International Journal of Electrical and Computer Engineering (IJECE)
Vol. 9, No. 6, December 2019, pp. 4920~4928
ISSN: 2088-8708, DOI: 10.11591/ijece.v9i6.pp4920-4928

A reliable approach to customizing linux kernel using custom build tool-chain for ARM architecture and application to agriculture

Mahendra Swain[1], Rajesh Singh[2], Anita Gehlot[3], Md Farukh Hashmi[4], Shiv Kumar[5], Manish Parmar[6]
[1,2,3,5]Lovely Professional University, India
[4]National Institue of Technology, India
[6]Nanhi Pari Seemant Engineering Institute, India

Article Info

Article history:

Received Feb 27, 2019
Revised Jun 25, 2019
Accepted Jul 5, 2019

Keywords:

Cloud server
Customization
Internet of things
Mobile APP
Scheduling
Tool chain

ABSTRACT

ARM processors are receiving more attention as per IoT customized devices are concerned. A novel framework design tool for Linux kernel customization on ARM architecture has been illustrated. The tool is best suit from ARM based platformss like Raspberry pi, Beagle Bone, Intel Edison etc. The proposed techniques uses different tool chains for the kernel customization. The paper represents an integral framework that integrates all the cross compiling tools and simplifies the overall process. The framework has been used for the development of a customized kernel for Raspberry Pi on Ubuntu 14.04 host computer. The custom kernel has been ported in to Raspberry Pi and the performance evaluation has been done. Furthermore, the analysis aims to help users choose and configure their tracers based on their specific requirements to reduce their overhead and get the most of out of them. The testing of customized OS with raspberry Pi device in the field of agriculture. The smart node/mote is designed based on it to deploy in the agriculture field to test its feasibility. The group of nodes data is gathered using ThingSpeak cloud server. The gathered sensory data is analyzed and forecast on farmer's mobile phone in the form of APP or handheld device for farmer.

Copyright © 2019 Institute of Advanced Engineering and Science.
All rights reserved.

Corresponding Author:

Mahendra Swain,
Lovely Professional University,
Jalandhar - Delhi G.T. Road, Phagwara, Punjab 144411, India.
Email: Er.mahendraswain@gmail.com

1. INTRODUCTION

Since the feature of technologies are being enhanced and the performances are also getting improved accordingly that the hardware and software are modified [1]. In this article we discuss the configuration Linux kernel for advanced ARM processors [2]. So it is necessary to update the old Linux kernel when that become not appropriate for interrupt handling, Scheduling different tasks, resources allocating, management of on chip memory, multitasking and Easy user interfaces [3, 4].

Porting of Linux kernel on a target platform depends upon number of factors. We concerns with the Linux kernel configuration and compilation for the raspberry pi on Host Ubuntu 14.04. Tool chains are build up around cross compiler and executable file can ported in target platform [5]. 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 create embedded Linux system in to raspberry pi Computer based on ARM1176JFZ-S processor with BCM2835 system on chip [6].

2. 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 [7]. Then there are architecture dependent and architecture independent layers. Process management execute the process and shares the CPU and the active threads. 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, filled and empty [8, 9]. Figure 1 shown the Linux kemel architecture. The device drivers have the source codes for Linux kernel. The arch subdirectory is the architecture-dependent and contain subdirectories for various architecture of machine [8].

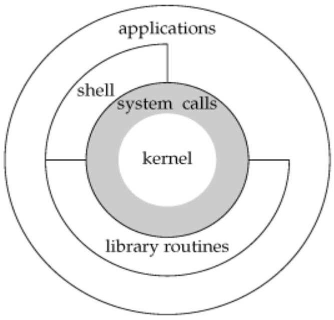

Figure 1. Linux kernel architecture

3. NODE ARCHITECTURE

The generalized block diagram in Figure 2 shows the various nodes are placed in the agriculture field. The nodes namely node1, node2 … and node N are homogeneous in nature. The nodes are having their own architecture and capable to communicate via Xbee network to coordinator node. The coordinator node having Wi-Fi to communicate over internet [10]. The cloud server namely ThingSpeak record the data of different fields of the particular channel. The server direct the decision to mobile app of the farmer using Blynk APP.

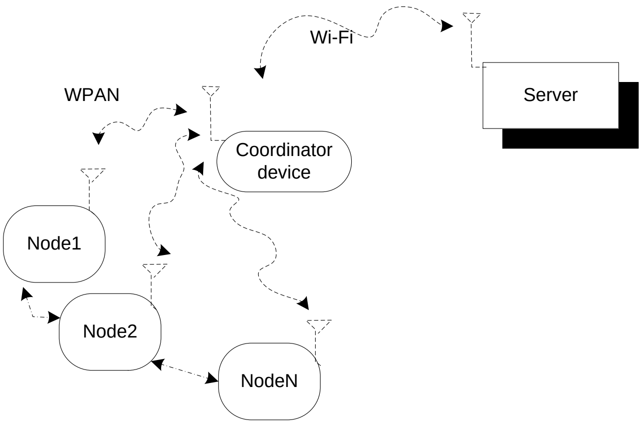

Figure 2. Generalized architecture

In remote agricultural field if internet facility is not available then the xbee network formed via the architecture as shown in Figure 3. The Node contains controller unit i.e Arduino, the group sensors i.e DHT11, BMP185, ultrasonic sensor as water level sensor, gas sensor measures hazardous gases, soil moisture sensor, flame sensor, display unit, xbee modem and power supply adaptor (+12V), power supply convertor [5].

The xbee based network formed a wireless personal area network [WPAN] required a coordinator whose architecture as shown in Figure 4 to collect data locally and having customized OS loaded raspberry pi3 capable to send the data to cloud. The coordinator are having customized OS loaded raspberry pi3, display unit, xbee modem and power supply convertor [3, 8, 9]. The agricultural field where internet facility is available then direct raspberry pi-based node directly upload data on cloud using the architecture as shown in Figure 5. The Node OS loaded raspberry pi3, the group sensors i.e DHT11, BMP185, ultrasonic sensor as water level sensor, gas sensor measures hazardous gases, soil moisture sensor, flame sensor, display unit and power converter [11].

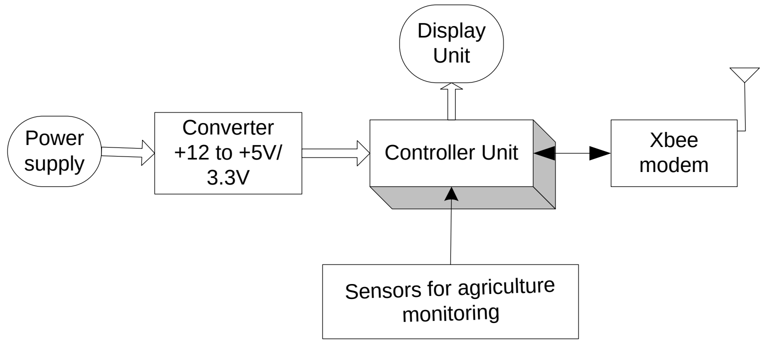

Figure 3. Node architecture if internet is not available in agriculture field

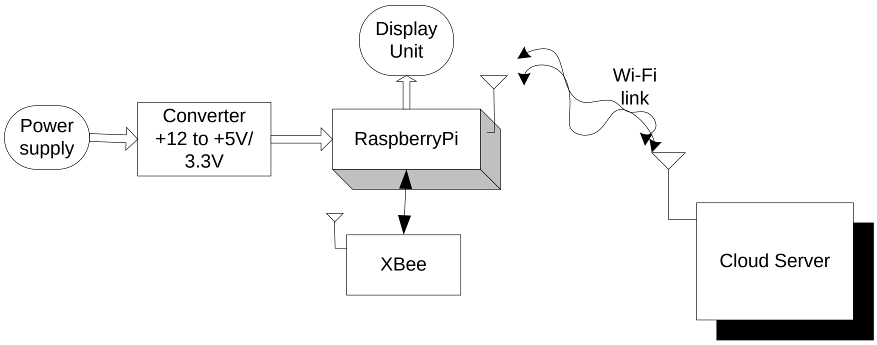

Figure 4. Intermediate device/coordinator architecture

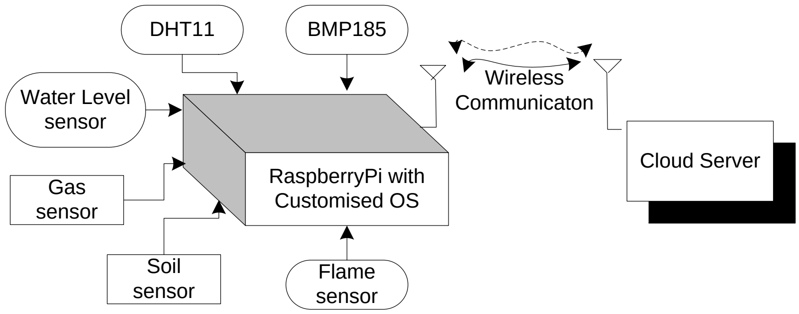

Figure 5. Node architecture if internet is available

4. CIRCUIT AND SCHEMATICS DIAGRAM

The system is design and tested as per given hardware circuitry as shown in Figure 6, Figure 7 and Figure 8. The Figure 7 shows the node circuit of remote agricultural filed using xbee modem. The various sensors and their mode of communication with Arduino is fairly discussed in schematics. The connection among xbee, Arduino and LCD20*4 also discussed.

The Figure 7 shows the node circuit of coordinator and its shows the interconnection among xbee, LCD 20*4, and raspberry pi. The various sensors and their mode of communication with Arduino is fairly discussed in schematics[12-13]. The Figure 8 shows the node circuit of if the internet connection is available in agricultural filed. The various sensors and their mode of communication with raspberry pi3 is fairly discussed in schematics. The connection among xbee, LCD20*4 and power supply also discussed [7].

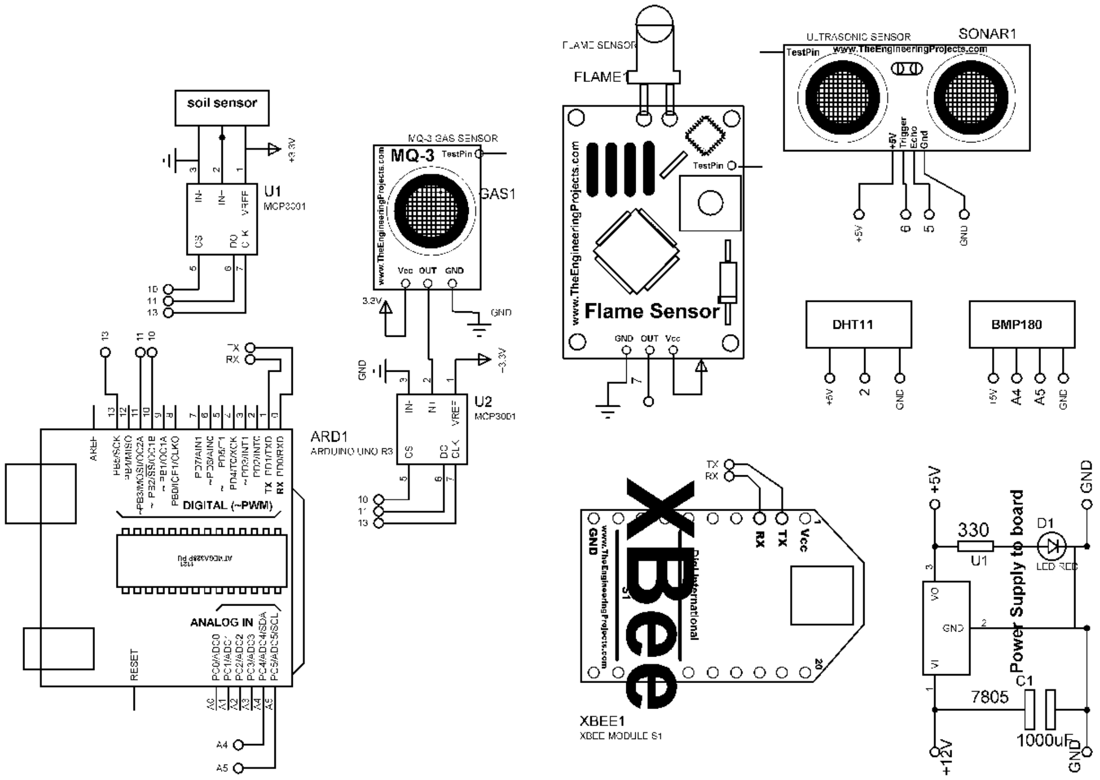

Figure 6. Node Circuit if internet is not available in agriculture field

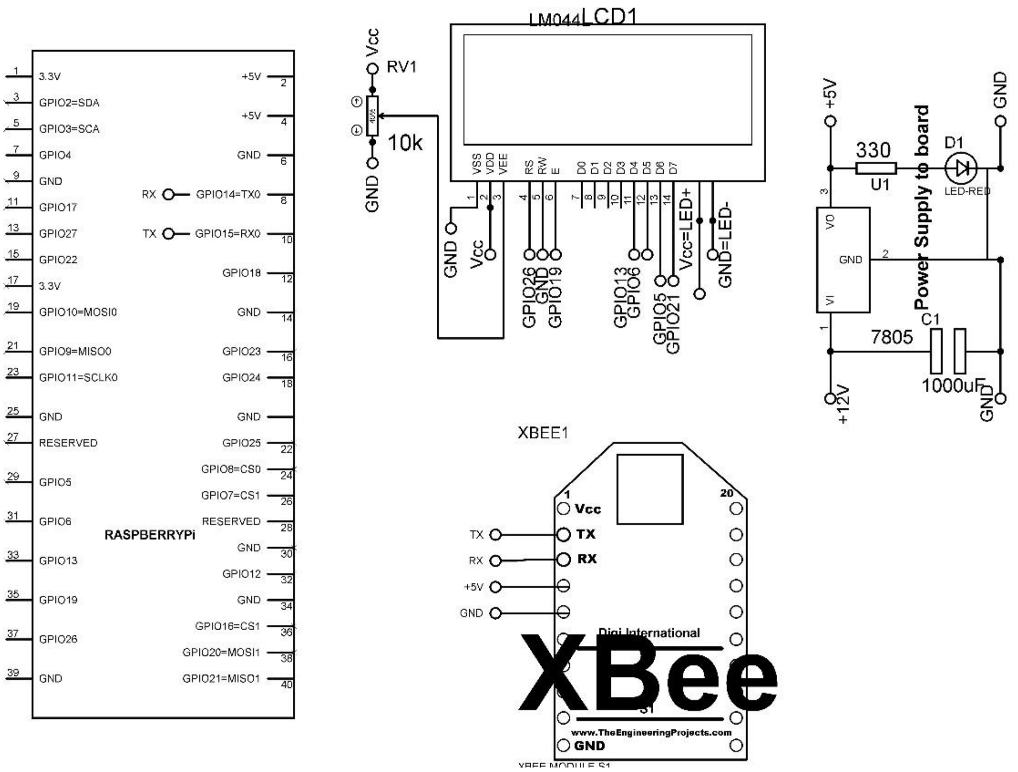

Figure 7. Intermediate device/coordinator architecture

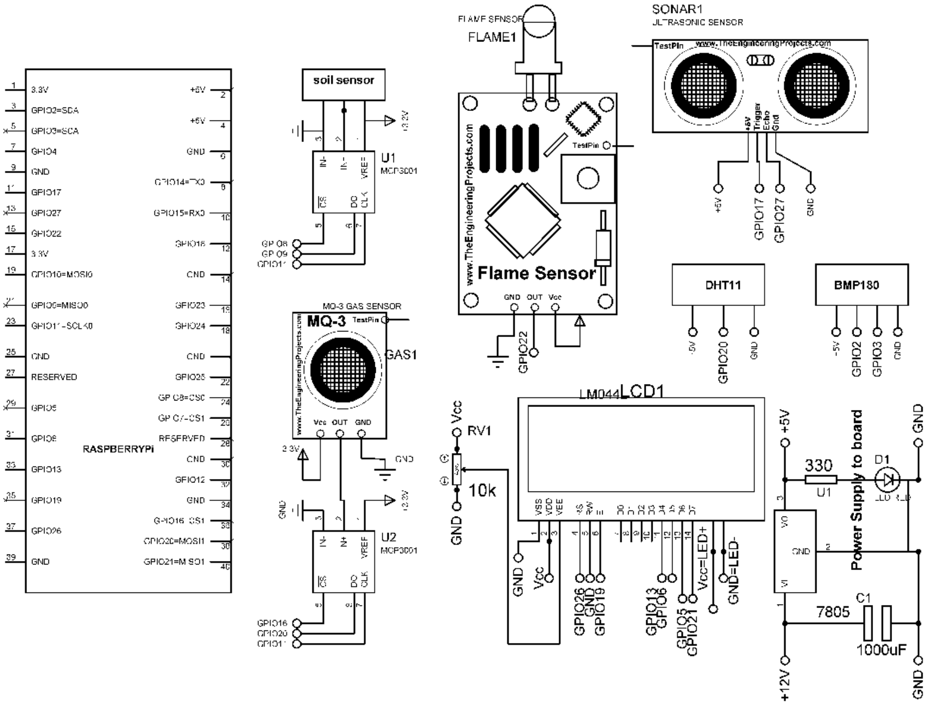

Figure 8. Node circuit if internet is available

5. CROSS COMPILATION AND SOFTWARE DEVELOPMENT

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) [8]. The ABI represents higher level language to machine level language. For different targets Linux kernel get updated with tool chains for different application. Figure 9 shown flow diagram for cusyomization of Linux Kemel [5, 14, 15].

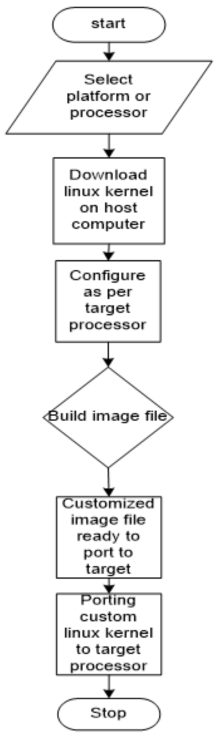

Figure 9. Flow diagram for customization of linux kernel

5.1. Compilation of linux kernel on ARM

5.1.1. Method 1

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 [16].

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

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

$ cd ~/linux-stable
$ make oldconfig

If we are not sure what to answer to those questions, you can select the default by simply pressing the Enter key for each of the questions. Once the kernel configuration is complete you are ready to actually start compiling the linux kernel [15].

$ make -j`cat /proc/cpuinfo | grep -c processor`

It will take 4-5 Hours' time would be required to build kernel.The above command builds the kernel image as well as the kernel modules that get loaded dynamically. Now, all that is left is to install the new kernel image and kernel modules and to get the bootloader (ex: GRUB) to recognize and boot the new kernel the next time you boot your computer [14].

$ 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 bootloader configuration so that the boot loader (ex: GRUB) recognizes the new kernel. The kernel modules are installed into /lib/modules with the kernel version as the name and are linked to the kernel image. The kernel headers are installed into /usr/src [17].

Reboot the system Verify the new kernal version
$ Uname -r.

If you decide that you no longer need a particular kernel version, you 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 [18]. Once we are done deleting these files, all that remains is to update the bootloader by running
"$ sudo update-grub2"**.**

If you decide 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 you are ready to start all over again [6]. Compilation of custom linux kernel for raspberrypi on Ubuntu 14.04 host
Create our own root directory and download linux and tools for raspberrypi
https://github.com/raspberrypi/tools.git
https://github.com/raspberrypi/linux.git
cd linux
$ 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 todownload, 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
Run the following commands to launch menuconfig, then follow the sub-sections below to configure the toolchain build parameters:
$ pushd ~/raspberry_armtools/build/toolchain
$ ct-ng menuconfig
Customization of Toolchain for ARM processor:
$ ct-ng build
$ popd

If the build was successful, the toolchain will be located at ~/raspberry_armtools/toolchains/arm-unknown-linux-gnueabihf/. All the tools (gcc, ld, gdb, etc) are located in the bin/ directory of the toolchain with the name of the toolchain prefixed [12, 19].

5.1.2. Method 2: using YOCTO project

Then you need to edit conf/local.conf to match your compilation environment and to set the target machine as Raspberry Pi, and possibly to adjust the GPU memory, by updating or adding the corresponding 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:
“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, you can change the kernel configuration before building the system with command:
$ bitbake virtual/kernel –c menuconfig.
This opens the same graphical kconfig menu that was used in the earlier compilation sections [21]. Through the menu selections you can do similar configuration changes as were described in the previous section, “Compiling for QEMU”.
The new configured kernel should be built with the “$ bitbake virtual/kernel”.

6. PERFORMANCE EVALUATION

Performance evaluation has been done on custom kernel for following details as shwon in Table 1, Table 2, and Table 3.

Table 1. Using python on raspbian

Evaluation parameters	Bubble sort	Binary Search	Merge sort
CPU cycles used	$2.2x10^{18}$	$3.2x10^{18}$	$2.5x10^{18}$
Context switch time in ms	1245	1324	1367
Task clock cycle	3456	3589	3678
Cache hit time in ms	976	976	945
Overall performance in percentage	75	69	79

Table 2. Using python on PiLFS

Evaluation parameters	Bubble sort	Binary Search	Merge sort
CPU cycles used	$2.1x10^{18}$	$2.9x10^{18}$	$2.3x10^{18}$
Context switch time in ms	1189	1201	1235
Task clock cycle	3879	3987	3794
Cache hit time in ms	1125	1192	1232
Overall performance in percentage	84	73	65

Table 3. Using python (YOCTO)

Evaluation parameters	Bubble sort	Binary Search	Merge sort
CPU cycles used	$3.2x10^{18}$	$4.5x10^{18}$	$2.7x10^{18}$
Context switch time in ms	1239	1287	1342
Task clock cycle	4232	3954	3875
Cache hit time in ms	1345	1356	1189
Overall performance in percentage	73	65	75

7. RESULTS AND CONCLUSION

Customizing Linux kernel for target processor is one effective tool. The analysis has been done on target processor i:e Raspberry Pi. Same techniques can be used for other platform as well. The Figure 10 shows that overall performance comparison among three OS namely Raspbian, PiLFS and Yocto (customized) with respect to three algorithm namely bubble sort, binary search and merge sort. The customized OS for raspberry pi-based system may be used in various domain of engineering like smart cities, agriculture, waste management, water management etc.

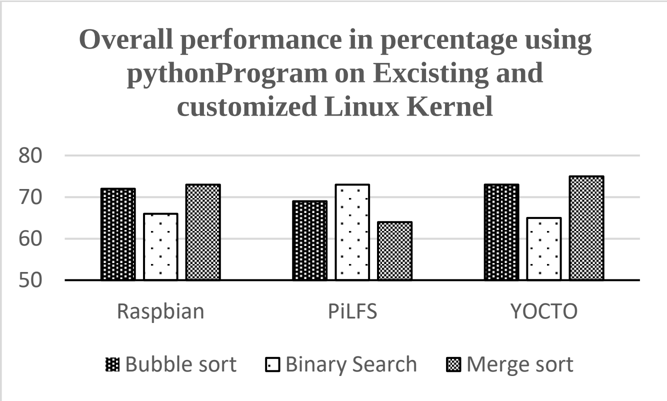

Figure 10. Performance comparison

REFERENCES

[1] A. Rigoni, *et al*., "A framework for the integration of the development process of Linux FPGA System on Chip devices," *Fusion Engineering and Design*, vol. 128, pp. 122-125, 2018.

[2] M. P. Karpowicz, *et al*., "Design and implementation of energy-aware application-specific CPU frequency governors for the heterogeneous distributed computing systems," *Future Generation Computer Systems*, vol. 78, pp. 302-315, 2018.

[3] R. R. Chodorek and A. Chodorek, "A Linux Kernel Implementation of the Traffic Flow Description Option," *Multimedia and Network Information Systems, Springer, Cham*, pp. 161-170, 2017.

[4] M. Gebai and M. R. Dagenais, "Survey and Analysis of Kernel and Userspace Tracers on Linux: Design, Implementation, and Overhead," *ACM Computing Surveys (CSUR)*, vol. 51, pp. 26, 2018.

[5] Khanna A., and Kaur, "S. Evolution of Internet of Things (IoT) and its significant impact in the field of Precision Agriculture," *Computers and electronics in agriculture*, vol. 157, pp. 218-231, 2019.

[6] Lingayat A., Badre R. R., and Gupta A. K., "Integration of linux containers in openstack: An introspection," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 12(3), pp. 1094-1105, 2018.

[7] Esquembri, S., "Embedded Linux Systems: Using Buildroot for building Embedded Linux Systems on Raspberry Pi 3," Dpto de Telemática y Electrónica Universidad Politécnica de Madrid, 2018

[8] Verma G., Imdad M., Banarwal S., Verma H., and Sharma A., "Development of Cross-Toolchain and Linux Device Driver," *In System and Architecture, Springer,* Singapore, pp. 175-185, 2018.

[9] P. Wang, *et al*., "How double-fetch situations turn into double-fetch vulnerabilities: A study of double fetches in the Linux kernel," *USENIX Security Symposium*, 2017.

[10] Sethi, P., and Sarangi, S. R., "Internet of things: architectures, protocols, and applications. Journal of Electrical and Computer Engineering, vol. 1, pp. 1-25, 2017.

[11] Suryani V., Sulistyo S., and Widyawan W., "Trust-based privacy for Internet of Things," *International Journal of Electrical and Computer Engineering*, vol. 6(5), pp. 2396, 2016.

[12] Lingayat A., Badre R. R., and Gupta, A. K., "Integration of linux containers in openstack: An introspection," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 12(3), pp. 1094-1105, 2018.

[13] Ramakrishnan R., Gaur L., Singh G., "Feasibility and Efficacy of BLE Beacon IoT Devices in Inventory Management at the Shop Floor," *International Journal of Electrical & Computer Engineering (IJECE)*, vol. 6(5), pp. 2088-8708, 2016.

[14] Y. W. Chen and H. M. Sun, "An Approach for Reducing the Traffic within Cloud Environments Based on Customized Linux Kernel," Cloud Computing Technology and Science (CloudCom), *2017 IEEE International Conference on*, 2017, pp. 227-230.

[15] P. Kamboj, *et al*., "Real-Time Implementation of Scheduling Policies for Education Using Raspberry Pi: A Review," *Proceedings of 2nd International Conference on Communication, Computing and Networking, Springer*, Singapore, 2019, pp. 127-134.

[16] Passos L., Queiroz R., Mukelabai M., Berger T., Apel S., Czarnecki K., and Padilla J., "A study of feature scattering in the linux kernel," *IEEE Transactions on Software Engineering*, 2018.

[17] C. Dall, *et al*., “Optimizing the design and implementation of the Linux ARM hypervisor,” *Proceedings of the 2017 USENIX Conference on Annual Technical Conference (USENIX ATC’17). USENIX Association*, Berkeley, CA, USA, 2017, pp. 221-233.
[18] Wang H., Chen Z., Xiao G., Zheng Z. “Network of networks in Linux operating system,” *Physica A: Statistical Mechanics and its Applications*, vol. 447, pp. 520-526, 2016.
[19] Lelli J., Scordino C., Abeni L., and Faggioli D. “Deadline scheduling in the Linux kernel,” *Software: Practice and Experience*, vol. 46(6), pp. 821-839, 2016.
[20] Díaz G., Rojas P., and Barrios C., "Methodology for Tailored Linux Distributions Development for HPC Embedded Systems,” In *Latin American High Performance Computing Conference*. *Springer*, Cham, Sep 2018, pp. 280-290.
[21] Abeni L., Balsini A., and Cucinotta T., “Container-based real-time scheduling in the linux kernel,” *In EWiLi'18, the embedded operating system workshop, Co-located with the Embedded Systems Week,* Oct 2016.