International Journal on Data Science and Technology
2017; 3(1): 8-15
http://www.sciencepublishinggroup.com/j/ijdst
doi: 10.11648/j.ijdst.20170301.12
ISSN: 2472-2200(Print); ISSN: 2472-2235(Online)

INL (Is Not Linux): Challenges of Building a New FOSS Operating System

Lucio Agostinho Rocha

Department of Software Engineering, Federal University of Technology (UTFPR), Dois Vizinhos, Brazil

Email address:
luciorocha@utfpr.edu.br

To cite this article:
Lucio Agostinho Rocha. INL (Is Not Linux): Challenges of Building a New FOSS Operating System. *International Journal on Data Science and Technology*. Vol. 3, No. 1, 2017, pp. 8-15. doi: 10.11648/j.ijdst.20170301.12

Received: April 12, 2017; **Accepted**: April 21, 2017; **Published**: May 22, 2017

Abstract: This article describes the main considerations to automate the building process to create new operating systems based on Linux From Scratch and Beyond Linux From Scratch projects. It is necessary to provide automation of this building to simplify, fix a lot of configuration bugs, and reduce the inherent effort to create a functional operating system. Our purpose is offering a Free Open Source Software (FOSS) with concise descriptions to guide the building of these operating systems. One of the major challenges is the necessary effort to deal with packagesand its dependencies. As a consequence, it was developed an optimized installer that follows rigorously the official LFS documentation to generate bootable virtual machines.

Keywords: FOSS, Operating System, Linux

1. Introduction

The goal of our paper is to answerthe question: how can webuild a functional operating systemthat follows the philosophy of Free Open Source Software (FOSS)? In order to answer this question we conduct this current research.

In this research, it was developed an automated way to create a new operating system based on *Linux From Scratch* (LFS) [2] and Beyond LFS (BLFS) [6]. Although there are some works that use the same documentation of these latter projects to automate the building of Linux operating systems, we highlight that our approach is a more convenient way to complete successfully the whole procedure. This research follows rigorously the documentation of LFS and BLFS projects.LFS is an *online* project that explains the step-by-step to build complete and functional Linux operating systems. As long as exist a lot of packages on severalservers in world wide web, the LFS project aims as a detailed guideabout how to find them, configure and install them. LFS follows rigorously POSIX.1-2008, *Filesystem Hierarchy Standard version* 3.0 *Draft* 1 (FHS), and *Linux Standard Base* (LSB). LSB has Five patterns: Core, C++ compiler, desktop environment, real-time languages, and printing. A basic LFS system has the following packages: *Autoconf, Automake, Bash, Bc, Binutils, Bison, Bzip2, Check, Coreutils, DejaGNU, Diffutils, E2fsprogs, Expect, File, Findutils, Flex, Gawk, GCC, GDBM, Gettext, Glibc, GMP, Grep, Groff, GRUB, Gzip, Iana-etc, Inetutils, IProute2, Kbd, Kmod, Less, Libpipeline, Libtool, Linux Kernel,M4, Make, Man-DB, Man-pages, MPC, MPFR, Ncurses,Patch, Perl, Pkg-config, Procps-NG, Psmisc, Readline, Sed, Shadow, Sysklogd, Sysvinit, Tar, Tcl, Texinfo, Udev, Util-linux, Vim, XZ Utils e Zlib*. On the other hands,*BLFS* is a complementary document that describes the additional steps to bring more features to LFS systems, such as graphical user, interface, thirdy-party softwares, multimedia, and many others packages and its necessary dependences.

LFS operating systems are created through an other Linux operating system. Although there is a lot of documentation about this process, it is very common to find errors during installation. Main reasons for thisare differences between distros, change of the source code links in Web, update of distro, many differences of building packages in Linux distributions, and mainly the high compilation time. Minor seemingly unimportant errors compromise the whole LFS installation process.

We emphasize that the building is highly dependent on the proper configuration of the target platform. In the face of these problems, we have developed a new simplified automation toolkit of the LFS and BLFS processes for the creation of FOSS operating systems. In this research it was created the automation to build the operating system known

as INL, and its name is a recursive acronym for Is Not Linux. Theobjective is to simplify and provide a complete and reliable alternative to using other operating system kernels beyond the Linux kernel. But, in this article, we focus on the challenges of creating INL system with the minimal set of tools of LFS and BLFS. This system supports a LXDE graphical interface [8], Mozilla Firefox 50.0 browser [9], and multimedia support. INL is designed for general purpose on 64-bit x86 architectures.

Remain sections are presented as follows: Section 2 describes the background of the project, and reports related works; Section 3 describes the methodology of building of the INL operating system; Section 4 discusses the challenges of creating the system, and presents results; Finally, Section 5 makes the final considerations.

2. Background

This section describes the work related to the LFS project. Figure 1 shows the graphical interface and the Firefox browser running on the INL operating system. Although there are a lot of fixes for many issues during setup process, it is observed that LFS automation is still little explored.Nexttopics describe briefly the many related projects as follows:

*LFS:*it is the basic design of building LFS systems. LFS project explains the step-by-step for complete and functional installation of the new Linux operating system software compatible with various international standards.

BLFS (*Beyond LFS*): it is the project that explains the addition of new features to the LFS system. Required, recommended, and optional tools and configurations are described.

CLFS (*Cross LFS*)[3]: it is the project that explains the creation of LFS systems for different architectures. Processes for the x86, x86, 64-bit, SPARC, MIPS, PowerPC and ARM architectures are currently described.

HLFS (*Hardened LFS*)[7]: it is the project for the security aspects of configuration and security update of packages on LFS systems.

ALFS (*Automated LFS*)[5]: It is a project for automating the LFS installation. The project is also known as jhalfs, with initials in tribute to its creator Jeremy Huntwork, has the latest update in 2007 and currently does not run correctly on newer versions of Ubuntu, despite being present on the LFS LiveCD. It is based on a set of Shell Bash scripts that use Subversion and xsltproc to download the source packages, describe them in XML, extract them, and install them. Jhalf scripts also allow you to fetch packages from the base system instead of fetching them from the Internet. The authors report that there is a possibility of recovery if errors are found in construction. Package management is likewise supported. An earlier version known as nALF by author Never Has was written in C, but was discontinued. The current version of jhalf has many undocumented bugs and its extensive set of software makes hard maintenance for it.

LFScript (Linux From Script)[13]: it is an unofficial project that provides scripts to automate the installation of LFS. There has been no recent activity in the project since 2014 and the link to the LiveCD of the latest version is not available. The project also has many undocumented bugs in the installation. The main criticism is that not all package installation scripts comply with what is recommended by the official LFS website, which is a potential source of failures. The scripts are divided into many folders. On the one hand, this simplifies the update of the download links, but makes it difficult to see the dependencies between the packages.

Related at these other solutions we emphasize that INL automated installer follows strictly the procedures described in the official website of the project with a few sets of scripts.The process for creating the INL system assumes that all packages and patches have been previously searched. So far, only the ALFS project is recommended by LFS project.

Also, we note that related at these other projects; using INL toolkit of building scripts has a lot of advantages, as shows Table 1. Our toolkit has significant improvements related at these other researched projects: a) reduced amount of source code of the automated toolkit, and reduced amount of languages used to perform the building; b) strictly follows LFS and BLFS documentation; c) simplicity to understand, modify, and update the setup scripts of our toolkit; d) log of processes during the installation; e) well documented and recent information about installation. A comparative description about these features and other aspects are described in Table 1.

Table 1.*Comparative Description.*

	INL Toolkit	**ALFS**	**LFScript**
Stable Versions	**[2016 – 2017]**	**[2005 – 2007]**	**[2011 – 2014]**
Source Code Language	Shell Script	C, Shell Script, Makefile, XSL, other libraries	Shell Script, Makefile, libraries, other binary files
Amount of Files	12 files	Above of 100 files	Aboveof 50 files
User Interaction during Install Process	Yes	No	No
Show of Installation Progress	Yes	Yes	No
Effort to Update Files From Installation	Reduced (It follows the LFS procedure)	High (It not follows strictly the LFS procedure)	High (It not follows strictly the LFS procedure)
Platform Target	x86_64	Generic	Generic
Logs	Yes, and describes strictly issues about LFS process.	Yes, but not involves details specifically about LFS process	Yes, but not involves details specifically about LFS process
Internet Access to Get Additional Files	Once at the initial step, only to *download* LFS+BLFS packages	*Download* of XML files + LFS+BLFS packages	Distributed during the installation.
Documentation	Yes	Yes, but it is not recent	Yes, but it is not recent.

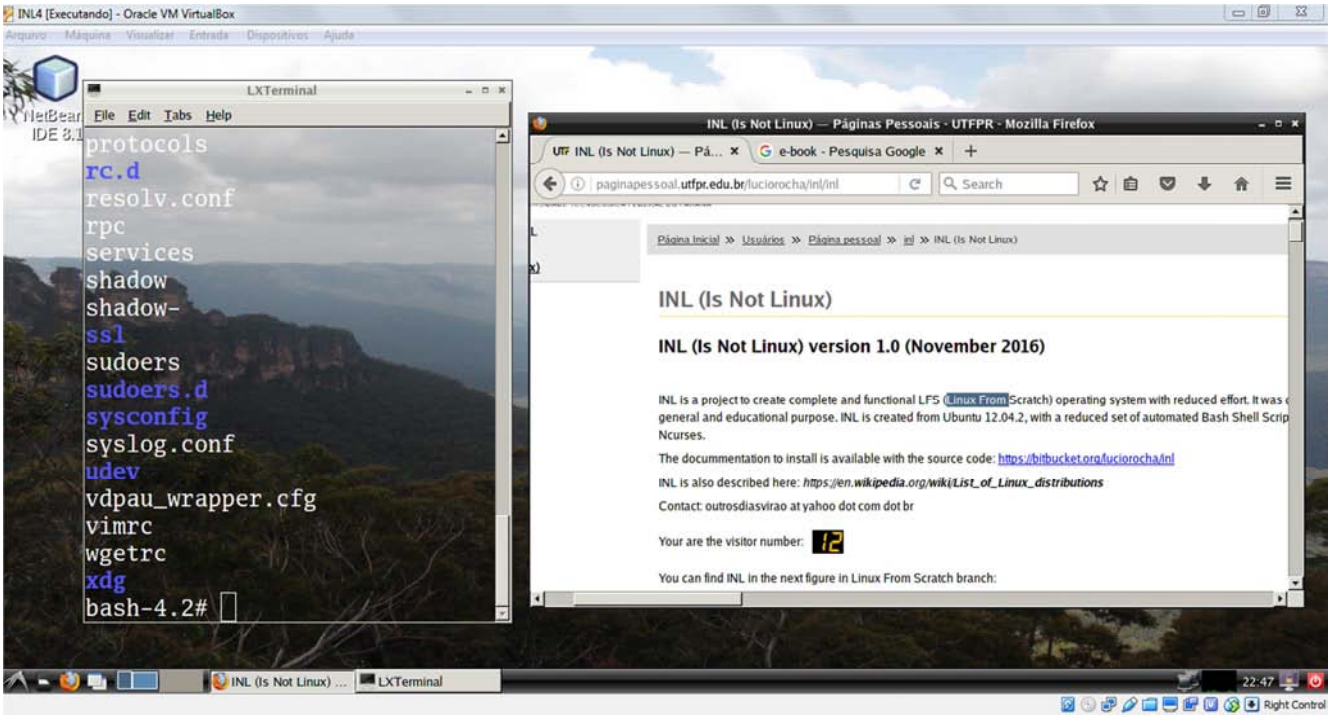

Figure 1. *INL Operating System Running Inside VirtualBox.*

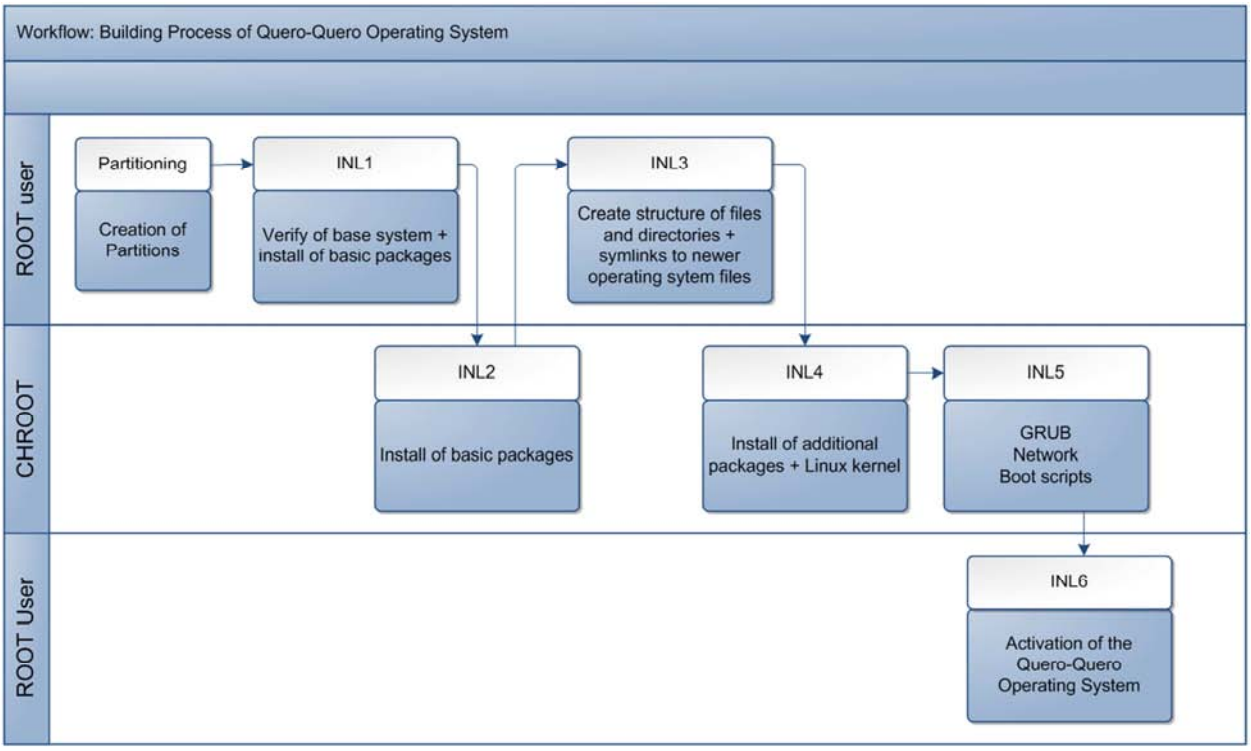

Figure 2. *Workflow of the INL Building Process.*

3. Methodology

A complete description of the process is described indetails in the documentation available on the project website [1]. Because of this, in this section we focus on the workflow description of the automation process. In this methodology, the INL operating system will be created from scratch through well-defined parts, explained as follows:

A. LFS Building Process

a. *Partitioning:* Building the INL operating system is done by previously creating your primary partition. Many alternatives are possible to create partitions, for example: fdisk, mkfs, Gparted, and others. We use a conventional Ubuntu installer to inform the distro

installation process of the two primary partitions: the first for Ubuntu and the second for the INL, in addition to one (1) common Swap partition for both. After installing Ubuntu, just run the main script INL1.sh. The remains of process is automated, and uses automatic root and chroot user switching. Several steps are performed with root user permissions in a protected environment (chroot) to avoid toolchain conflicts with packages that are already installed on the host system. At the end of the process will be created the INL system with LFS software, LXDE graphical interface and Mozilla Firefox 50.0 browser. The workflow in Figure 2 shows this process.

b. *INL1 (Root user):* This is the main program that controls the other programs. Its function is to coordinate the steps of the installer and treat the dependencies.
c. *INL2 (Chroot Environment):* Installation of the minimum packages in the base system, necessary for compiling the packages in the next steps.
d. *INL3 (Root user):* Preparing the chroot environment. The chroot environment runs an interactive shell with a new root directory entered by the user. This means that changes in this environment will not affect the original environment. This is necessary to prevent new installed packages from conflicting with those already in the parent directory structure. In this environment the new file and directory tree is created, initial device nodes, / dev assembly, assembly of kernel virtual file systems, symbolic links to directories, devices and shared libraries, and creation of users, groups and their permissions.
e. *INL4 (Chroot Environment):*This step is about the installation of the INL packages, but now in the newly installed Bash environment in the chroot environment. The minimum packages will be compiled in this protected environment.
f. *INL5 (User chroot):* This step is about the installation in the chroot environment. This step will install the Linux kernel, and add-on packages.
g. *INL6 (Root user):*This step is about the completion of the installation in the chroot environment, with network preparation, virtual terminals, file partition table (fstab) and boot manager (Grub) for the first boot. At the end of this step you have the functional and bootable INL system. We complement the installation with BLFS in the second part.

B. BLFS Complementary Process

In the previous steps a basic LFS system will be generated. In this next part the INL is complemented with BLFS for the addition of more functions:

a) *INL_pos1*: Preparation for automatic discovery of devices with udev.
b) *INL_pos2*: Installation of VirtualBox Guest Additions.
c) *INL_pos3*: Network configuration for Internet access.
d) *INL_pos4*: Installation of basic packages for graphical interfaces.
e) *INL_pos5*: Installation of the graphical interface LXDE and Mozilla Firefox.
f) *INL_pos6*: Multimedia support (sound and video drivers), Adobe Flash Player, OpenJDK, and additional packages.

The first part of the LFS process, which uses 6 scripts, with time configuration about 8 hours on hardware with 2 Intel i3 processors and 2GB of RAM. Major steps require few user involve during installation. At the conclusion of this process the bootable operating system will occupy little more than 5GB of disk space. The second part installs a minimal set of BLFS and the duration was not estimated. At the conclusion of this other part we will have a LFS and BLFS system with LXDE graphical interface and the Mozilla Firefox 50.0 browser.

4. Results and Discussion

There are many undocumented details that hinder the complete and functional creation of LFS and BLS systems.In this section a discussion and important recommendations for the success of the configuration, installation and execution of the new system are made. The authors of the LFS project suggest as prerequisites a minimum knowledge of Linux to use the shell for various commands, as well as minimal knowledge about how to install Linux software. As for hardware, the main architecture of LFS is an AMD / Intel x86 32-bit and 64-bit x86 processors, and it also supports cross-compiling for other architectures.The process of installing the LFS system uses another Linux operating system, which we will call the base system. This base system provides the programs needed to build the new system. The base system should be compatible with GNU / Linux tools, such as a Slackware, Ubuntu, Red Hat, Fedora, Debian, Mandriva, Suse or other Linux distribution. Empirically, it is suggested using a new installed host system, which suggests using virtualization to test multiple of them if necessary. Unless strictly necessary, it is suggested performing the installation for recent hardware with 64-bit architectures with at least 2GB of RAM, noting that 32-bit distributions also work on 64-bit architectures. Each base system has its own set of tools, but it is recommended to install the packages and patches for the LFS version to be followed. It is observed that different base system distributions generate different configuration processes as well. To start the LFS installation it isnecessary to obtain the following minimum set of packages in the base system: Bash, Binutils, Bison, Bzip2, Coreutils, Diffutils, Findutils, Gawk, GCC, Glibc, Grep, Gzip, Linux kernel, M4, Make, Patch, Perl, Sed, Tar and Xz.Some recommendations are described below:

1) Minimal setup: For minimal LFS system installation without graphical interface, we suggest at least 15 GB of disk space and at least 2GB of RAM in multiprocessed Intel architectures. But for an LFS system with graphical interface, we suggest at least 30GB of disk space. For a virtualized installation, changing the disk size is straightforward, but must be preplanned so as not to interfere with the installation

due to the lack of disk space of the virtual machine. It is important to note that the recommended packages for a particular version of LFS must be strictly used. The order of installation of the packages must also be strictly adhered to it;

*2) Processing time:*At least 8 hours will be required to complete the entire process for the first boot of LFS with the INL installer as long as no errors are found. The LFS site generally reports this value with Standard Build Units (SBUs) to compare build times against the first compiled package, in this case the Binutils package. For example, if the Binutils compile time was 10 minutes, another 3.5 SBU pack will take an average of 35 minutes to compile on the same platform;

*3) Optimization aspects:*The software source code can be removed after installation in order to increase the available disk space. For example, only the uncompressed Mozilla Firefox 50.0 software source code will occupy approximately 5GB of disk space and you will not need to keep this folder after installation. It is important to strictly follow the steps outlined in LFS and, in case of an error, do not proceed until the problem is solved. This consideration is significantly minimized with INL scripts. It is also important that all packages are downloaded before using them to optimize the installation time, and making checks of the packagesonly when strictly necessary (eg.: verify if gcc binaries are properly built). The LFS authors themselves report that this check is optional for most users. There arerelatively good chances that the package has been properly compiled if there are no errors during compilation.Another note is to avoid installing unnecessary documentation to save disk space. On the other hand, it is suggested that you do not remove packages with the package source code before finalizing the installation of the entire LFS, unless strictly informed in the process, to avoid linking problems before boot. In this regard, it is important to leave the host active during critical steps, and not create snapshots before the first boot. Snapshots should not be created with active shared folders during the installation process because they corrupt the new system in the virtual machine. Another consideration is that the major "villains" that burden the LFS build time are the GCC-related packages. So it is interesting to make the most of the hardware features: using make -jX allows you to use X processors to build most packages.

*4) Disk resizing:*As illustrated in Figure 3, after installing the packages it is useful to remove the folders with their source code to conserve disk space. However, the resizing of virtual machines has some details [11]: a) newer versions of Linux identify the boot hard disk by the drive ID; B) Size adjustment of VDI virtual disks with VBoxManage is only incremental (without reducing image size); C) simple cloning is not enough to make the disk booze again. Because of these factors the following procedure was proposed. After the installation, INL will be on the /dev/sda2 partition, because a new partition has been created to install it, and the /dev/sda1 partition with Ubuntu can be removed. To use a disk with only the /dev/sda1 partition and to resize (reduce) the space used, a combination of actions with the GParted and Clonezilla softwares (or *fdisk*, *mkfs*, and *resize2fs* commands) is suggested as follows:

1) Use two images on the VirtualBox SATA controller. The original image is connected to Serial ATA 0 (SATA0), and the new one is connected to SATA1, but the latter is not contained on the disk; 2) Gparted: create a /dev/sdb1 partition on disk in SATA1.Without creating the partition table on that disk, will not possible to do cloning; 3) Gparted: Resize the partition /dev/sda2, where INL is installed on SATA0. Note that the two partitions must be the same size; 4) Clonezilla: Cloning the /dev/sda2 partition to the /dev/sdb1 partition.
2) After cloning, it is still not possible to boot to the new disk in /dev/sdb1, but the MBR (Master Boot Record) partition table will be present. It is necessary to reactivate your access via Grub. For this, use the disk in SATA0, which has the INL in /dev/sda2:
   a) From the Grub boot menu in /dev/sda2, inform with the edit option to boot /dev/sdb1. The boot will be performed on the new cloned partition.
   b) After the boot of /dev/sdb1:
      a. Update of /etc/fstab file: /dev/sdbX entries to /dev/sdaX, where X indicates the partition number.
      b. Update of /boot/grub/grub.cfg file: change entries to (hd0,1), and /dev/sda1.
      c. With the image that was booted (/dev/sdb1), type in Bash: *grub-install /dev/sdb*
      d. The Grub manager will be re-installed on the new partition, and will make it bootable. Note that the disk in SATA0 must be removed, and the change of the SATA1 disk to SATA0 in the VirtualBox manager has been performed.

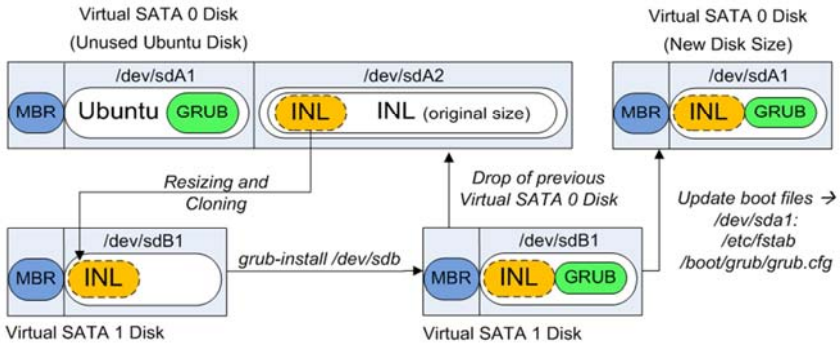

Figure 3. *Cloning, Resizing and Restarting of Grub.*

5) Network Virtualization: VirtualBox Guest Additions extends the possibilities of accessing the system through shared folders and host machine network access. Additional BLFS packages are downloaded to the host machine and sent to the INL guest machine through shared folder. NAT network access will not work with VirtualBox PCnet interfaces, but for Intel / PRO MT and related interfaces. The MAC address of the INL network interface should be the same as the NAT interface reported for the virtual machine in VirtualBox. After the first boot the Grub boot manager still allows access to the 2 operating systems, but we suggest deleting the primary partition of the base system and resizing the INL partition to occupy that additional space.

*6) Building of INL LiveCD:*After the creation of LFS + BLFS it is interesting to offer the distribution of the system in a media, be it a liveCD or a USB drive. We illustrate this boot process in Figure 4. This process uses Isolinux, which is a boot loader for images, CDs and floppies that allows you to boot kernels of different operating systems with parameter passing. In summary, in the boot process, there is nowriting permission enabled on the CD. Due to this issue, the boot strategy is to create a set of temporary directories with write permission in RAM. The process starts as follows:a) isolinux boot; b) kernel uncompress; c)basic Linux system loading (that is compressed in initramfs_data file);d) load of the complete filesystem that is compressed inside root.ext2 file of the liveCD. The next steps are the loading of the /dev directories, recognizing of other devices, and passing of the root control (with root and chroot commands) to the liveCD file system loaded in RAM (/etc/rc.d/rcS.d/S00createramdisk). In the remastering process, this LiveCD only becomes bootable (via the create-initramfs command from the liveCD itself) if the kernel modules are available in /lib/modules. Also, it is necessary to use two partitions in the base host to create this LiveCD filesystem: one source partition to load source files and one destination partition for the remastered image files. The complete description of this process is in reference [1].

Figure 4. *Boot process via LiveCD.*

7) Linux compatibility: We believe that any Linux application is able torun natively in the INL environment. Figure 5 illustrates the MobileSim 0.7.3 [12] robotic application simulator in the execution of a Java application written in the Netbeans IDE 8.1, all installed in the INL. Figure 6 shows the execution of the simulator V-REP [14] also running in the INL. It has been found that the process of installing additional packages (BLFS) is the most time-consuming process because it is not sequential and, unlike other distros, there is no package manager that simplifies the installation. BLFS systems present a set of tools and configurations that complement the minimum LFS. Therefore, the required disk space will depend on the needs and characteristics intended for the new operating system. In BLFS the biggest challenge is to correctly enable the graphical interface, because even the simplest interface depends on more than 200 software packages with mandatory, recommended and optional dependencies. We suggest installing mandatory packages as well as recommended packages, and avoid installing optional packages unless required. Using the INL installer greatly simplifies this step because a number of dependency bugs and settings not mentioned in the BLFS documentation are resolved. Finally, the Mozilla Firefox 50.0 browser is not buildable in almost all 32-bit architectures [9][10]. It has been empirically observed that at least 2 Intel i3 processors and 2.5 GB of RAM are required for standard build.

5. Conclusions and Future Works

Our article presents a methodology to build acomplete FOSS operating system. This automation is important because simplify a lot of configurations that are necessary to correct compilation of its many softwares related and its dependencies. We argue that this process that is described in this article is better than the other studied projects in many aspects: a) reduced amount of scripts; and employ of shell scripts as installation language; c) strict compromise in follows LFS and BLFS; d) simplicity to modify/update the source code of thetoolkit scripts; e) logs during the installation; f) well documented and recent information about installation. We develop an automated software to prepare and compile each software that is necessary to create this new operating system. The automated process of creating new operating systems with INL provides a simple, expandable, complete, and reliable alternative for creating new LFS and BLFS systems. Creating from scratch operating systems is only recommended for users seeking alternatives in the face of the many limitations of performance of conventional systems.

Although many operating systems are built to optimize the capabilities of the target platform, there are also many disadvantages: the installation process will require an intricate set of configurations and fine control of the versions and their dependencies. As future work is being considered modifications in the toolkit scripts to provide more interactivity with the users.

Acknowledgment

The author gratefully acknowledges the contribution of the Grupo de Pesquisa em Engenharia de Software e Informática(GPESI) - (http://dgp.cnpq.br/dgp/espelhogrupo/6364090264037055) a Brazilian research group.

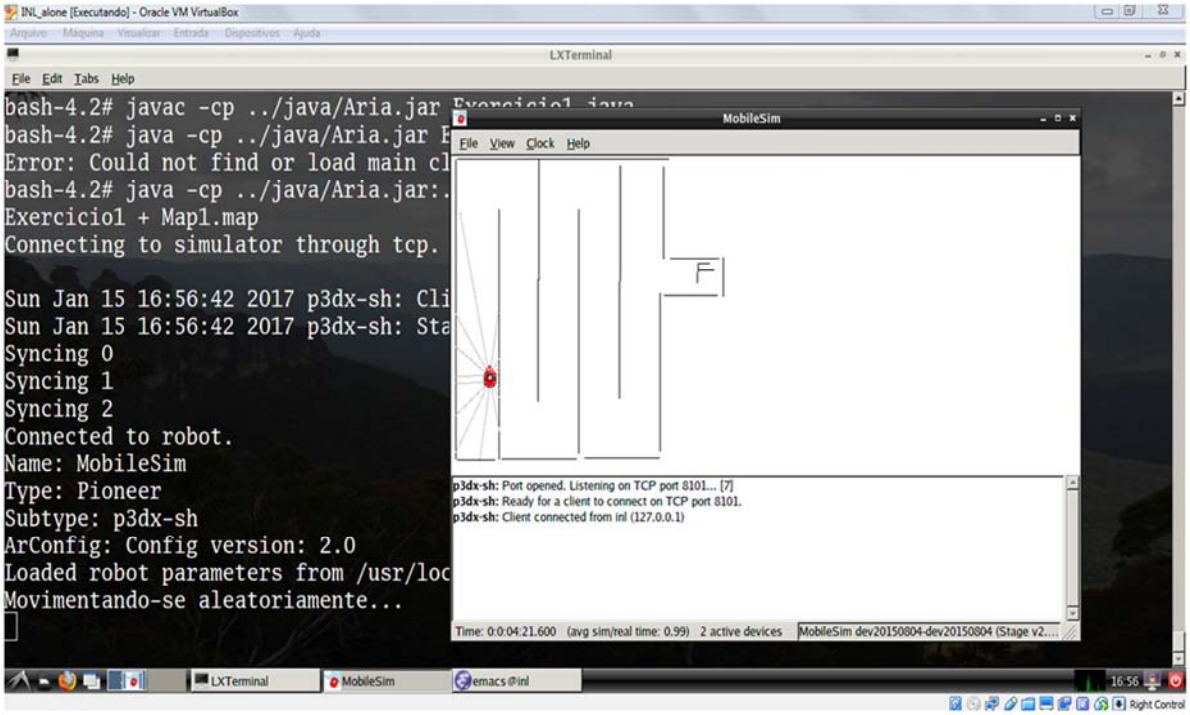

Figure 5. *Java Application with MobileSim Robotic Simulator in INL.*

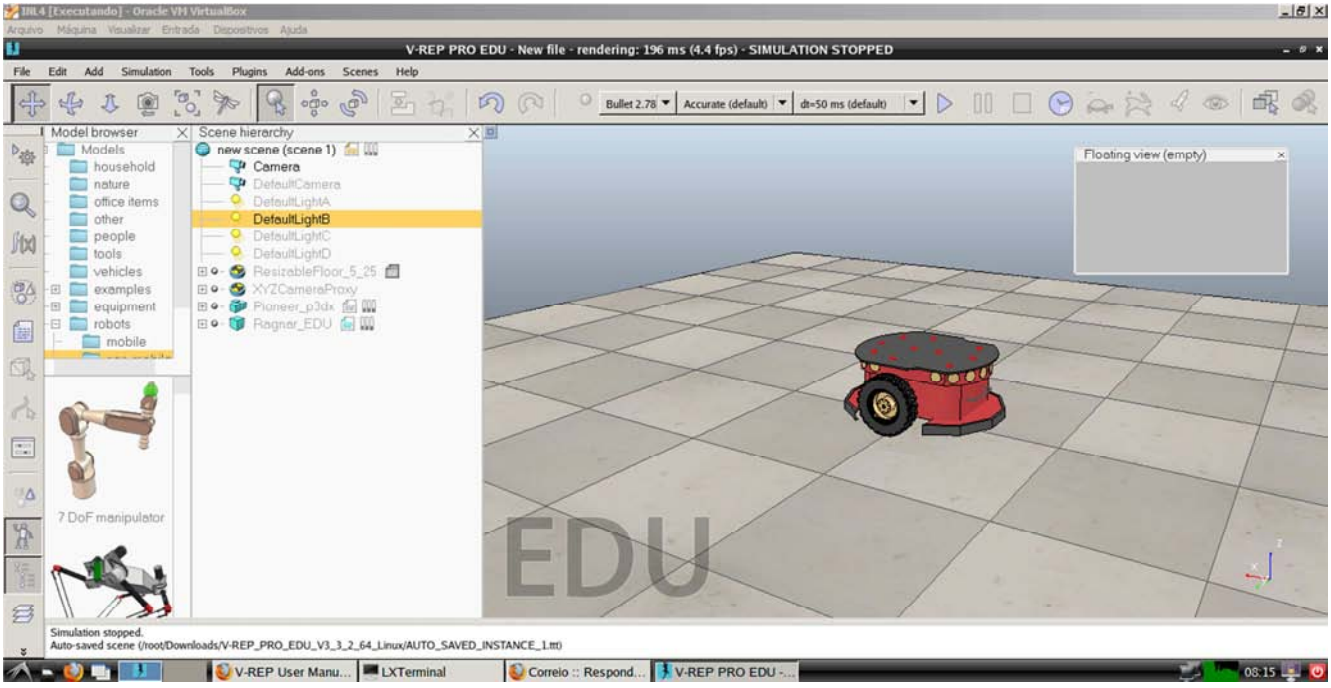

Figure 6. *Robotics simulator V-REP in INL.*

References

[1] L. A. Rocha, “INL (Is Not Linux)”. [Online]: http://paginapessoal.utfpr.edu.br/inl/inl, 2017.

[2] M. Burgess, and B. Dubbs, “Linux From Scratch Versão 7.5 Created by Gerard Beekmans”, [Online e-book]. Available: http://linuxfromscratch.org, 2017.

[3] R. Oliver, J. Gifford, J. Ciccone, et al., “Cross Linux From Scratch”. [Online], Available: http://trac.clfs.org, 2016.

[4] P. Gerum, K. Yaghmour, J. Masters, G. Ben-Yossef, “Building Embedded Linux Systems”. O’Reilly, 2002

[5] J. Huntwork, “Automated LFS”. [Online] Available: http://linuxfromscratch.org/alfs, 2007.

[6] B. Dubbs, D. R. Reno, DJ Lucas, et al., “Beyond Linux From Scratch”. [Online], Available: http://linuxfromscratch.org/blfs, 2016.

[7] R. Connoly, M. C. Esparcia “Hardened Linux From Scratch”. [Online], Available:http://linuxfromscratch.org/hlfs, 2016.

[8] LXDE. org, “A lightweight X11 for desktop environment”. [Online] Available: http://lxde.org, 2016.

[9] Mozilla Foundation, “Mozilla Firefox”, [Online] Available: http://mozilla.org/firefox, 2017.

[10] Mozilla Developer Network, “Building Firefox”, [Online] Available: https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Build_Instructions/Simple_Firefox_build, 2017.

[11] Oracle Corporation,“Oracle VM VirtualBox – User Manual – Cloning disk images”, pp. 92, [Online], Available: http://www.virtualbox.org, 2016.

[12] MobileRobots Inc, ”MobileSim – Simulator for MobileRobots / ActivMedia Robots”. [Online] Available:http://robots.mobilerobots.com, 2013.

[13] Marcel Van den Boer, “Linux from Script”. [Online] Available:https://www.lfscript.org, 2016.

[14] Coppelia Robotics, "V-REP PRO Edu - Virtual Robot Experimentation Platform". [Online], Available: http://coppeliarobotics.com, 2016.