Volume 4, Issue 2, February 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Automated Installation of Multiple Linux Operating System Using Fast Reboot Technology

Asif Chand Shaikh, Niranjan Liladhar Chaudhari

Computer Engineering Department

University Of Pune, India

Abstract— Time has become an important aspect of life for every individual on this planet. Machines that take a lot of time to complete, proves to be redundant in the society and people would generally not prefer to use that machine in time critical aspect. It's a true fact that each and every traditional procedures that were followed are transforming into the automatic and efficient way, thus increasing the productivity of the organisation. In this paper we have proposed a tool "Automated Installer for Linux Operating Systems". It is a proposed technology that can be used by the industrial teams to automatically set the environments for their workstations according to the requirements of their project. Most often the working environment changes from one project to other, so in this case the engineer in the organisation is not expected to sit in front of the machine and set the complete working environment manually and then start the actual work. There are many things that are to be done to set the environment in a workstation before the work on that actually begins viz. installation of various packages, software's, installation of operating system according to the needs of the project and if there are more than one operating system to be installed then there is no other way except to sit in front of the computer and complete the task installation one after the other. So, the proposed tool "Automated Installer for Multiple Operating Systems" is a tool which will automate the installation process of Linux distribution on a workstation. The most striking feature of this tool is that it can install more than one Linux operating systems in one go without manual intervention. This paper will describe various ways and method which will be used to develop the proposed tool.

Keywords— Linux Kernel, Kexec functionality, GRUB, Kickstart, PXE server, kickstart file, autoyast files, pre-seeding.

1. INTRODUCTION

Installation of operating system is a tedious task and also very boring except if it is done for the first time, i.e. the scenario will be good if there is only one personal machine that needs to be formatted followed by the installation of the operating system. This task proves to be very redundant and mundane if the installation is to be carried on large number of machines and that too frequently. Therefore there is a good scope to bring in the automation aspect here and automate the installation of more than one operating system. Here the installation will be taking place through network; hence there will not be any need to maintain a large collection of CD's and DVD's, for various Linux operating systems [1].

In this project we have proposed to install multiple Linux operating systems on the same hard disk back-to-back automatically i.e. without any manual intervention. This proposed tool will be very helpful from the industry point of view especially in the IT industry where the frequency of installation of various operating systems is very high. Generally, in industries to carry out any work the machine on which the task is to be carried out must be set and there is a lot of time required to set the machine. This pre-installation process requires a lot of manual task and the engineer has to spend a lot of time in doing the same redundant and mundane work all the time and also the engineer is expected to sit in front of the screen throughout the installation process to answer the questions that are asked by the installer. Due to this the valuable time is wasted as the engineer is not doing any productive work. So if there is any tool or mechanism that will automate the above process will let the engineer work on the task for which he is intended and not waste his/her valuable time in non-trivial tasks like the one described above. On the other hand this installation tasks are mandatory as the engineer will not be able to carry out the required work until the machine is completely set. So here we have proposed an idea that will help to install more than one operating systems automatically and also the proposed tool will allow the engineer to customize the installation process before the installations start like the engineer can customize the size of the / (root) partitions before installation starts, i.e. the tool will not perform the installation in predefined way but will allow the engineer to customize some of the options.

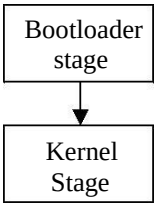
The tool will help to install the following Linux distributions back to back automatically:-

1. Red Hat Enterprise Edition.
2. Fedora
3. Open Suse
4. Ubuntu

This tool will basically involve logic to start the automatic installation of the next operating system once the installation of the previous one is finished. The details of how the partitioning of the hard disk will be done are explained in details in further sections.

2. THEORY OF BOOTING KERNEL

The boot process in Linux as mainly two broad stages [3]:-



The main components of the *bootloader stage* are the hardware stage, the firmware stage, the first-level bootloader, and the second-level bootloader. The kernel booting process begins as soon as the power is on. After some initialization, control goes to the firmware. Firmware also known as "BIOS", detects the various devices on the system, including memory controllers, storage devices, bus bridges, and other hardware. The firmware based on the settings, hands over control to a minimal bootloader known as the master boot record, which could be on a disk drive, on a removable media, or over the network. The actual job of transferring control to the operating system is performed by the second-stage bootloader (commonly referred to as simply the "bootloader"). This bootloader allows the user to choose the kernel to be loaded, loads the kernel and related parameters onto memory, initializes the kernel, sets up the necessary environment, and finally "runs" the kernel. The next stage of booting is the *kernel stage*, when the kernel takes control. It sets up the necessary data structures, probes the devices present on the system, loads the necessary device drivers, and initializes the devices. The last stage of the booting process involves user-level initialization. In this stage, the kernel checks the integrity of file systems, mounts file systems, sets up swap partitions (or swap files), starts system services, sets up system terminals, and sets up a whole lot of other things. So, after the bootloader stage the control goes into the kernel stage and then the actual working of the operating system takes place. Similarly the installation of the any operating system takes place i.e. the bootloader stage and the kernel stage. The bootloader stage will load the appropriate kernel image and the initial ram disk and then passes the control to it.
So in the proposed tool the bootloader which will be used should be capable to load the kernel image the initrds appropriately and automatically one after the other.

3. LINUX AS BOOTLOADER

The concept of using the full fledged Linux kernel as the host to run the application (bootloader) will help us in achieving our target. The basic task of a bootloader is to load the appropriate kernel image and the initrd to run the kernel image when the system is powered on or when we are installing any new operating system. The traditional bootloader i.e. GRUB (Grand Unified Bootloader) or LILO all run on the functionalities provided by the BIOS legacy. Hence the reason this bootloader will provide a very minimum set of functionality to accomplish the task of automating the installation of more than one operating system. As the traditional boot loader will not allow the user to select multiple operating systems at once. Which will force the user to come in front of the screen to select the next operating system to install and then start the installation of the next OS, which will not fulfil the concept of automating, moreover GRUB functionalities cannot be used to maintain a state machine through which we can make the state of the system persistent and the previous installation state will be lost once the system is rebooted. So here we have proposed to use the Linux kernel as the base to host the application which in our case can also be called as bootloader, as this will trigger the installation of target OSes, which will in fact prove to be a very strong bootloader. This can allow interacting with the user in graphical mode using the X server of the host operating system, that will make the application worth of using it as the user can customize the installations at once for more than one operating system before the actual installation starts. Using Linux kernel as the host to run the bootloader will not only help in loading the kernel and initrd images of the next operating system to be installed but will also help in many other features which are necessary viz.

a. Maintaining a state machine
b. Updating the application with the updates available on the server.
c. Updating the final GRUB entries, once all the installations are over.

Now the main question that arises is how to use the Linux kernel as a bootloader. To load a new kernel the kernel memory on the ram must be free. In traditional bootloader when the system is powered on, the control passes to the master boot record and GRUB will pop's up on the console, during the installation or during normal booting. At this stage the kernel memory is free and whichever kernel image is loaded by the bootloader can be efficiently go into the memory without any dispute. But when we talk about using Linux kernel as the bootloader, we have to first free the space used by the current kernel for the next kernel to be loaded, further in Linux the kernel run from a fixed address in memory. This means that the new kernel needs to sit at the same place that the current kernel is running from. On x86 systems, the kernel sits at the physical address 0x100000 (virtual address 0xc0000000, known as PAGE_OFFSET).

Fortunately in Linux there is some kernel functionality using which we can achieve the above goal i.e. by using the kexec functionality of the Linux kernel [1].

In computing, kexec (coming from kernel execution, derived from the Unix/Linux kernel call exec) is a mechanism of the Linux kernel that allows "live" booting of a new kernel "over" the currently running kernel. Essentially, kexec skips the bootloader stage (hardware initialization phase by the system firmware, BIOS or UEFI) and directly loads the new kernel into memory, where it starts executing immediately. This avoids the long times associated with a full reboot, and can help systems to meet high-availability requirements by minimizing downtime. With kexec, you can reboot directly into another kernel, without having to go through the firmware and bootloader stages. Skipping the lengthiest part of the sequence reduces the reboot time drastically. The task of overwriting the old kernel with the new one by kexec is done in three stages:

1. Copy the new kernel into memory.
2. Move this kernel image into dynamic kernel memory.
3. Copy this image into the real destination (overwriting the current kernel), and start the new kernel.

To load a kernel using kexec, the syntax is as follows:-

kexec -l <kernel-image> --append="<command-line-options>"

where <kernel-image> is the kernel file that you intend to reboot to and <command-line-options> contain the command-line parameters that need to be passed to the new kernel

Then, to actually reboot to the loaded kernel, just type:

Kexec –e

The system will reboot immediately. Unlike the normal reboot process, kexec does not perform a clean shutdown of the system before rebooting. It is left to us to kill all applications and unmount file systems before attempting a kexec reboot.

4. MAIN COMPONENTS OF THE SYSTEM

Following are the components that are involved to carry out the back to back installation.

1. HTTP/NFS server, which will contain all the repositories of all operating system
2. A light weight host operating system with X server to run the app which will assist the engineer in customization of the installation
3. Pre configuration files which will automate the installation of a particular operating system
4. A reliable network connection between the client and the server
5. Workstation on which the installation should be carried on.

The application which will be running on the host operating system can be developed in any high level language like java or python since it has X server support for graphical interface.

5. STRUCTURE OF THE SYSTEM

The basic architecture that will be followed by the system is client and server architecture. Here the server will be installation server with the pre configuration files of all the operating system and the client will be any normal OEM workstation. As said earlier a light weight Linux kernel will be used as the host that run the app(bootloader), interact with the client, take the inputs , do the required validation and customization of the Kickstart files that will be used during the installation and finally trigger the booting of the first kernel image from the queue, using the kexec. The host operating system will be installed first on the clean hard disk. Therefore there will be some space that will be reserved solely for the host operating system. The installation of the host operating system must be done in the following way:-

1. /boot partition will be installed on the first partition of the hdd i.e. for example sda1
2. Swap partition will be installed on the second partition of hdd i.e. sda2
3. And lastly the root (/) partition will be installed on the third partition i.e sda3

All the above partitions will be primary partitions. Since the MBR allows at max only four primary partitions, on any hard disk, therefore now we will be left with only one partition now that can be used for the installation of target OSes. The first three primary partitions are used by the host OS the only remaining partition i.e. sda4 will be used to install all the target operating systems.

All the target operating system will be installed automatically in the following manner:-

1. /boot partition will be mounted on sda1, which is already created during the installation of the host operating system
2. Swap partition will be automatically mounted on sda2
3. Lastly the root (/) partition of the target operating system will be installed on the next free space available from sda4., this partition will be an extended partition

In this way all the target operating system will be sharing the /boot partition and swap partition and will have their own / (root) partition as an extended partition.

The figure given below depicts the entire scheme:

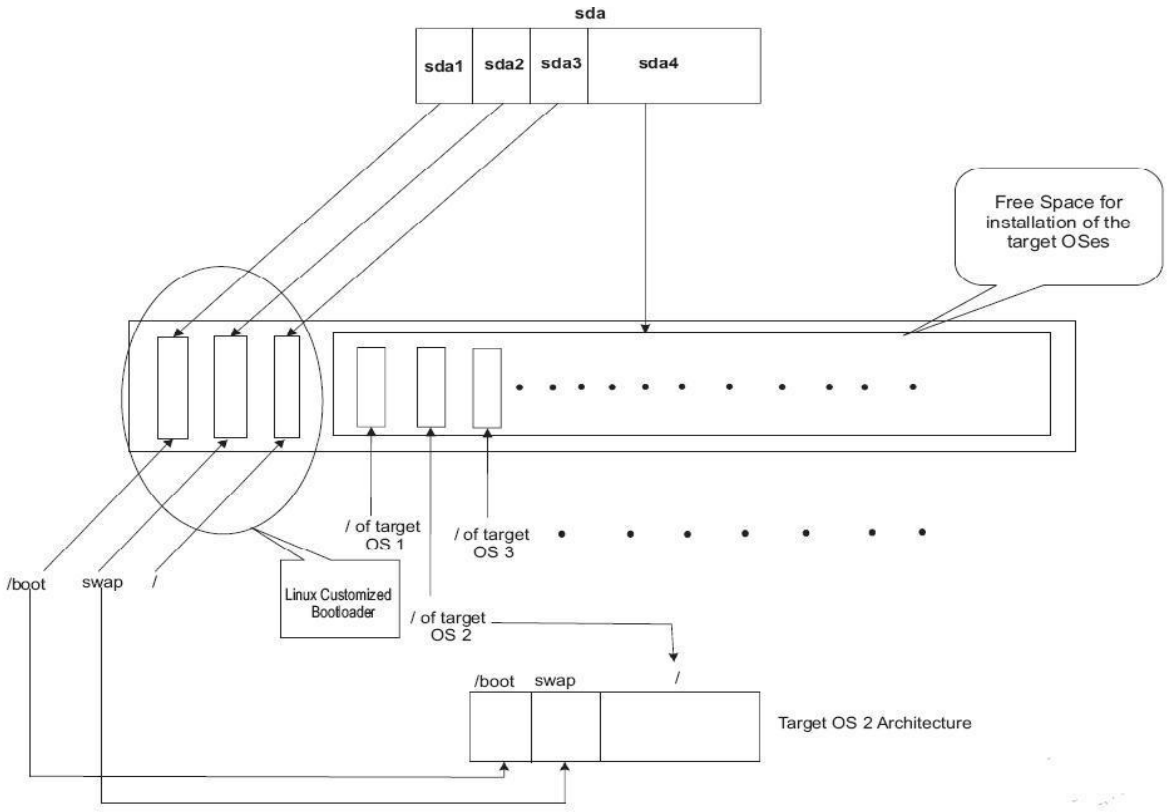


Fig. 1 Hard disk Architecture of the client machine

The installation of operating system will be automatic i.e. using the kickstart technology [6]. Kickstart file is a file that will help to automate the installation process of installer like anaconda. It generally consists of the answers to the questions that are asked by the installer. The main issue in kickstart is it should automatically install the target operating system in the above manner. It should mount the /boot partition on sda1 and use swap partition while at the same time create a new partition as an extended one for the /.

This can be achieved by using the following lines of code in the kickstart file:

```
# Partition clearing information #clearpart --
none

# Disk partitioning information

part /boot --fstype="ext3" --onpart=/dev/sda1 --noformat part / --fstype="ext3" --
size=30000

        #part swap --fstype="swap" --size=4096
```

And the bootloader of the target operating system obviously is not to be installed, as we will be using the bootloader of the host operating system.

This can be achieved by including the following lines of code in the Kickstart file:-

```
# System bootloader configuration bootloader --
location=none

# Clear the Master Boot Record #zerombr
```


6. WORKING MECHANISM OF SYSTEM

According to the above mentioned architecture, the host operating system will be installed on first three primary partitions of the hard disk and the grub of the host operating system will be used, since here we are disabling the installations of the GRUB of each target operating system.

Initially when the system boots up, the client will boot into the host operating system and the application will pop up on system start-up. This application is a graphical user interface and runs with the support of X server, hence it is very important that the host operating system must have X server. The client will choose the operating systems from the list of available operating systems on the server, here the client will be able to choose more than one operating system. Using this graphical user interface the client can also customize the installations, like he may change the size of root partitions or can change the default password, and also customize the package selection, and user accounts etc. Once all the interactions are done, the user clicks on the start install button, here all the validations are done according to the user selection, like the available size on the hard disk is compared to the space that will be needed for the next installations and some network validations will be done. Once all the validations are over, a state machine will be created which will now handle the installations.

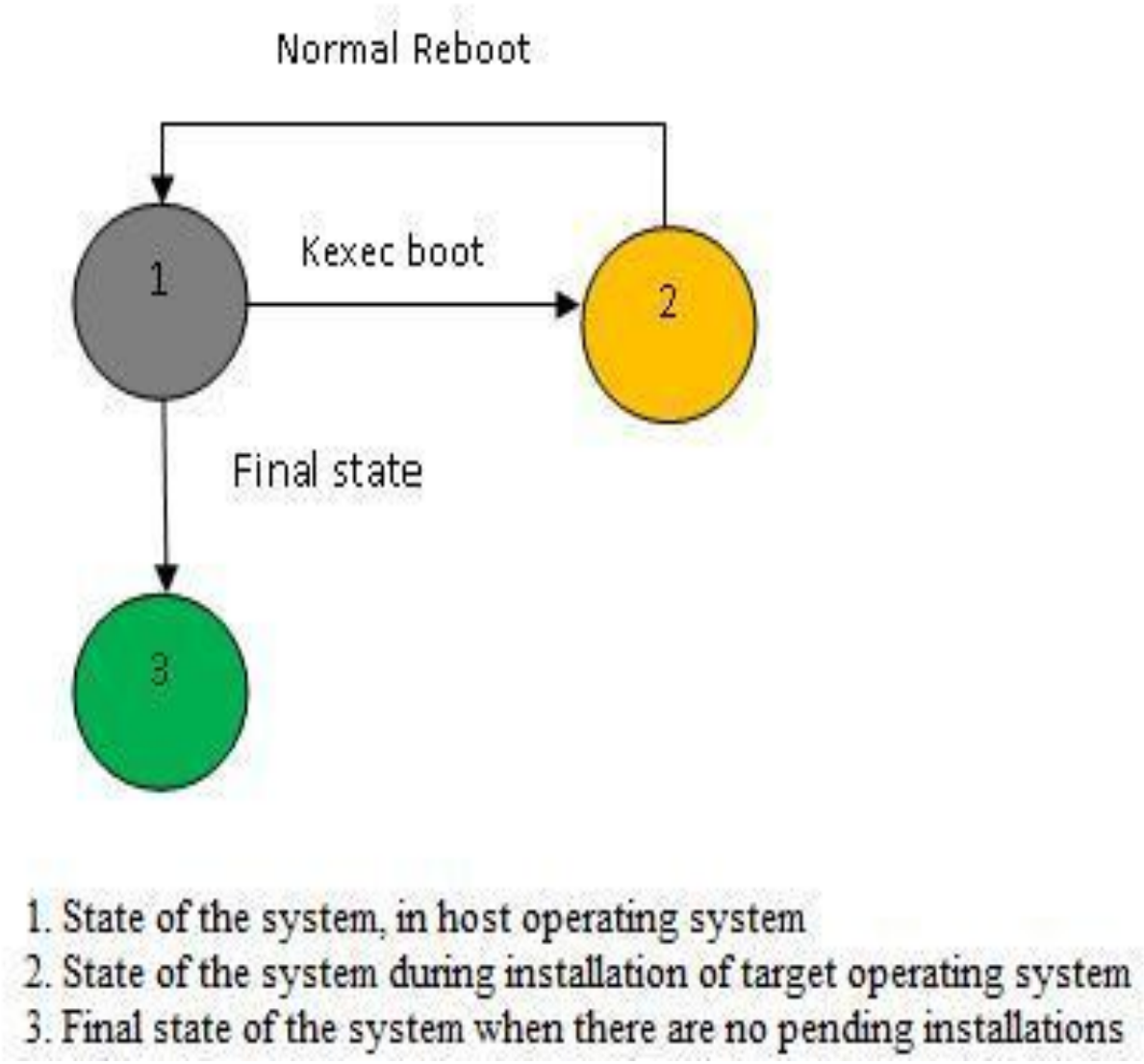


Fig. 2 State Machine

This state machine maintains the state of the system persistent, i.e. even after the system reboot the previous state of the system will be restored which will help the system to take the appropriate actions. After every installation the control passes back to the host operating system, and the state machine which is present there will decide the next actions. If there are some pending installations it will complete the installation of those first and lastly when there are no pending installations the state machines goes into the final state and the grub is updated automatically to reflect the installation of the operating systems.

Below is the graphical representation of the working of the system.

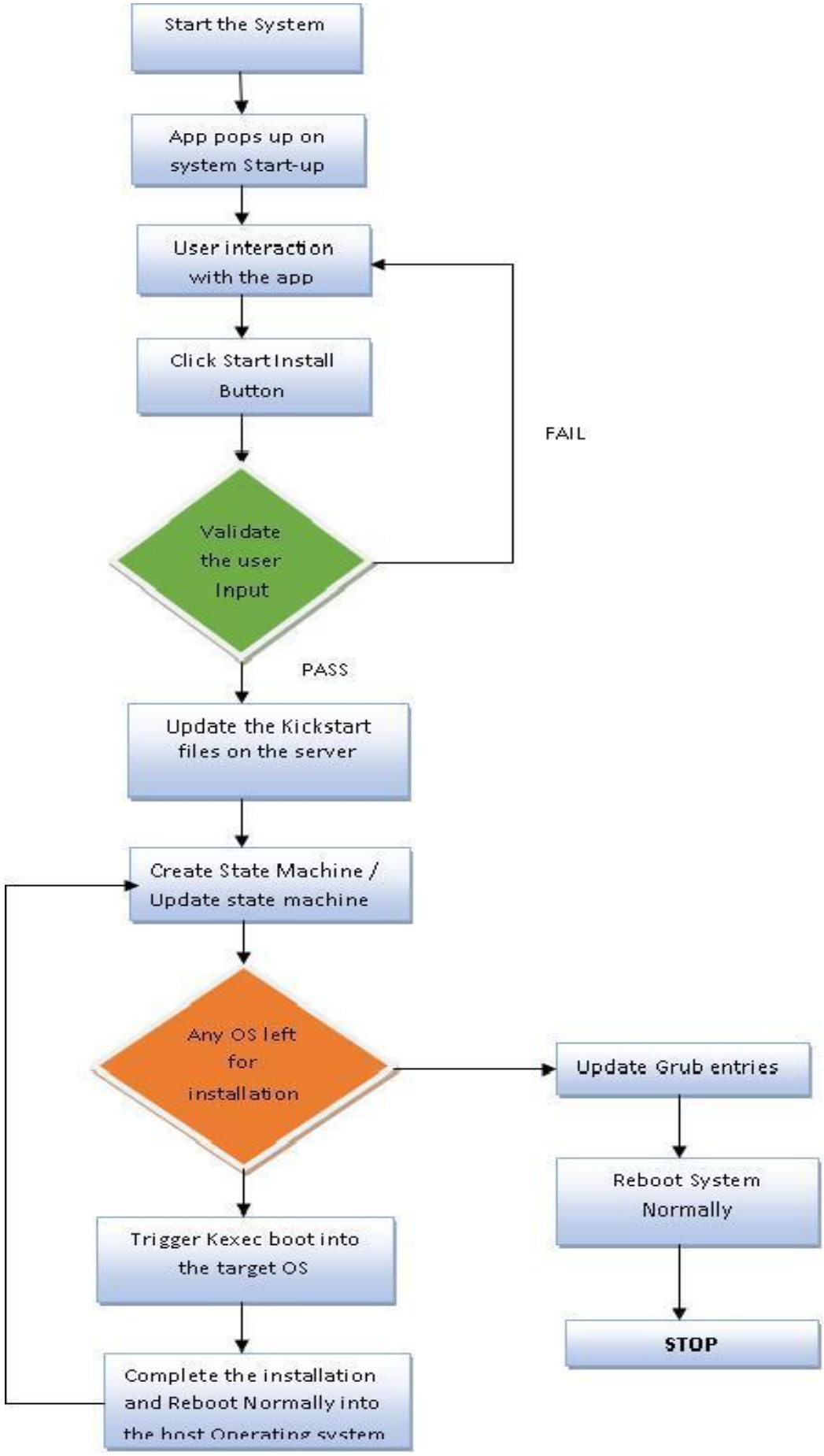


Fig. 3 Flow Chart

7. CONCLUSION

Using the above proposed system, various industrial project teams can automate the installation process of multiple operating system back to back, while at same time concentrating on the development part like coding. This will not only save the time of the engineer but would also increase the productivity of engineer.

ACKNOWLEDGMENT

I would like to express my deepest thanks to few people who have helped me a lot in designing and working with the project, without whom it would have been impossible for me to come up with such a good solution for automating the manual task of operating system installation. Few people I would explicitly like to name them are

1. Mr. Dhaval Dave
2. Mr. Bibhuty Prusty
3. Mr. Steven Yang
4. Mr Manish Katoche

And also a special thanks to all my friends and colleagues for been so supportive.

REFERENCES

[1] Andy Pfiffer Open Source Development Labs, Inc.., *Reducing system Reboot Time with Kexec,* 15275 SW Koll Parkway - Suite H Beaverton, OR 97006 503-626-2455x30
[2] Red Hat Documentation, *http://www.redhat.com/docs*
[3] D. P. Bovet, M. Cesati, *Understanding the Linux kernel,* O'Reilly Press, 2002
[4] J. Wei, T. Long, and F. Liu, ―Real-time net-booting system in large-scale DSP network,‖ Presented at the IE EE Radar Conference, 2006.
[5] M. Bar Linux File System, McGraw-Hill 2001
[6] Ananthakrishnan Ravi and Roopak Venkatakrishnan *Multi-Level Kick Start Server* International Journal of Computer Theory and Engineering, Vol. 5, No. 2, April 2013
[7] Patrick Mochel. The Linux Kernel Device Model. In *Linux.conf.au*, Perth, Australia, January 2003.
[8] Haogang Chen Yandong Mao Xi Wang Dong Zhouy Nickolai Zeldovich M. Frans Kaashoek Linux kernel vulnerabilities: State-of-the-art defenses and open problems MIT CSAIL yTsinghua University
[9] Amon Ott. The Rule Set Based Access Control (RSBAC) Linux Kernel Security Extension. In *Proceedings of the 8th International Linux Kongress*, November 2001.
[10] Customizing Boot Media for Linux* Direct Boot, October 2013White Paper Bruce Liao Platform Application Engineer Intel Corporation
[11] The GNU GRUB manual "*The GRand Unified Bootloader* ", version 2.00, 23 June 2012. Gordon Matzigkeit Yoshinori K. Okuji Colin Watson Colin D. Bennett This manual is for GNU GRUB (version 2.00, 23 June 2012)

