# Skills Learnt

Omkar Girish Kamath

May 6, 2023

## Contents



# 1 Makefile

## 1.1 Code

```
CC=gcc
SRCS=libll.c libll.h
OBJS=$(SRCS:.c=.o)
CFLAGS=-fPIC
SONAME=libll.so
VERSION=1.0
LIBPATH=/home/kali/lib


$(SONAME).$(VERSION)    :  $(OBJS)
        $(CC) $(CFLAGS) -shared -Wl,-soname,$(SONAME) -o $(SONAME).$(VERSION) $(OBJS)

$(OBJS)                 :  $(SRCS)
        $(CC) -c $(SRCS)

clean      :
        rm -f $(SONAME).$(VERSION) libll.o libll.h.gch

install    :     $(SONAME).$(VERSION)
        mkdir -p $(LIBPATH)
```

```
mkdir -p /home/kali/include
cp ./$(SONAME).$(VERSION) $(LIBPATH)
cp ./libll.h /home/kali/include
ldconfig -n $(LIBPATH)
```

## 1.2 Description

To create a shared object file using a makefile, you can use the following steps:

Define the compiler and compiler flags to be used for creating the shared object file. For example, you might use gcc as the compiler and the -fPIC flag to generate position-independent code: makefile Copy code CC=gcc CFLAGS=-fPIC Define the source files and object files to be used in the shared object file. For example: makefile Copy code SRCS=foo.c bar.c OBJS=$(SRCS:.c=.o) In this example, the SRCS variable lists the source files for your program, and the OBJS variable lists the corresponding object files that will be generated.

Define the name and version of the shared object file. For example: makefile

Copy code SONAME=mylib.so VERSION=1 In this example, the SONAME variable specifies the name of the shared object file, and the VERSION variable specifies the version number.

Define the build target for the shared object file. For example:

Copy code $(SONAME): $(OBJS) $(CC) -shared -Wl,-soname,$(SONAME).$(VERSION) -o $(SONAME).$(VERSION) $(OBJS) This target specifies that the shared object file should be built from the object files generated by compiling the source files. The -shared flag tells the linker to generate a shared object file, and the -Wl,-soname option specifies the name of the shared object file. Finally, the -o option specifies the output file name.

Define a clean target to remove any generated object files or shared object files. For example: makefile Copy code clean: rm -f $(OBJS) $(SONAME).$(VERSION) This target removes any object files generated during the build process, as well as the shared object file.

Here is an example makefile that puts all these steps together:

```
CC=gcc CFLAGS=-fPIC

SRCS=foo.c bar.c OBJS=$(SRCS:.c=.o)
```

```
SONAME=mylib.so VERSION=1

$(SONAME): $(OBJS) $(CC) -shared -Wl,-soname,$(SONAME).$(VERSION)
-o $(SONAME).$(VERSION) $(OBJS)

clean: rm -f $(OBJS) $(SONAME).$(VERSION)
```

You can run the makefile by
navigating to the directory containing the makefile and running
make. This will build the shared object file specified by the SONAME
variable. You can also run make clean to remove any generated files.

User is version number necessary for the above code ChatGPT While it is not strictly necessary to include a version number in the shared object file name, it is a common practice to do so. Including a version number in the shared object file name allows for multiple versions of the same library to be installed on a system, and for programs that depend on a specific version of the library to link to that version.

In the makefile example I provided earlier, the version number is included in the shared object file name using the following line:

javascript Copy code -Wl,-soname,$(SONAME).$(VERSION) This line uses the -Wl option to pass an argument to the linker, and specifies the soname of the shared object file as $(SONAME).$(VERSION). This means that the shared object file will be named mylib.so.1, where 1 is the version number.

If you do not include a version number in the shared object file name, you may run into issues if you need to update the library or install multiple versions of the library on a system. Therefore, it is generally a good practice to include a version number in the shared object file name.

User CFLAGS=-fPIC what does this do in the above code ChatGPT In the makefile example I provided earlier, the line CFLAGS=-fPIC sets the compiler flag -fPIC for all compilation steps.

The -fPIC flag is short for "Position Independent Code", and it

```
generates code that can be loaded and executed at any memory address.

This is useful for shared libraries, as it allows the library to be
loaded at any address in memory, rather than being tied to a specific
address.

When creating a shared object file, it is important to generate
position independent code, as the

shared object file may be loaded into memory at different addresses on
different systems. By using the -fPIC flag, we ensure that the code
generated by the compiler is position independent and can be loaded
into memory at any address.

Therefore, the CFLAGS=-fPIC line in the makefile is setting a compiler
flag that is necessary for creating a shared object file.
```

# 2 Running the Code

## 2.1 terminal command

gcc -L /home/kali/lib/ libll_test.c -lll

## 2.2 Description

Here -lll is -l option with **ll** which is libll (my library name) minus the "lib" part (standard convention) for running this .