Introdução ao Minix

Projeto 1 de Sistemas Operacionais (MC504)

Randerson Araújo de Lemos, R.A. 103897; Victor Lago, R.A. 194784

***Abstract*—This document presents the main results derived from the first contact of the authors with the installation of the Operating System (OS) Minux into a virtual machine and with the recompilation kernel process of the OS itself.**

I. Introduction

OPERATING system (OS) is a software designed to manage computer resources such as those made available by the disk, graphic cards, network, processors, etc. With these roles, an OS makes the bridge between the hardware and the software, abstracting most of the complexity associated with the correct usage of the first, allowing users and developers to focus on the goal of their projects.

Taking this into account, it seems like an appropriate starting point for those who intend to study operating systems to install an OS by itself and start to experiment with its kernel, i.e. the codes that make up the operating system. Therefore, this report contains the main results obtained by the authors in relation to two main tasks: 1. Installation of the operating system Minix into a virtual box; 2. Recompilation of the Minix kernel after changes in its files. A detailed description of these tasks is present in the sections II and III.

II. Installation

The Minix installation was done according to the documentation provided with the project proposal. Some of the difficulties encountered during the installation were as follows:

- It was not clear how to properly remove the Minix iso file from the virtual box machine after installation.
- While modifying the ”sshd_config” file, it took some time for us to notice that the line ”PermitRootLogin” was commented.
- At the first moment, It was faced problems to enabling ssh connection from the host machine due problem in the configuration network bridge mode from the VirtualBox.

Nevertheless, after 3 or 5 re-installation cycles all problems were resolved and the OS was made ready for the recompilation of the kernel. Appropriated snapshots were taken.

III. Kernel recompilation

In this section, we describe the main results of the activities: changing the banners to the message requested by the project and changing the exec.c file to show the path of the executed commands. Some general challenges encountered during the process of implementing the project were: the kernel's time of compilation was longer than expected, sometimes greater than 4 hours; and the OS version that we used didn't contain the correct options for Brazilian keyboards, which made it difficult to navigate through the files on the virtual machine.

A. Changes in banners

In this part of the project, we needed to change two banners from the kernel: the first one is the message that appears after booting the system, and the second one appears after logging into the system.

The first one was a little harder than the second. First, we tried to look on the internet to find out which OS files we needed to change, so we could change the message that appears after booting the system, but we couldn't find it due to the lack of information online.

After this, we tried another strategy, using the command grep to find the string pattern that we needed to substitute for the new string. We used the following command for this:

```
grep –Rnw /usr/src/ –e <pattern>
```

Where **pattern=”Minix 3.4.0. Copyright 2016, Vrije Universiteit, Amsterdam, The Netherlands”**. However, we were again unsuccessful in finding the appropriate file with this command. The reason the search strategy failed comes from the string pattern tried initially because the file searched doesn't have the exact string pattern written on it, as shown in Figure 1.

Fig. 1. File main.c in the banners text region.

We overcame the problem by changing the string search pattern to **pattern=”Vrije Universiteit”**. With a simpler pattern, it was easier to find the main.c file from the kernel folder, since it was the only file with such a string pattern. After changing the necessary files, we successfully recompiled the kernel.

The second one was not that difficult to change, with a little bit of research we found out about the Message of The Day (MoTD), and with that knowledge using the command ”find” within the Minix's files, we were able to find the path for the file that we needed to modify **”[...]/minix/etc/motd”**. Figure 2 shows printed strings after login.

Fig. 2. Welcome banner after login.

B. Changes in exec.c

Different from the process of finding the banner's files, finding the exec.c file was not difficult, the problem lay in determining which exec.c file was the correct one that we needed to modify since there were more than one, as shown in Figure 3.

```
minix# find ./ -iname exec.c
./bin/csh/exec.c
./bin/ksh/exec.c
./bin/sh/exec.c
./minix/servers/pm/exec.c
./minix/servers/rs/exec.c
./minix/servers/vfs/exec.c
./sys/arch/i386/stand/lib/exec.c
```

Fig. 3. Files exec.c returned be search.

Moreover, the compilation time was more time-consuming than expected, after 30 hours of compilation and modifying the code, we finally found the correct exec.c we needed to modify: **/usr/src/minix/servers/vfs/exec.c**. Adding the command **printf("Executando: %s\n", fullpath);** was enough to show the commands as requested. The modified code is shown in Figure 4.

```
/*===========================================================================*
 *                                get_read_vp                                *
 *===========================================================================*/
static int get_read_vp(struct vfs_exec_info *execi,
  char *fullpath, int copyprogname, int sugid, struct lookup *resolve, struct fproc *fp)
{
/* Make the executable that we want to exec() into the binary pointed
 * to by 'fullpath.' This function fills in necessary details in the execi
 * structure, such as opened vnode. It unlocks and releases the vnode if
 * it was already there. This makes it easy to change the executable
 * during the exec(), which is often necessary, by calling this function
 * more than once. This is specifically necessary when we discover the
 * executable is actually a script or a dynamically linked executable.
 */

       printf("Executando: %s\n", fullpath);
```

Fig. 4. Files exec.c modified.

After recompiling the kernel with this modification, executing any program on the OS shell will print its full path on the command line.

```
Executando: /bin/hostname
Executando: /usr/pkg/bin/vim
Executando: /usr/libexec/ld.elf_so
mExecutando: /usr/bin/clear
Executando: /bin/sh
Executando: /sbin/tput
Executando: /usr/sbin/tput
Executando: /bin/tput
Executando: /usr/bin/tput
Executando: /bin/ls
Executando: /usr/bin/clear
Executando: /bin/sh
Executando: /sbin/tput
Executando: /usr/sbin/tput
Executando: /bin/tput
Executando: /usr/bin/tput
Executando: /bin/ls
Executando: /bin/ls
Executando: /bin/cat
```

Fig. 5. Shell prints after command execution.

IV. Conclusions

After completing the activity, the main conclusions of the authors are:

- It is not trivial to manipulate the OS at the kernel level due to these pieces of software having extensive and hard-to-understand documentation.
- Making simple kernel changes can be hard. Between long compilation times due to the hardware being used, finding specific files within the kernel, and compilation errors, various things can generate problems.
- Simple kernel manipulations, such as the ones requested in this project, are a good starting point for learning about OS and kernel compilation.
- Showing the path of the commands fired is a good strategy for understanding more about how the OS works.

V. Work organization

The current work was done by both students with equal amounts of effort. All steps were realized together and with time for discussions, changes of ideas, and doubts.