOW
Build version of a terminal window belongs to 1.18.0 here allocation of a setup will initiate. It manages various configuration of embedded Linux in device. Build system will formats to x86_64-linux. Nativebsstringin the format of Ubuntu 14.04.Target_sys is X86_X64-linux. Interior machine version is qemuarm. Distropacky will depend on they poky formation 1.4.4.

VIII. COMPARISON

<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 Connectivity</td>
<td>Limited to 15 Devices</td>
<td>N number of Device Connectivity</td>
</tr>
</tbody>
</table>

Table 1.1: Comparison of Parameters

IX. 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. Development of kernel by various instruction of command sets. Here bit bake will helps to identify the beagle configurations in a conversion module. 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.

X. FUTURE ENHANCEMENT
The kernel development can be further enhanced to real-time application and implementation of software will extend to hardware. Then number of Wi-Fi device will automatic make in beagle bone configuration in extensive communication also can be introduced in the future implementation.

REFERENCES
[9] JaydevsinhJadeja, ChintanKapadiya, “Porting the Linux Kernel to Beagle Bone Black”IJSRD -
