IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.
PP 13-18
www.iosrjournals.org

Custom Linux Kernel for Raspberry Pi Using Ubuntu 12.04 Host

Mahendra Swain[1], Abhishek Kumar Srivastava[2]
[1,2]*(Department of ECE, Lovely Professional University, Punjab, India)*

Abstract: *In this paper, it is discussed that the Linux kernel customization for ARM11 platform based Raspberry Pi. In this customization of kernel using different tool chains is done. It will show how to configure and cross compile Linux kernel for the Raspberry Pi on Ubuntu 12.04. An overall idea to building Linux kernel for Raspberry Pi using Yocto Project. The custom Linux kernel synchronizes the time, scheduling, resources allocation and managing of all the hardware peripherals and require less memory.*
Keywords *- Scheduling, Customization, Tool chain, Bitbake , Metadata, Poky, Dora*

I. INTRODUCTION

Now a day's Linux operating system is the choice for almost all new embedded device projects .Linux provides a powerful, flexible kernel and open runtime platform which is being improved by the Linux community to support new processors, buses, devices, and protocols. Cost-effective and time-critical embedded hardware projects can take advantage of its freedom i: e free availability. Embedded device projects can often reduce hardware costs by taking advantage of the power and flexibility that a true multi-tasking operating system brings to embedded devices. The Linux kernel and associated open source infrastructure is the core of a new ecosystem for embedded operating system, infrastructure, and application prototyping, optimization, and deployment. In this article we have discussed the configuration Linux kernel for advanced ARM processors.so it is necessary to update the old Linux kernel when that becomes not appropriate for interrupt handling, Scheduling different tasks, resources allocating, management of on chip memory, multitasking and Easy user interfaces.

Porting of Linux kernel on a target platform depends upon number of factors. It concerns with the Linux kernel configuration and compilation for the raspberry Pi on Host Ubuntu 12.04. Tool chains are build up around cross compiler and executable file can port in target platform. The Linux kernel supports different types of architectures, such as X86, ARM. So the protocols for are different for each architectures. In this article we have taken Embedded Linux Device, Raspberry Pi (Model B) based on ARM1176JFZ-S processor with BCM2835 system on chip for research purpose.

II. RASPBERRY PI

Raspberry Pi is a small, multifunctional credit size computer operated in Linux Operating System.

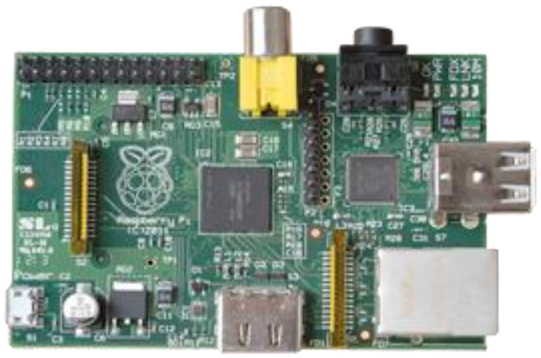


Fig.1. Raspberry Pi model B

Raspberry Pi is $35 which is affordable the advantage of this computer is portable, Pi is clocked in at 700MHZ Powerful desktop. The most advantage thing regarding ARM processor is it consumes less power. The Model B Pi has 512MB of RAM. It supports Ethernet cable to access the network, USB cable to interface with peripheral devices. RCA Video (works with older TVs), 3.5mm Audio Standard headphone socket, HDMI Audio & Video (works with modern TVs and DVI monitors).so after discussing the all the advantages of Raspberry Pi we can clarify our doubts.

III. LINUX KERNEL CUSTOMIZATION

A. LINUX KERNEL ARCHITECTURE

There are three different layers in Linux kernel. At the top level SCI (system call interface), the significance of this layer is to read and write instruction and socket calls. Then there are architecture dependent and architecture independent layers. Process management execute the process and shares the CPU and the active threads. [2] The virtual file system provides common interface abstraction for file system in kernel. The kernel also concerns with management for memory for keeps track of which pages are partially filled and empty. [1]

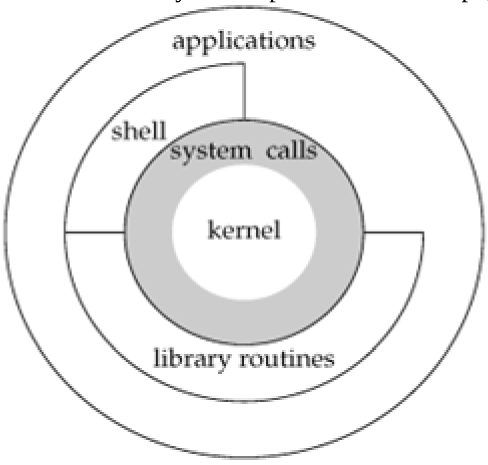


Fig.2. Linux kernel architecture

The device drivers have the source codes for Linux kernel. The arch subdirectory is the architecture-dependent and contains subdirectories for various architecture of machine. [3]

B. Cross compilation

Cross compiler provides the platform to generate and execute codes for a target in which compiler is running. Cross compilation environments support Application Binary Interface (ABI) and Embedded Application Binary interface (EABI). [3]The ABI represents higher level language to machine level language. For different targets Linux kernel get updated with tool chains for different application.

C. Compilation of Linux Kernel for ARM

We have to download the latest stable kernel release from www.kernel.org and extract it in ~/linux-stable to speed up the process, use the current kernel named .config which can be found in /boot with a name starting with config- followed by the kernel version and copy it to the top src directory of the kernel. [4]

```
$ cp /boot/config-* ~/linux-stable/.config
```

The new kernel may include options not found in our current kernel and thus there may be a few configuration options that we need to still specify.

```
$ cd ~/linux-stable
$ make oldconfig
$ make -j`cat /proc/cpuinfo | grep -c processor`
$ sudo make modules install
```

The above command installs the kernel image and copies the configuration for the new kernel in the /boot directory. It also modifies the boot loader configuration so that the boot loader (ex: GRUB) recognizes If we decide that we no longer need a particular kernel version, we can completely get rid of it by deleting the corresponding kernel's config, vmlinuz, System. Map and initrd from the /boot folder and the corresponding kernel modules from /lib/modules and the kernel header from /usr/src. [5]

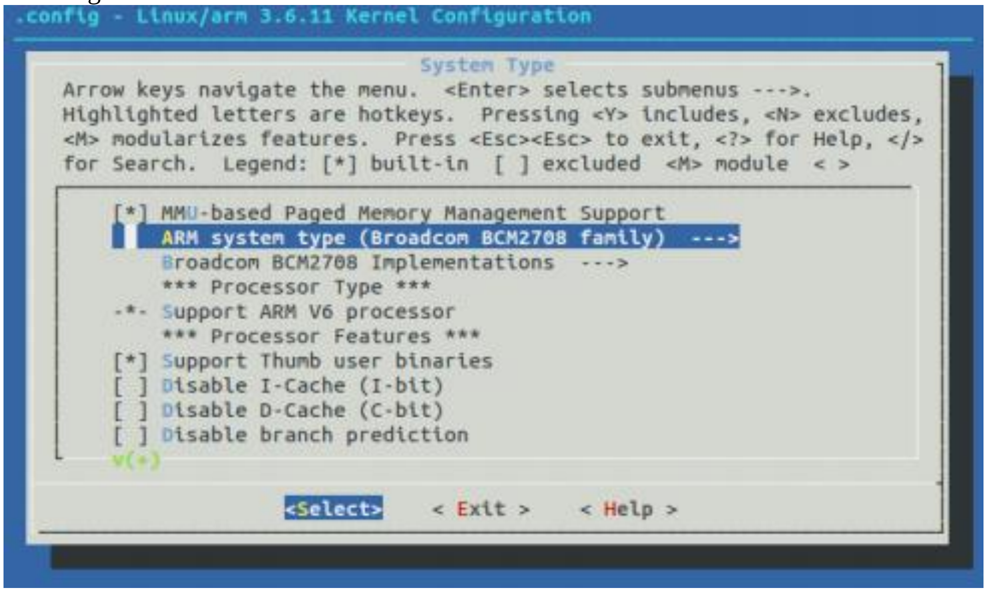
Once we are done deleting these files, all that remains is to update the boot loader by running
"$ sudo update-grub2".

If we decided to rebuild the new kernel, run "$ make mrproper" in ~/linux-stable to clean the kernel configuration and all the files that have already been built and we are ready to start all over again. [3]

Compilation of custom linux kernel for raspberry Pi on Ubuntu 12.04 host

Create our own root directory and download linux and tools for raspberry Pi

https://github.com/raspberryPi/tools.git
https://github.com/raspberryPi/linux.git
$cd linux
In order to avoid error .We have taken following steps
mahi@ubuntu:~/rasp_ckernel$gitclone https://github.com/raspberryPi/tools.git
$sudo apt-get install git
Install git mahi@ubuntu:~/rasp_ckernel$ sudo apt-get install git
[sudo] password for mahi:
Using Configtool-NG to customize and install toolchain for ARM processor Configtool requires number of packages for smooth installation.
$ sudo apt-get install gawk bison flex gperf cvs texinfo automake\libtoolncurses-dev g++ subversion python-dev \ libexpat1-dev cmake
Create directory which are to be used
$ mkdir -p ~/raspberry_armtools/build/toolchain \ ~/raspberry_armtools/toolchains \
Crosstool-NG isn't available in the standard Ubuntu
Repositories, so we must build it. Run the following commands to download, build, and install Crosstool-NG:
$ pushd ~/raspberry_armtools/build
http://crosstoolng.org/download/crosstoolng/crosstool-ng-1.18.0.tar.bz2
$ tar xf crosstool-ng-1.18.0.tar.bz2 && cd crosstool-ng-1.18.0
We have run the following commands to launch menuconfig, then follow the sub-sections below to configure the toolchain build parameters. [5]
$ pushd ~/raspberry_armtools/build/toolchain
$ ct-ng menuconfig



```
.config - Linux/arm 3.6.11 Kernel Configuration
                         System Type
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      [*] MMU-based Paged Memory Management Support
          ARM system type (Broadcom BCM2708 family)  --->
          Broadcom BCM2708 Implementations  --->
          *** Processor Type ***
      -*- Support ARM V6 processor
          *** Processor Features ***
      [*] Support Thumb user binaries
      [ ] Disable I-Cache (I-bit)
      [ ] Disable D-Cache (C-bit)
      [ ] Disable branch prediction
      v(+)

             <Select>    < Exit >    < Help >
```

Fig.3.Menuconfig. with Raspberry Pi setting

Navigating the Paths and misc options.
$ ct-ng build
$ popd
$export PATH=~/raspberry_armtools/toolchains/arm-unknown-linux-gnueabihf/bin:$PATH
$ popd
For permanently include PATH for toolchain (previous method include path temporarily since PATH is limited to current terminal instance), modify ~/.bashrc file and at the bottom of file write. [4]
export PATH=~/raspberry_armtools/toolchains/arm-unknown-linux-gnueabihf/bin:$PATH
Generating the .config file from the pre-packaged raspberry Pi one:
$makeARCH=arm CROSS_COMPILE=arm-unknownlinuxgnueabihfbcmrPi_cutdown_defconfig
To customize Linux Kernel as per our requirement, we are using following command

```
$makeARCH=arm      CROSS_COMPILE=arm-unknown-linux-gnueabihf- menuconfig
```

Once we have made the desired changes, save and exit the menuconfig screen. Now we have started the build. We can speed up the compilation process by enabling parallel make with the -j flag. The recommended use is 'processor cores + 1′, e.g. 5 if we have a quad core processor: [3]

```
$makeARCH=armCROSS_COMPILE=/usr/bin/arm-linux-gnueabi- -k -j5
```

Assuming the compilation was successful, create a directory for the modules. Modules are parts of the kernel that are loaded on the fly, as they are needed. They are stored in individual files (e.g. ext3.o). The more modules

```
$mkdir ../modules
```

Then compile and 'install' the loadable modules to the temp directory:

```
$makemodules_installARCH=armCROSS_COMPILE=arm-unknown-linux-gnueabihf
INSTALL_MOD_PATH=../modules/
```

Now we need to use imagetool-uncompressed.py from the tools repo to get the kernel ready for the Pi.

```
$sudo cp -a lib/firmware/ /media/rootfs- partition-uuid/lib/
$sync
```

IV. CUSTOMIZATION USING YOCTO PROJECT

The Yocto project [15] is a Linux Foundation project that provides open source, high quality infrastructure and tools to create custom Linux distributions for any hardware architecture. It was founded in 2010 as a collaboration between many hardware manufacturers, open-source operating systems vendors, and electronics companies. Among the members of the project are Intel, Texas Instruments, LSI, Monta Vista Software, Juniper Networks, Huawei, Mentor Graphics etc [3]. Yocto uses Open Embedded [3] build framework that supports ARM, MIPS, PowerPC and x86 architectures for a variety of platforms including x86-64 and emulated ones. Yocto Project offers tools and components to design, develop, build, debug, simulate, and test the complete software stack using Linux, the X Window System, GNOME Mobile-based application frameworks, and Qt frameworks. [7] An image developed with the Yocto Project can boot inside a QEMU emulator supporting testing and development. The Yocto development environment consists of 1) Poky that is the platform-independent, cross-compiling integration layer that utilizes Open Embedded Core, 2) Bitbake that is build and metadata manager, tells the system how to build packages, and 3) metalayer which contains information on how to build packages.We start create the Raspberry Pi environment by installing Poky and the metalayer [3]:

```
$ mkdir yocto
$ cd yocto
$ git clone -b dora git:
git://git.yoctoproject.org/poky.git
$ cd poky
$git clone -b dylan
git://git.yoctoproject.org/meta-raspberryPi
```

Note: Dylan version of Poky has problems with sqlite. Therefore we use dora version of Poky.
To initialize environment variables and the build directory.

```
$ . oe-init-build-env build
```

Then we need to edit conf/local.conf to match our compilation environment and to set the target machine as Raspberry Pi, and possibly to adjust the GPU memory, by updating or adding the correspo -nding lines in local.conf

```
BB_NUMBER_THREADS = "2"
PARALLEL_MAKE = "-j 2"
MACHINE ?= "raspberrypi"
GPU_MEM = "16"
```

Other system parameters such as GPU memory, license codecs and overclocking can be adjusted as described in [20]. The path to meta-raspberrypi needs to be added to bblayers.conf file located in poky/build/conf, so that it would look like to this [7]

```
“BBLAYERS ?= " \
/home/mahi/yocto/poky/meta \
/home/mahi/yocto/poky/meta-yocto \
/home/mahi/yocto/poky/meta-yocto-bsp \
/home/mahi/yocto/poky/meta-raspberrypi \
```

"
Now we can create the image by invoking the command:

```
$ bitbake rpi-basic-image
```

This image will contain ssh server support. After the system is compiled and built there will be a file in tmp/deploy/images/rpibasic-image-raspberry.rpi-sdimg. This is a symlink to the binary image that can be copied into a SD card:

```
$ sudo dd.sh if=tmp/deploy/images/rpi-basicimage-raspberrypi.rpi-sdimg of=/dev/sdb bs=1M
```

The SD boots the Raspberry Pi with the newly compiled kernel and modules.
To add features or adjust memory of the kernel, we can change the kernel configuration before building the system with command. [3]

```
$ bitbake virtual/kernel –c menuconfig
```

This opens the same graphical kconfig menu that was used in the earlier compilation sections. Through the menu selections we can do similar configuration changes as were described in the previous section, “Compiling for QEMU”. The new configured kernel should be build with the “$ bitbake virtual/kernel”. [7]

A .Performance comparison with existing Kernel
We have run a c program to test the speed of kernel. We have observed that the speed has been increased in comparison to existing kernel.

```
 $ time sleep 2
real    0m2.009s
user    0m0.000s
sys     0m0.004s
```

V. CONCLUSION

For porting a new kernel to Raspberry Pi, it requires SoC drivers of the BCM2708/BCM2835. In this paper creation of linux kernel using crosstool-NG is shown .The yocto projects guide for own Linux kernel using different packages. It is very flexible method to add and remove the packages for cutom linux kernel according to our requirement. This paper will give an overall idea to create own linux kernel for Raspberry Pi and other Board Support Package Devices like Beagle bone.

VI. ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to Mr.Abhishek Kumar Srivastava ,Assistant Professor, Department of Electronics and Communication Engineering, Lovely Professional University, Punjab for his continuous guidance and support throughout this report work. Lastly, I would like to thank my parents for their years of unyielding love and encourage. They have always wanted the best for me and I admire their determination and sacrifice. Thanks to god for this life in this beautiful creature.

VII. REFERENCES

[1] Using Phase Behavior in Scientific Application to Guide Linux Operating System Customization, Chandra Krintz Rich Wolski Computer Science Department University of California, Santa Barbara, IEEE computer society, 2011
[2] Runtime CPU Scheduler Customization Framework for a flexible mobile operating system, Proceedings of 2009 Student Conference on Research and Development (SCOReD 2009), 16-18 Nov. 2009, UPM Serdang, Malaysia
[3] Building Linux Kernel for Raspberry Pi Hannu Flinck, Nokia Solutions and Networks hannu.flinck@saunalahti.fi
[4] The GNU Toolchain for ARM targets HOWTO by Wookey, Chris Rutter, Jeff Sutherland, Paul Webb
[5] Building Linux Applications with the ARM ® Compiler toolchain and GNU Libraries, 2010-2011 ARM.ARM DUI 0483C (ID080411)
[6] BCM2835ARMPeripherals,http://raspberryPi.org/wpcontent/upload/2012/02/BCM2835-ARMperipherals.pdf
[7] http://www.yoctoproject.org/docs/yocto-quick-start/yocto-project-qs.html
[8] Diagnosys: Automatic Generation of a Debugging Interface to the Linux Kernel Tegawendé University of Bordeaux, France

[9] http://help.1and1.com/hosting-c37630/linux-c85098/php-c37728/importing-and-exporting-mysql-databases- using- php-a595887.html
[10] http://net.tutsplus.com/tutorials/php/how-to-send-text -messages- with-php/
[11] https://github.com/raspberryPi/linux
[12] QEMU Emulator User Documentation, http://qemu.weilnetz.de/qemu-doc.html
[13] https://wiki.debian.org/ArmEabiPort#In_a_nutshell
[14] Raspberry Pi Kernel Compile, http://mitchtech.net/raspberryPi-kernel-comPile/
[15] Yocto Project Quick Start,https://www.yoctoproject.org/docs/current/yocto-projectqs/yocto-project-qs.html
[16] Yocto Project participants,https://www.yoctoproject.org/ecosystem/yocto-projectparticipants
[17] http://www.openembedded.org/wiki/Main_Page
[18] 12MB Minimal Image for Raspberry Pi using the Yocto Project, http://www.cnx-software.com/2013/07/05/12mbminimal- image-for-raspberry-Pi-using-the-yocto-project/
[19] https://github.com/djwillis/metaraspberryPi/blob/master/README
[20] XEC Design, QEMU – Emulating Raspberry Pi the easy way, http://xecdesign.com/comPiling-qemu/