SunS LFS Linux

Builing a custom Linux operating system from scratch

Supervised by: Prof. Mustapha Saidallah

By: Mohamed Abdelkoddous El Idrissi
TRI 2A, ISTAG Bab Tizimi, 2018-2019

Table of Contents

I. Preface

This work constituts my End of Training Project. It consists of two parts, the first part is the resulting operating system (named **SunS**) and the second being this document. In this document I will describe the steps followed to reach the final goal, the issues encountered and the solutions applied to them as well as the improvements that still can be brought in.

The resulting operating system can be run and tested on either a virtual machine or a real, physical one. However, it has only been tested on a Dell Latitude e6420 and Virtualbox 6.0, so it is unknown whether it will run on different hardware and virtualization software or not, especially since LFS instructions are optimized to build a Linux operating system for the specific hardware and architecture of the host system.

II. Introduction

1. The Linux Operating System family

Linux as made by Linus Torvalds is a kernel for an operating system. It is practically unusable by its own, so some other userland utilities are needed in order to allow the use of what the kernel has to offer. Must distributors choose to populate their userland with software from the GNU Project which I will talk about later. For this reason, some open source activists have strong opinion about the naming of these distributions and prefer to call them “GNU/Linux”.

2. The GNU Project and the GPL

Most of the programs that we will compile to build the SunS OS are from the GNU Project. The aim of it is to give computer users freedom and control over their computers. The software it makes is distributed under GNU Public License (GPL), a free and open source license which has many versions, notably GPLv2 and GPLv3.

The GPL gives the users the right to run, copy, distribute, study and modify the software released under it. Thus being described as free, free to do with the software whatever the user wishes. But it does not mean that the use of that software is always free of financial charges, only free from any restrictions while using it. It is free as in *freedom*, not as in *free beer*.

3. Linux From Scratch Project

Linux From Scratch (LFS) is a project that provides step-by-step instructions for building a custom Linux system, entirely from source code. It is a journey to learn the internal working of a Linux distribution and the indispensable packages it needs to run.

An LFS system is fairly minimal, but is designed to provide a strong base on which you can add any packages you want. Other similar projects (from the same community) that assists in extending and customizing a core LFS system exists to continue where LFS has finished.

One of those projects is Beyond Linux From Scratch (BLFS). It provides a broad range of instructions for installing and configuring various packages on top of a base LFS system, from building and changing to a shell different than Bash, such ZSH, to building and installing the whole Xorg dependencies in order to run a graphical server.

An older but less interesting project is the LFS Hints, which inspires the creation of BLFS. It collects a variety of hints to improve a base LFS system. Each hint is written by a different contributor independently as a text file and submitted to the project, which makes it a little messy and hard to navigate. But BLFS solved this problem as it is organized as a book.

The SunS OS has some packages which are not part of LFS but discussed in BLFS.

4. Needed Packages

To avoid redundancy, a detailed list of the packages the SunS OS need will not be listed here. But one can construct it from the chapters Building a clean toolchain and Building the SunS OS, that in addition to the inevitably necessary `linux-4.20.12` kernel.

III. Preparing The Environment

The host on which I will build the SunS needs to be prepared to start the building. It must have the necessary tools, and a partition to hold the system I will create from scratch.

1. Checking the host tools

The following packages are needed on the host system:

```
Bash-3.2
Binutils-2.25
Bison-2.7
Bzip2-1.0.4
Coreutils-6.9
Diffutils-2.8.1
Findutils-4.2.31
Gawk-4.0.1
GCC-5.2
Glibc-2.11
Grep-2.5.1a
Gzip-1.3.12
Linux Kernel-3.2
M4-1.4.10
Make-4.0
Patch-2.5.4
Perl-5.8.8
Python-3.4
Sed-4.1.5
Tar-1.22
Texinfo-4.7
Xz-5.0.0
```

The host system is an Arch Linux machine running on a Dell Latitude e6420. A Linux distribution is usually shipped with those tools, so it should not be an issue. As for Arch Linux, most of them are from the `base` and `base-devel` groups.

The version numbers listed represents the minimal version required. Newer versions work as well. For example, the host system kernel version is 5.1.3, although only 3.2 is specified. The same goes for other packages and I will be using newer versions.

The LFS book provides a script to test the availability of these tools. The script code will be attached to the appendix. Here is the output of running it:

```
bash, version 5.0.7(1)-release
/bin/sh -> /usr/bin/bash
Binutils: (GNU Binutils) 2.32
bison (GNU Bison) 3.3.2
yacc is bison (GNU Bison) 3.3.2
bzip2,  Version 1.0.6, 6-Sept-2010.
Coreutils:  8.31
diff (GNU diffutils) 3.7
find (GNU findutils) 4.6.0
GNU Awk 4.2.1, API: 2.0 (GNU MPFR 4.0.2, GNU MP 6.1.2)
/usr/bin/awk -> /usr/bin/gawk
gcc (GCC) 8.3.0
g++ (GCC) 8.3.0
(GNU libc) 2.29
grep (GNU grep) 3.3
gzip 1.10
Linux version 5.1.3-arch1-1-ARCH (builduser@heftig-19477) (gcc version 8.3.0
(GCC)) #1 SMP PREEMPT Thu May 16 20:59:36 UTC 2019
m4 (GNU M4) 1.4.18
GNU Make 4.2.1
GNU patch 2.7.6
Perl version='5.28.2';
Python 3.7.3
sed (GNU sed) 4.7
tar (GNU tar) 1.32
texi2any (GNU texinfo) 6.6
xz (XZ Utils) 5.2.4
g++ compilation OK
```

2. Creating a new partition

Many partitioning schemes can be adopted as a disk layout for Linux system. The key point is to separate the top level directories each on its own disk or partition, so for example you would have `/usr` on a separate partition, `/home` on its own independent one, and the same for `/var`, `/opt` and `/boot`.

Examples of disk layouts:

Example 1:

Mount point	Partition	Suggested size
`/boot` or `/efi`	`/dev/sda1`	260 MiB
`/`	`/dev/sda2`	23–32 GiB
`/home`	`/dev/sda3`	Remainder of the device

This disk layout only creates three partitions. The first one used for the bootloader or the UEFI, the second one is the root of the file system, and the last one for the users home directories.

Example 2:

Mount point	Partition	Suggested size
`/`	`/dev/sda1`	9-16 GiB
`/home`	`/dev/sda2`	> 50 GiB
`[SWAP]`	`/dev/sda3`	Depends on RAM
`/usr`	`/dev/sda4`	25-32 GiB
`/var`	`/dev/sda5`	25 GiB

Example 3:

Mount point	Partition	Suggested size
`/boot`	`/dev/sda1`	260 MiB
`/`	`/dev/sda2`	23–32 GiB
`[SWAP]`	`/dev/sda3`	More than 512 MiB
`/home`	`/dev/sda4`	32-50 GiB
`/data`	`/dev/sda5`	Remainder of the device

While there are many layout that represents how a disk should partitioned for a Linux system, I decided just to go with the simplest one: a single partition for the whole file system. It is not worth the overhead of having more than one.

The minimal required size as stated in the LFS book is 6 GB, and 20 GB is considered as reasonable to “allow for growth”. However, the partition I created is 10 GB in size. It must give enough space to work comfortably with a little to waste. The partition has been formatted with the `ext4` filesystem.

It is also to mention that no swap partition will be created as it won’t make a lot of difference.

3. Setting up a clean environment

The environment of a user or a running process is the set of variable that are defined in the shell it runs in. They are used to affect the behavior of the said processes.

To avoid that any predefined environment variable causes any disturbance for build process which would be advantageous for the unexperienced readers, the LFS book chooses to create a new, empty environment by creating a new user, called `lfs`, and customizing its profile.

A new user however is not the only solution for this. We can simply run a new shell in an empty environment with the help of the `env` command:

```
$ env -i HOME=$LFS TERM=$TERM /bin/bash;
```

`$LFS` is a important variable that is required to set throughout all procedure during the building. It point to the mountpoint the LFS partition. So I need to set it first before using it:

```
$ export LFS=/home/workspace/LFS # mointpoint on my computer
$ env -i HOME=$LFS TERM=$TERM /bin/bash;
```

This new environment now will have only two variables set: `$HOME` and `$TERM` which are fundamental and do not have any side effects that we should worry about.

However, `bash` still read the initialization files from `/etc/profile`, `/etc/bash*`, `~/.profile` and `~/.bashrc`. As we do not want anything intruding in our clean environment and do not know what those initialization files do exactly, it is better just to tell bash not to read them. We can achieve that by passing to it the `--noprofile` and `--norc`. Or better yet, we can give it an initialization file to use instead. This way we can also has the ability to customize this environment as we want.

Bash allows specifying initialization file in a beautiful way: if a script has the argument `--init-file` appended to its *shebang*, it will be interpreted and at the end of it, a new interactive shell will start, turning it into an initialization file!

Note:
The *shebang* is the first line a script. It looks like this: `#!/bin/sh` and it is used to tells the shell with what interpreter it should run the script.
Please note that this not the same mechanism as the one that will be mentioned in glibc-2.29 chapter.

We can utilize this feature to simplify creating and empty environment and running a shell in it without the default initialization files. Here is what the customized initialization file will look like:

```
1  #!/bin/bash --init-file
2
3  set +h
4  umask 022
5
6  LFS=$HOME
7  LC_ALL=POSIX
8  LFS_TGT=$(uname -m)-lfs-linux-gnu
9  PATH=/tools/bin:/bin:/usr/bin
10 PS1='\n\[\e[33m\]\u:\w\$\[\e[0m\] '
11 export LFS LC_ALL LFS_TGT PATH PS1
12
13 alias ls='ls –color'
```

We will save this file to `$LFS/lfshell.sh`, give it execution permission, and call it with the `env` command instead of `bash`:

```
$ env -i HOME=$LFS TERM=$TERM $LFS/lfshell.sh;
```

So before we start, or resume working on this project, we need to set the LFS variable, mount the LFS partition, and start the lfshell. It is fairly a quite tedious thing to do each time. So I might as well create a script will do this for us with a single command.

Here's the script and the explanation of what it does below:

```
1  #!/bin/sh
2
3  export LFS=/home/workspace/LFS
4
5  mountpoint=$( mount | grep sda6 | sed 's/.*on \(.*\) type.*/\1/' )
6
7  if [[ $mountpoint = $LFS ]]; then
8       echo -n "LFS already mounted. ";
9
10 else
11       sudo mount /dev/sda6 $LFS;
12       if [[ $? -ne "0" ]]; then
13            echo "could not mount LFS partition.";
14            exit 1;
15       fi
16 fi
17
18 echo "Starting env";
19 env -i HOME=$LFS TERM=$TERM $LFS/lfshell.sh;
20
```

I saved this script to `.local/bin/lfs.sh`, so now I just have to type `lfs.sh` to:

1. Set `$LFS`. (line 3)

2. Check whether the LFS partition (`/dev/sda6`) is mounted (line 5)
   2.1. if it is already mounted (line 7), it will just inform about it (line 8)
   2.2. if not (line 10), it tries to mount it at `$LFS` (line 11) and report (line 13) if any error occurred (line 12) before it exits the script with a negative return code (line 14).
3. Finally, runs the command I showed before to start the customized environment (line 19).

4. Downloading the sources

Now that we have our environment set up and automated, the final thing we have to do it to download the packages' sources. The list of the urls we have to fetch is attached in the appendix. I'll use it with the `wget` utility to download them.

```
$ lfs.sh
Starting env

:/home$ mkdir /sources
:/home$ wget --input-file=wget-list.txt
```

And that would be all we need to get to the real work.

IV. Building a clean toolchain

This is a temporary toolchain (compiler, assembler, linker, libraries, and a few useful utilities), that will only ensure that the system that we'll build will be totally unaffected by the host system. It will be used to rebuild, again for the second time, a new final toolchain that is "pure" for sure.

This temporary toolchain can be deleted after, so it is better that we separate it from the rest of the system. So, it will be installed under a directory `/tools` in the root of the filesystem. For convenience and this directory will be symlinked on the host system to allow a seamless workflow.

The first package will compile is `binutils-2.32`. It contains a linker, an assembler, and other tools for handling object files.

The time needed to compile this package will be used a reference for how long would it take to compile other packages. It will be referred to as SBU (Standard Build Unit).

The procedure to compile other packages is not very different from this one. So I will only explain it once.

1. The building

`binutils-2.32`

First, we go into the `/sources` directory and extract the tar archive of the source files of `binutils`:

```
$ cd /sources
$ tar -xvf binutils-2.32.tar.gz
$ cd  binutils-2.32
```

Then we create a directory to build in:

```
$ mkdir build && cd build
```

We run the configure script which adjust the source code to the current system.

```
$ ../configure --prefix=/tools            \
               --with-sysroot=$LFS        \
               --with-lib-path=/tools/lib \
               --target=$LFS_TGT          \
               --disable-nls              \
               --disable-werror
```

Note:
The `\` at the end of line allows to expand the command to more than one line.

The LFS book explains the meaning of the arguments:

`--prefix=/tools`

This tells the configure script to prepare to install the Binutils programs in the `/tools` directory.

`--with-sysroot=$LFS`

For cross compilation, this tells the build system to look in `$LFS` for the target system libraries as needed.

`--with-lib-path=/tools/lib`

This specifies which library path the linker should be configured to use.

`--target=$LFS_TGT`

Because the machine description in the `LFS_TGT` variable is slightly different than the value returned by the config.guess script, this switch will tell the configure script to adjust Binutil's build system for building a cross linker.

`--disable-nls`

This disables internationalization as i18n is not needed for the temporary tools.

`--disable-werror`

This prevents the build from stopping in the event that there are warnings from the host's compiler.

Then, we run the `make` command which calls the compiler on the appropriate source files. We will also use the `time` command to measure how long it will take so we can have a idea about the time needed to compile the other packages. (LFS book provides the time needed in SBU units)

```
$ time make
make[1]: Entering directory '/sources/binutils-2.32/build'
mkdir -p -- ./libiberty
Configuring in ./libiberty
configure: creating cache ./config.cache

***omitted output***

make[2]: Leaving directory '/sources/binutils-2.32/build/ld'
make[1]: Nothing to be done for 'all-target'.
make[1]: Leaving directory '/sources/binutils-2.32/build'
```

```
real    3m3.006s
user    2m31.549s
sys     0m30.465s
```

As you can see, the time it took was 3 minutes and 3 seconds. So we can expect that the slowest package, which is `gcc-8.2.0`, which take about 92 SBU would build in approximately 92 * 3.5 = 325.5 minutes, which is about **five hours and half!**

Now we install the compiled binaries and libraries:

```
$ make install
```

The following are what was just installed. For a brief description of what they are or what they do, refer to either the LFS book in the part discussing the package or see its manual page.

```
/tools/bin/addr2line
/tools/bin/ar
/tools/bin/as
/tools/bin/c++filt
/tools/bin/elfedit
/tools/bin/gprof
/tools/bin/ld
/tools/bin/ld.bfd
/tools/bin/ld.gold
/tools/bin/nm
/tools/bin/objcopy
/tools/bin/objdump
/tools/bin/ranlib
/tools/bin/readelf
/tools/bin/size
/tools/bin/strings
/tools/bin/strip
/tools/lib/libbfd.a
/tools/lib/libbfd.so
/tools/lib/libopcodes.a
/tools/lib/libopcodes.so
```

The next package we will build next is `gcc-8.2.0`

gcc-8.2.0

The GCC package contains the GNU compiler collection, which includes the C and C++ compilers.

GCC requires the GMP, MPFR and MPC packages. We just need to unpack each of packages we had already downloaded into the GCC source directory and rename the resulting directories so the GCC build procedures will automatically use them.

```
tar -xf ../mpfr-4.0.2.tar.xz
mv -v mpfr-4.0.2 mpfr
```

And we do the same the other two packages. After that procedure is not so different, just `configure && make && make install`. Although some modifications are made on the LFS book, they are nothing so meaningful.

Here are the other packages that we will compile and install under `/tools`:

glibc-2.29

The Glibc package contains the main C library. This library provides the basic routines for allocating memory, searching directories, opening and closing files, reading and writing files, string handling, pattern matching, arithmetic, and so on.

An enormous problem was encountered after building and installing this package, although the operation was made successfully and without any errors.

The problem was that no newly compiled binary was able to run in the `/tool` temporary environment now! Every time I try to execute it I get the same error, this is error raises when trying to run a newly compiled dummy binary:

```
bash: ./a.out: No such file or directory
```

And this now does not make sense because I can clearly see the `./a.out` in the file tree.

The next thing I did was to check the type of `./a.out` using the `file` command:

```
$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, with
debug_info, not stripped
```

So I am sure now the file is an executable, so why it might not be running in this new environment? Since the compiler works correctly and produces correct binaries (proof is that the `file` utility identified `a.out` correctly), the problem is definitely with interpreter, which is `/lib64/ld-linux-x86-64.so.2` as shown above.

Upon checking on the interpreter, the problem seem to solved: The interpreter from glibc is installed under `/tools/lib`, while binaries or x86_64 architectures are built to find it in `/lib64/ld-linux-x86-64.so.2`. This also explains why the majority (if not all) of Linux distributions have a symbolic link in `/lib64` referring to either `/lib` or `/usr/lib` !

Everything about this issue is clear now, when the bash shell receives the command to run the dummy binary a.out, it scans it to know where to look for a suitable interpreter. When it goes to load the interpreter, it can not find it in the expected path so that is what “No such file or directory” means.

The solution, which won’t be needed if I followed the book to the letter, is to create the symbolic link `/lib64`, which is instructed whiling the fist package `binutils`:

On x86_64 architectures, libraries are often looked up in the directory `/lib64` as well. We will symlink it to `/tools/lib`

```
$ ln -sv /tools/lib /tools/lib64
```

Now everything works fine, we run the dummy binary again and the check that the return code is 0:

```
$ ./a.out && echo $?
0
```

tcl-8.6.9

The Tcl package contains the Tool Command Language.

expect-5.45.4

This package contains a program for carrying out scripted dialogues with other interactive programs. It is commonly used for testing other packages. It is only installed in the temporary toolchain.

dejagnu-1.6.2

This package contains a framework for testing other programs. It is only installed in the temporary toolchain.

m4-1.4.18

This package contains a general text macro processor useful as a build tool for other programs.

ncurses-6.1

This package contains libraries for terminal-independent handling of character screens. It is often used to provide cursor control for a menuing system. It is needed by a number of packages in LFS.

bash-5.0

Bash is the famous shell and scripting language. This package satisfies an LSB core requirement to provide a Bourne Shell interface to the system. It was chosen over other shell packages because of its common usage and extensive capabilities beyond basic shell functions.

bison-3.3.2

This package contains the GNU version of yacc (Yet Another Compiler Compiler) needed to build several other LFS programs.

bzip2-1.0.6

This package contains programs for compressing and decompressing files. It is required to decompress many LFS packages.

coreutils-8.30

This package contains a number of essential programs for viewing and manipulating files and directories. These programs are needed for command line file management, and are necessary for the installation procedures of every package in LFS.

It is one hell of an interesting package, it is the one that provides the `ls`, `mv`, `rm`, `cp`, ect.. commands

diffutils-3.7

This package contains programs that show the differences between files or directories. These programs can be used to create patches, and are also used in many packages' build procedures.

file-5.36

This package contains a ***magic*** utility for determining the type of a given file or files. A few packages need it to build.

findutils-4.6.0

This package contains programs to find files in a file system. It is used in many packages' build scripts.

gawk-4.2.1

This package contains programs for manipulating text files. It is the GNU version of awk (Aho-Weinberg-Kernighan). It is used in many other packages' build scripts

gettext-0.19.8.1

This package contains utilities and libraries for internationalization and localization of numerous packages.

grep-3.3

This package contains programs for searching through files. These programs are used by most packages' build scripts.

gzip-1.10

This package contains programs for compressing and decompressing files. It is needed to decompress many packages in LFS and beyond.

make-4.2.1

This package contains a program for directing the building of packages. It is required by almost every package in LFS.

patch-2.7.6

This package contains a program for modifying or creating files by applying a patch file typically created by the diff program. It is needed by the build procedure for several LFS packages.

perl-5.28.1

This package is an interpreter for the runtime language PERL. It is needed for the installation and test suites of several LFS packages.

The old and ugly Perl programming language!

python-3.7.2

This package provides an interpreted language that has a design philosophy that emphasizes code readability.

Oh the lovely Python programming language!

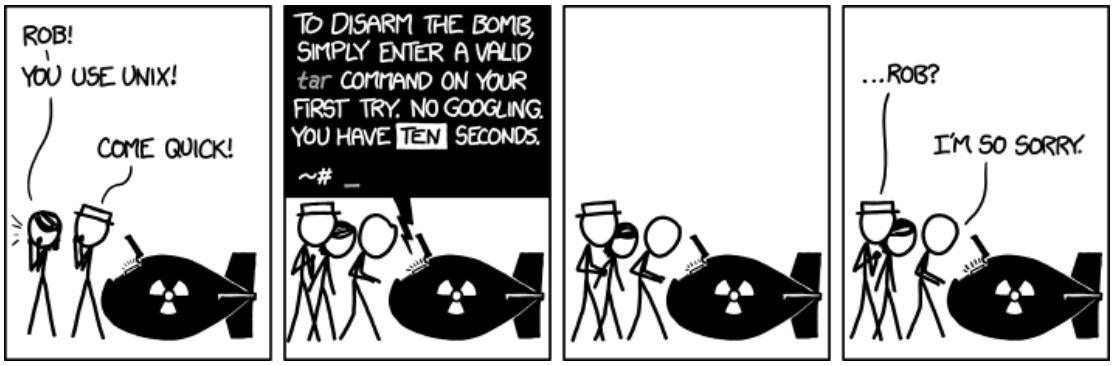
sed-4.7

This package allows editing of text without opening it in a text editor. It is also needed by most LFS packages' configure scripts.

The Swiss army knife (or as I lately used to call it: The GNU army knife.)

`tar-1.31`

The Tar package contains an archiving program.



Comic strip #1168 from xkcd by Randall Munroe.

Alt-text: "I don't know what's worse--the fact that after 15 years of using tar I still can't keep the flags straight, or that after 15 years of technological advancement I'm still mucking with tar flags that were 15 years old when I started."

The comic is not very accurate anymore since the latest versions have been simplified so you do not have to use a lot of flag to extract a `tar.gz` archive. But try and older one and tell me!

`texinfo-6.5`

The Texinfo package contains programs for reading, writing, and converting info pages.

`xz-5.2.4`

The Xz package contains programs for compressing and decompressing files. It provides capabilities for the lzma and the newer xz compression formats. Compressing text files with xz yields a better compression percentage than with the traditional `gzip` or `bzip2` commands.

2. Wrap-up

Now we have a known-good set of tools that are isolated from and unaffected by the host system. By using `chroot`, the commands in the remaining chapters will be contained within that environment, ensuring a clean, trouble-free build of the target LFS system, the **SunS OS**. After the constructing our system, we can then delete these temporary tools as the new system will have its own.

“If you can build and install Glibc,
you can build and install the rest too.”
— LFS book

V. Building the SunS OS

1. Creating the system backbones

The most substantial thing that can describe an operating system is that, it is a hierarchy of files (at least in the Unix world) on a disk. So now is the time for the filesystem hierarchy (or tree) of the SunS OS to start showing up. We will create a structure of directories in the LFS partition which complies with the FSH standard as defined by the Linux Foundation.

Please refer to the LFS book for the commands used to create the directory tree. Here is a part of the tree that was created:

```
/bin
/boot
/etc/sysconfig
/home
/mnt
/opt
/media
/sbin
/srv
```

```
/var
/root
/tmp
/usr/bin
/usr/include
/usr/share
/var/log
/var/mail
/run
```

We also create the directories to mount the kernel virtual filesytems, such as `proc`, `sys`, and `dev`. We create special devices nodes too, like `/dev/console`, `/dev/null`. Then we mount the filesystems exposed on the host to these directories we just created:

```
mount -vt devpts devpts $LFS/dev/pts -o gid=5,mode=620
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run
```

Now we are ready to enter (via `chroot`) the new system. It is technically the same as booting into the new system except with a kernel from the host.

2. Automating the boring stuff

The work in this chapter, as in the previous one, won’t be done is one sitting as it does not take a little time. So I modified the script that we used in the previous section and adjusted it to the current

situation and to set up that is necessary, i.e. mounting the root partition, populating `/dev`, `/proc`, etc.. and starting a clean shell, all while preserving the functions it used to perform before.

This is the new resulting script:

```
1  #!/bin/sh
2
3  export LFS=/home/ader/workspace/LFS
4
5  mountpoint=$( mount | grep sda6 | sed 's/.*on \(.*\) type.*/\1/' )
6
7  if [[ $mountpoint = $LFS ]]; then
8       echo -n "LFS already mounted. ";
9
10 else
11       sudo mount /dev/sda6 $LFS;
12       if [[ $? -ne "0" ]]; then
13             echo "could not mount LFS partition.";
14             exit 1;
15       fi
16 fi
17
18
19 if [[ -z "$1" ]]; then
20
21       echo "Starting env";
22       env -i HOME=$LFS TERM=$TERM $LFS/lfshell.sh;
23
24 elif [[ "$1" = "chroot" ]]; then
25
26       echo "Chrooting";
27
28       mountpoint -q $LFS/dev || sudo mount -v --bind /dev $LFS/dev;
29       mountpoint -q $LFS/sys || sudo mount -vt sysfs sysfs $LFS/sys;
30       mountpoint -q $LFS/run || sudo mount -vt tmpfs tmpfs $LFS/run;
31       mountpoint -q $LFS/proc || sudo mount -vt proc proc $LFS/proc;
32       mountpoint -q $LFS/dev/pts || sudo mount -vt \
33             devpts devpts $LFS/dev/pts -o gid=5,mode=620;
34
35       sudo chroot "$LFS" \
36             /usr/bin/env -i HOME=/root TERM=$TERM /lfshell.sh;
37
38 fi
```

Part of it (until line 16) remained the same. After it, the script has two operational modes:

- the first (from line 19) correspond with invoking the script without any argument (just `lfs.sh`); this is its old normal behavior.
- And the second mode (from line 24) is when the script is invoked with the `chroot` argument (`lfs.sh chroot`).

In the latter mode, it perform the preparation we did in this chapter, i.e. mount the main partition and the kernel virtual filesystems, and then `chroot` (line 35) to SunS partition and start a new environment there (it is not necessary anymore).

In lines 28-32, we check whether those filesystems have been mounted before or not before we proceed with mounting them again. The reason for this is that if we just keep mounting devices every time and ignore the previously mounted, we might run out `ptss` which will malfunction the host system.

3. Compiling packages

In this section, we will compile all the package that constitute the SunS OS. The procedure is the same as we did in the chapter Building a clean toolchain: `configure && make && make install`. In Addition to the packages used in that chapter, these packages will be compiled too:

Note:
this list is unordered and some package may depend on another to be installe before it builds successfully. So the order in which the packages are compiled in the LFS book must respected.

acl-2.2.53

attr-2.4.48

autoconf-2.69

automake-1.16.1

bc-1.07.1

check-0.12.0

e2fsprogs-1.44.5

elfutils-0.176

eudev-3.2.7

expat-2.2.6

flex-2.6.4

gdbm-1.18.1

gmp-6.1.2

gperf-3.1

groff-1.22.4

grub-2.02

iana-etc-2.30

inetutils-1.9.4

intltool-0.51.0

iproute2-4.20.0

kbd-2.0.4

kmod-26

less-530

libcap-2.26

libffi-3.2.1

libpipeline-1.5.1

libtool-2.4.6

man-db-2.8.5

man-pages-4.16

meson-0.49.2

mpc-1.1.0

mpfr-4.0.2

ninja-1.9.0

openssl-1.1.1a

pkg-config-0.29.2

procps-ng-3.3.15

psmisc-23.2

readline-8.0

shadow-4.6

sysklogd-1.5.1

sysvinit-2.93

util-linux-2.33.1

vim-8.1

xml::parser-2.44

xz-5.2.4

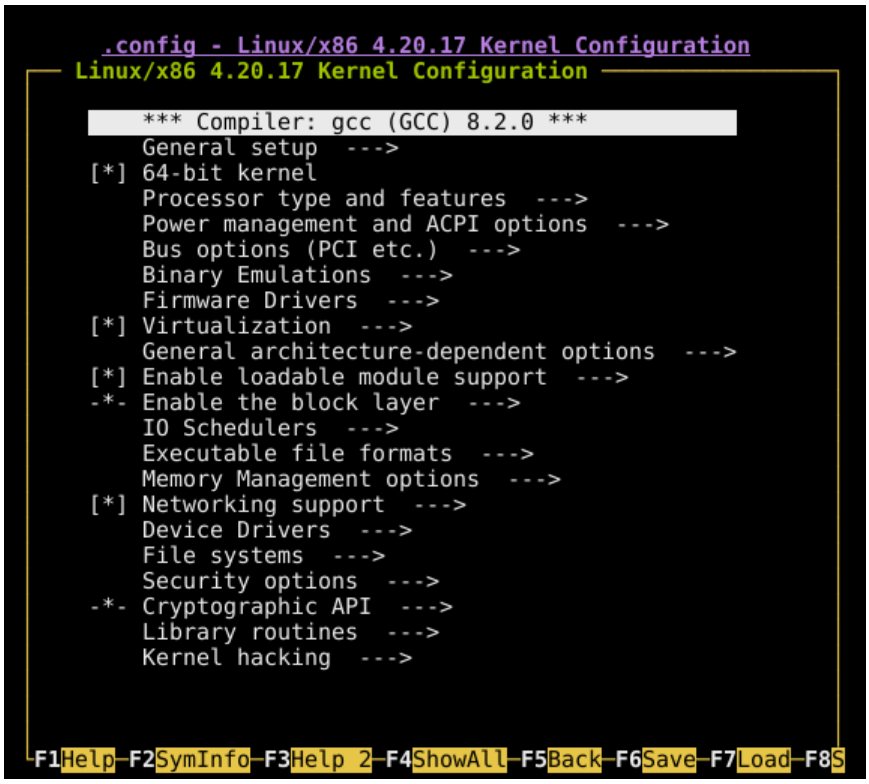
zlib-1.2.11

VI. building The Kernel

The most interesting package of all is the Linux kernel package. For the SunS OS, I will be using the version 4.20.12 of kernel.

The Linux kernel has a slightly different compiling process. A simple configure script would be enough to tweak the uncountable number of functionalities the kernel has. Instead, there is a specific tools, which can be used in many ways, that set the necessary parameter to build and customize the kernel.

The `menuconfig` (accessible via `make menuconfig`) is a menu-driven interface to toggle the statue of activation of modules and settings of the kernel. Some of those modules can be enabled to be build internally with the kernel and other have an option to enable them as loadable modules which can be loaded and used on demand.



screenshot of the `nconfig` configuration utility for the Linux kernel

The other available way to set the configuration of the Linux kernel is via the `nconfig` utility (which is also accessible via `make nconfig`). It is offers the same functionalities as `menuconfig` and it just looks a little newer and neater.

1. Notable kernel features

Some of the kernel features that interested me the most as a student in the TRI branch are the following:

- Cryptographic API:
  - RSA algorithm
  - Diffie-Hellman algorithm
  - HMAC support
  - MD5 digest algorithm
  - SHA1 digest algorithm
  - AES cipher
  - Blowfish cipher
  - Pseudo Random Number Generation for Cryptographic
- Security options
  - NSA SELinux Support
  - AppArmor support
- File systems
  - Second extended fs support
  - The Extended 3 (ext3) filesystem
  - The Extended 4 (ext4) filesystem
  - Quota support
  - Network File Systems
    - NFS client support
    - SMB3 and CIFS support
  - DOS/FAT/NT Filesystems

    - NTFS file system support
    - NTFS write support
- Virtualization
- General setup
    - Default hostname (has been set to **SunS**)

After setting the desired options, configuration has been saved and output to a `.config` file, which the make process will read later and build a kernel based on that configuration. The `.config` file is attached to this document to consult.

2. Compiling the kernel

Now we just have to run the `make` command to compile the kernel. It only takes about 20 minutes, it is not a large as one might expect. `glibc` is in fact heavier than the kernel!

We run the command `make modules_install` to copy the enabled kernel to `/lib/modules`.

We will find the newly build kernel in `arch/x86_64/boot/bzImage`.

3. Making the system bootable

The `GRUB` bootloader has been compiled and installed in the previous chapter. We need now to configure it load the new kernel.

As for me, I already have grub installed on `/boot` partition. I need just to add to it a new entry, for the SunS OS.

We put first the kernel image in `/boot` partition as it is only partition available during boot and before a system starts.

```
$ mkdir -v /boot/SunS
$ cp -v /sources/linux-4.20.12/arch/arch/x86_64/boot/bzImage
/boot/SunS/vmlinuz-4.20.12-lfs-8.4
```

We add the following to grub configuration file in /boot/grub/grub.cfg:

```
cat >> /boot/grub/grub.cfg << "EOF"
menuentry "SunS GNU/Linux, Linux 4.20.12-lfs-8.4" {
        set root=(hd0,6)
        linux   /boot/vmlinuz-4.20.12-lfs-8.4 root=/dev/sda6 ro
}
```

```
EOF
```

We reboot now a choose the SunS menu entry from GRUB menu.

VII. Finishing things up

A lot of additional configurations have not be discussed or even mentioned in the document. I did not talk about choosing the suitable firmware and drivers for the kernel, did not talk about the `/etc` which is not any less important, especially the file `/etc/passwd` and `/etc/shadow` without which you can’t login if they were not valid. The locale, character sets and timezone which are a very painful subject for any developer also were not discussed. But the necessary steps to ensure the good working the SunS OS have been taken practically. In the end, the LFS book talks about all these with details for any needed reference.

VIII. What’s Next?

Although the SunS OS is booting and marvelously operational, it is not quite the system that anyone would be attracted to (except for it creator, right?). A few things might need to considered to add if there was any future release of SunS OS are:

- A graphical user interface with a fancy desktop and windows
- An organized package manger
- Other hardware support
- A decent independent boot process.

IX. Appendices

1. Scripts

Host test

Tests the availability of the host system utilities.

```
#!/bin/bash
# Simple script to list version numbers of critical development tools
export LC_ALL=C
bash --version | head -n1 | cut -d" " -f2-4
MYSH=$(readlink -f /bin/sh)
echo "/bin/sh -> $MYSH"
echo $MYSH | grep -q bash || echo "ERROR: /bin/sh does not point to bash"
unset MYSH

echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-
bison --version | head -n1

if [ -h /usr/bin/yacc ]; then
  echo "/usr/bin/yacc -> `readlink -f /usr/bin/yacc`";
elif [ -x /usr/bin/yacc ]; then
  echo yacc is `/usr/bin/yacc --version | head -n1`
else
  echo "yacc not found"
fi

bzip2 --version 2>&1 < /dev/null | head -n1 | cut -d" " -f1,6-
echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
diff --version | head -n1
find --version | head -n1
gawk --version | head -n1

if [ -h /usr/bin/awk ]; then
  echo "/usr/bin/awk -> `readlink -f /usr/bin/awk`";
elif [ -x /usr/bin/awk ]; then
  echo awk is `/usr/bin/awk --version | head -n1`
else
  echo "awk not found"
fi

gcc --version | head -n1
g++ --version | head -n1
ldd --version | head -n1 | cut -d" " -f2-  # glibc version
grep --version | head -n1
gzip --version | head -n1
cat /proc/version
m4 --version | head -n1
make --version | head -n1
```

```
43 patch --version | head -n1
44 echo Perl `perl -V:version`
45 python3 --version
46 sed --version | head -n1
47 tar --version | head -n1
48 makeinfo --version | head -n1  # texinfo version
49 xz --version | head -n1
50
51 echo 'int main(){}' > dummy.c && g++ -o dummy dummy.c
52 if [ -x dummy ]
53   then echo "g++ compilation OK";
54   else echo "g++ compilation failed"; fi
55 rm -f dummy.c dummy
```

Sources URLs

These are the URL from where I downloaded the packages' sources.

```
http://download.savannah.gnu.org/releases/acl/acl-2.2.53.tar.gz
http://download.savannah.gnu.org/releases/attr/attr-2.4.48.tar.gz
http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.xz
http://ftp.gnu.org/gnu/automake/automake-1.16.1.tar.xz
http://ftp.gnu.org/gnu/bash/bash-5.0.tar.gz
http://ftp.gnu.org/gnu/bc/bc-1.07.1.tar.gz
http://ftp.gnu.org/gnu/binutils/binutils-2.32.tar.xz
http://ftp.gnu.org/gnu/bison/bison-3.3.2.tar.xz
http://anduin.linuxfromscratch.org/LFS/bzip2-1.0.6.tar.gz
https://github.com/libcheck/check/releases/download/0.12.0/check-
0.12.0.tar.gz
http://ftp.gnu.org/gnu/coreutils/coreutils-8.30.tar.xz
https://dbus.freedesktop.org/releases/dbus/dbus-1.12.12.tar.gz
http://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.2.tar.gz
http://ftp.gnu.org/gnu/diffutils/diffutils-3.7.tar.xz
https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.44.5/
e2fsprogs-1.44.5.tar.gz
https://sourceware.org/ftp/elfutils/0.176/elfutils-0.176.tar.bz2
https://dev.gentoo.org/~blueness/eudev/eudev-3.2.7.tar.gz
https://prdownloads.sourceforge.net/expat/expat-2.2.6.tar.bz2
https://prdownloads.sourceforge.net/expect/expect5.45.4.tar.gz
ftp://ftp.astron.com/pub/file/file-5.36.tar.gz
http://ftp.gnu.org/gnu/findutils/findutils-4.6.0.tar.gz
https://github.com/westes/flex/releases/download/v2.6.4/flex-2.6.4.tar.gz
http://ftp.gnu.org/gnu/gawk/gawk-4.2.1.tar.xz
http://ftp.gnu.org/gnu/gcc/gcc-8.2.0/gcc-8.2.0.tar.xz
http://ftp.gnu.org/gnu/gdbm/gdbm-1.18.1.tar.gz
http://ftp.gnu.org/gnu/gettext/gettext-0.19.8.1.tar.xz
http://ftp.gnu.org/gnu/glibc/glibc-2.29.tar.xz
http://ftp.gnu.org/gnu/gmp/gmp-6.1.2.tar.xz
http://ftp.gnu.org/gnu/gperf/gperf-3.1.tar.gz
http://ftp.gnu.org/gnu/grep/grep-3.3.tar.xz
http://ftp.gnu.org/gnu/groff/groff-1.22.4.tar.gz
https://ftp.gnu.org/gnu/grub/grub-2.02.tar.xz
```

```
http://ftp.gnu.org/gnu/gzip/gzip-1.10.tar.xz
http://anduin.linuxfromscratch.org/LFS/iana-etc-2.30.tar.bz2
http://ftp.gnu.org/gnu/inetutils/inetutils-1.9.4.tar.xz
https://launchpad.net/intltool/trunk/0.51.0/+download/intltool-0.51.0.tar.gz
https://www.kernel.org/pub/linux/utils/net/iproute2/iproute2-4.20.0.tar.xz
https://www.kernel.org/pub/linux/utils/kbd/kbd-2.0.4.tar.xz
https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-26.tar.xz
http://www.greenwoodsoftware.com/less/less-530.tar.gz
http://www.linuxfromscratch.org/lfs/downloads/8.4/lfs-bootscripts-
20180820.tar.bz2
https://www.kernel.org/pub/linux/libs/security/linux-privs/libcap2/libcap-
2.26.tar.xz
ftp://sourceware.org/pub/libffi/libffi-3.2.1.tar.gz
http://download.savannah.gnu.org/releases/libpipeline/libpipeline-
1.5.1.tar.gz
http://ftp.gnu.org/gnu/libtool/libtool-2.4.6.tar.xz
https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.20.17.tar.xz
http://ftp.gnu.org/gnu/m4/m4-1.4.18.tar.xz
http://ftp.gnu.org/gnu/make/make-4.2.1.tar.bz2
http://download.savannah.gnu.org/releases/man-db/man-db-2.8.5.tar.xz
https://www.kernel.org/pub/linux/docs/man-pages/man-pages-4.16.tar.xz
https://github.com/mesonbuild/meson/releases/download/0.49.2/meson-
0.49.2.tar.gz
https://ftp.gnu.org/gnu/mpc/mpc-1.1.0.tar.gz
http://www.mpfr.org/mpfr-4.0.2/mpfr-4.0.2.tar.xz
https://github.com/ninja-build/ninja/archive/v1.9.0/ninja-1.9.0.tar.gz
http://ftp.gnu.org/gnu/ncurses/ncurses-6.1.tar.gz
https://openssl.org/source/openssl-1.1.1a.tar.gz
http://ftp.gnu.org/gnu/patch/patch-2.7.6.tar.xz
https://www.cpan.org/src/5.0/perl-5.28.1.tar.xz
https://pkg-config.freedesktop.org/releases/pkg-config-0.29.2.tar.gz
https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-
3.3.15.tar.xz
https://sourceforge.net/projects/psmisc/files/psmisc/psmisc-23.2.tar.xz
https://www.python.org/ftp/python/3.7.2/Python-3.7.2.tar.xz
https://docs.python.org/ftp/python/doc/3.7.2/python-3.7.2-docs-html.tar.bz2
http://ftp.gnu.org/gnu/readline/readline-8.0.tar.gz
http://ftp.gnu.org/gnu/sed/sed-4.7.tar.xz
https://github.com/shadow-maint/shadow/releases/download/4.6/shadow-
4.6.tar.xz
http://www.infodrom.org/projects/sysklogd/download/sysklogd-1.5.1.tar.gz
https://github.com/systemd/systemd/archive/v240/systemd-240.tar.gz
http://anduin.linuxfromscratch.org/LFS/systemd-man-pages-240.tar.xz
http://download.savannah.gnu.org/releases/sysvinit/sysvinit-2.93.tar.xz
http://ftp.gnu.org/gnu/tar/tar-1.31.tar.xz
https://downloads.sourceforge.net/tcl/tcl8.6.9-src.tar.gz
http://ftp.gnu.org/gnu/texinfo/texinfo-6.5.tar.xz
https://www.iana.org/time-zones/repository/releases/tzdata2018i.tar.gz
http://anduin.linuxfromscratch.org/LFS/udev-lfs-20171102.tar.bz2
https://www.kernel.org/pub/linux/utils/util-linux/v2.33/util-linux-
2.33.1.tar.xz
ftp://ftp.vim.org/pub/vim/unix/vim-8.1.tar.bz2
https://cpan.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.44.tar.gz
```

```
https://tukaani.org/xz/xz-5.2.4.tar.xz
https://zlib.net/zlib-1.2.11.tar.xz
http://www.linuxfromscratch.org/patches/lfs/8.4/bzip2-1.0.6-install_docs-
1.patch
http://www.linuxfromscratch.org/patches/lfs/8.4/coreutils-8.30-i18n-1.patch
http://www.linuxfromscratch.org/patches/lfs/8.4/glibc-2.29-fhs-1.patch
http://www.linuxfromscratch.org/patches/lfs/8.4/kbd-2.0.4-backspace-1.patch
http://www.linuxfromscratch.org/patches/lfs/8.4/sysvinit-2.93-consolidated-
1.patch
http://www.linuxfromscratch.org/patches/lfs/8.4/systemd-240-security_fixes-
2.patch
```

Sources md5 checksums

This file is used to check the integrity of the downloaded files.

```
007aabf1dbb550bcddde52a244cd1070  acl-2.2.53.tar.gz
bc1e5cb5c96d99b24886f1f527d3bb3d  attr-2.4.48.tar.gz
50f97f4159805e374639a73e2636f22e  autoconf-2.69.tar.xz
53f38e7591fa57c3d2cee682be668e5b  automake-1.16.1.tar.xz
2b44b47b905be16f45709648f671820b  bash-5.0.tar.gz
cda93857418655ea43590736fc3ca9fc  bc-1.07.1.tar.gz
0d174cdaf85721c5723bf52355be41e6  binutils-2.32.tar.xz
c9b552dee234b2f6b66e56b27e5234c9  bison-3.3.2.tar.xz
00b516f4704d4a7cb50a1d97e6e8e15b  bzip2-1.0.6.tar.gz
31b17c6075820a4341195929411 86f70  check-0.12.0.tar.gz
ab06d68949758971fe744db66b572816  coreutils-8.30.tar.xz
e1b07516533f351b3aba3423fafeffd6  dejagnu-1.6.2.tar.gz
4824adc0e95dbbf11dfbdfaad6a1e461  diffutils-3.7.tar.xz
8d78b11d04d26c0b2dd149529441fa80  e2fsprogs-1.44.5.tar.gz
077e4f49320cad82bf17a997068b1db9  elfutils-0.176.tar.bz2
c75d99910c1791dd9430d26ab76059c0  eudev-3.2.7.tar.gz
ca047ae951b40020ac831c28859161b2  expat-2.2.6.tar.bz2
00fce8de158422f5ccd2666512329bd2  expect5.45.4.tar.gz
9af0eb3f5db4ae00fffc37f7b861575c  file-5.36.tar.gz
9936aa8009438ce185bea2694a997fc1  findutils-4.6.0.tar.gz
2882e3179748cc9f9c23ec593d6adc8d  flex-2.6.4.tar.gz
95cf553f50ec9f386b5dfcd67f30180a  gawk-4.2.1.tar.xz
4ab282f414676496483b3e1793d07862  gcc-8.2.0.tar.xz
988dc82182121c7570e0cb8b4fcd5415  gdbm-1.18.1.tar.gz
df3f5690eaa30fd228537b00cb7b7590  gettext-0.19.8.1.tar.xz
e6c279d5b2f0736f740216f152acf974  glibc-2.29.tar.xz
f58fa8001d60c4c77595fbbb62b63c1d  gmp-6.1.2.tar.xz
9e251c0a618ad0824b51117d5d9db87e  gperf-3.1.tar.gz
05d0718a1b7cc706a4bdf8115363f1ed  grep-3.3.tar.xz
08fb04335e2f5e73f23ea4c3adbf0c5f  groff-1.22.4.tar.gz
8a4a2a95aac551fb0fba860ceabfa1d3  grub-2.02.tar.xz
691b1221694c3394f1c537df4eee39d3  gzip-1.10.tar.xz
3ba3afb1d1b261383d247f46cb135ee8  iana-etc-2.30.tar.bz2
87fef1fa3f603aef11c41dcc097af75e  inetutils-1.9.4.tar.xz
12e517cac2b57a0121cda351570f1e63  intltool-0.51.0.tar.gz
```

```
f3dab4c812812bbb5873cb90f471bcbf  iproute2-4.20.0.tar.xz
c1635a5a83b63aca7f97a3eab39ebaa6  kbd-2.0.4.tar.xz
1129c243199bdd7db01b55a61aa19601  kmod-26.tar.xz
6a39bccf420c946b0fd7ffc64961315b  less-530.tar.gz
e08811a18356eeef524b2ed333e8cb86  lfs-bootscripts-20180820.tar.bz2
968ac4d42a1a71754313527be2ab5df3  libcap-2.26.tar.xz
83b89587607e3eb65c70d361f13bab43  libffi-3.2.1.tar.gz
4c8fe6cd85422baafd6e060f896c61bc  libpipeline-1.5.1.tar.gz
1bfb9b923f2c1339b4d2ce1807064aa5  libtool-2.4.6.tar.xz
0be56db8e91d057cf6d45f35cb63b2b8  linux-4.20.17.tar.xz
730bb15d96fffe47e148d1e09235af82  m4-1.4.18.tar.xz
15b012617e7c44c0ed482721629577ac  make-4.2.1.tar.bz2
c5c6c3434be14a5527d43b5ad0f09a13  man-db-2.8.5.tar.xz
ad9f1ff81276fe8d90d077484d6d4b5e  man-pages-4.16.tar.xz
0267b0871266056184c484792572c682  meson-0.49.2.tar.gz
4125404e41e482ec68282a2e687f6c73  mpc-1.1.0.tar.gz
320fbc4463d4c8cb1e566929d8adc4f8  mpfr-4.0.2.tar.xz
f340be768a76724b83e6daab69009902  ninja-1.9.0.tar.gz
98c889aaf8d23910d2b92d65be2e737a  ncurses-6.1.tar.gz
963deb2272d6be7d4c2458afd2517b73  openssl-1.1.1a.tar.gz
78ad9937e4caadcba1526ef1853730d5  patch-2.7.6.tar.xz
fbb590c305f2f88578f448581b8cf9c4  perl-5.28.1.tar.xz
f6e931e319531b736fadc017f470e68a  pkg-config-0.29.2.tar.gz
2b0717a7cb474b3d6dfdeedfbad2eccc  procps-ng-3.3.15.tar.xz
0524258861f00be1a02d27d39d8e5e62  psmisc-23.2.tar.xz
df6ec36011808205beda239c72f947cb  Python-3.7.2.tar.xz
107ade7bb17efd104a22b2d457f4cb67  python-3.7.2-docs-html.tar.bz2
7e6c1f16aee3244a69aba6e438295ca3  readline-8.0.tar.gz
777ddfd9d71dd06711fe91f0925e1573  sed-4.7.tar.xz
b491fecbf1232632c32ff8f1437fd60e  shadow-4.6.tar.xz
c70599ab0d037fde724f7210c2c8d7f8  sysklogd-1.5.1.tar.gz
041dbe36a5dd80b2108aff305bc10620  sysvinit-2.93.tar.xz
bc9a89da1185ceb2210de12552c43ce2  tar-1.31.tar.xz
aa0a121d95a0e7b73a036f26028538d4  tcl8.6.9-src.tar.gz
3715197e62e0e07f85860b3d7aab55ed  texinfo-6.5.tar.xz
b3f0a1a789480a036e58466cd0702477  tzdata2018i.tar.gz
d92afb0c6e8e616792068ee4737b0d24  udev-lfs-20171102.tar.bz2
6fcfea2043b5ac188fd3eed56aeb5d90  util-linux-2.33.1.tar.xz
1739a1df312305155285f0cfa6118294  vim-8.1.tar.bz2
af4813fe3952362451201ced6fbce379  XML-Parser-2.44.tar.gz
003e4d0b1b1899fc6e3000b24feddf7c  xz-5.2.4.tar.xz
85adef240c5f370b308da8c938951a68  zlib-1.2.11.tar.xz
6a5ac7e89b791aae556de0f745916f7f  bzip2-1.0.6-install_docs-1.patch
a9404fb575dfd5514f3c8f4120f9ca7d  coreutils-8.30-i18n-1.patch
9a5997c3452909b1769918c759eff8a2  glibc-2.29-fhs-1.patch
f75cca16a38da6caa7d52151f7136895  kbd-2.0.4-backspace-1.patch
aaa84675e717504d7d3da452c8c2eaf1  sysvinit-2.93-consolidated-1.patch
```

2. Resources, References & Standards

- Portable Operating System Interface (POSIX):
  http://get.posixcertified.ieee.org/prodstandards.html
- Linux Standard Base:
  https://wiki.linuxfoundation.org/lsb/start
- Filesystem Hierarchy Standard:
  https://wiki.linuxfoundation.org/lsb/fhs

- The Linux From Scratch Project:
  http://www.linuxfromscratch.org/lfs/
- The Beyond Linux From Scratch Project:
  http://www.linuxfromscratch.org/blfs
- The LFS mailing lists:
  http://www.linuxfromscratch.org/lists
- The LFS IRC channel:
  irc://irc.freenode.net/lfs-support
- The Arch Linux wiki:
  https://wiki.archlinux.org/index.php
- Linux Kernel in a Nutshell:

  http://www.kroah.com/lkn
- The Linux Kernel documentation:
  https://www.kernel.org/doc/html/latest/
- Stack Overflow:
  http://stackoverflow.com/

contact: ko.elidrissi@gmail.com