Umeå University
Department of Applied Physics and Electronics

Linux as Development Environment 7.5 ECTS
5EL142 HT-16

Assignment 6 - Libraries

Submitted 2016-11-07
Author: LORENZ GERBER, 20161202-2033 (lorenzottogerber@gmail.com)
Instructor: Sven Rönnbäck, Björne A Lindberg

1 Library 'libpower'

1.1 Function and Usage

I implemented the library 'libpower' and small program to test it's functionality. The usage of the test program is straight forward as it does not need any command line arguments. After calling the program (which we here can call 'test'), a short text-based user menu is shown to access the two functions included in the library. The two functions are: 1. calculate power in a circuit from voltage and resistance and 2. calculate power in a circuit from voltage and current. After choosing the function by entering either 'R' or 'I' and pressing the 'enter' key, the program will query for the respective values and then calculate power. The result is printed to stdout.

1.2 Algorithm

In the case of 'libpower' there was not much of an algorithm to implement. The functions could be written directly as arithmetic expressions and needed also no special library.

1.3 Building

Building the library and the test program was done in four steps:

```
gcc -Wall -std=c11 -c -fPIC libpower.c
gcc -shared -o libpower.so libpower.o
gcc -Wall -std=c11 -c test.c
gcc -o test test.o -L. -lpower -Wl,-rpath,.
```

First the library is compiled and an object file is generated. Then a shared library file is generated. Now the test program is compiled. Finally the object from the test programme and the shared library are linked in the last gcc call.

Used Compiler / Linker Flags

In all compile steps, `Wall` and `-std=c11` is used. `-Wall` is the flag to enable all warnings while `-std=c11` requires checking of the code according to the ISO C11 standard. The `-c` flag instructs gcc to stop after compilation with object files. With the `-o` *file* flag an output file can be indicated. The `-L.` option states the search path for libraries. The `-Wl` flag is used to pass comma-separated options to the linker program. In the current case, this is the option `rpath` to add a directory to the runtime library search path. Here the working directory is indicated.

2 'electrotest' and 'makefile'

We wrote the main program 'electrotest' to conform with the output given in the laboration instructions. 'electrotest does not take any command line arguments but works interactively. First the user is asked to enter voltage, type of linking (sequence or parallel) and number of components. All user input is handled with the 'fgets' function and a string array 'buf'. After the first three questions, the program goes into a FOR loop based on the number of components. Here the user enters the resistances of each component. If the user enters an invalid resistance or hit the enter key without entering any resistance, the FOR loop is ended. Then follows the part where the library functions are used to calculate the results. Here we implemented an end of program error handling with 'goto' jumps. Dynamic memory

allocation is used once in electrotest, namely to reserve space for the resistance values as the number of components is decided at run-time by the user.

2.1 Group Members

The group members where Peter Püschel (libresistance), Stephen Burleigh (libcomponent) and Lorenz Gerber (libpower).

2.2 Usage of the Dynamic Libraries

The libraries were made accessible in electrotest by including the header file of each library. In this case, we did not make a header file for 'electrotest' hence the includes were stated directly in the main file 'electrotest.c'. After including the header files, the functions of each library were available and recognized in the main program.

2.3 Building

As requested in the assignment, the building process was automated using a makefile. The makefile includes 14 targets: all, install, lib, electrotest_local, electrotest, electrotest.o, libcomponent.so, libresistance.o, libresistance.so, libpower.o, libpower.so, clean and uninstall. 'make all' is used to build the program in the working directory while the libraries are copied in new subdirectory called 'lib'. 'make install' builds 'electrotest' and the 'libraries' but then copies them to `/usr/bin/` and `/usr/lib/` respectively. 'make uninstall' removes binaries and libraries from all earlier mentioned locations while 'make clean' just cleans up in the working directory. Some `make` commands may need sudo rights to proceed.

The actual build process works in the same way as described earlier: compiling the libraries using -fPIC into object files, producing shared libraries by using the '-shared' flag, then compiling 'electrotest' into an object file and finally linking 'electrotest' object file with the dynamic libraries. Here the used flags are `-L.` to indicate the search path for the libraries then followed by the names of the libraries. Finally the -rpath flag is used to add a run-time library search path for the libraries. This is specific for dynamic libraries.

3 Collaborative Development

After assignment of the group by the supervisor, we had a short chat session on Skype where we decided the rough setup for the project. Peter Püschel was suggested as project coordinator by the supervisor, which everybody agreed on. Then the workpackages where assigned to each member. Initially, we just assigned the libraries and left open who would implement 'electrotest' and the makefile. Finally we agreed on an approximate time schedule. The start of the work was about one week into the future. Git was chosen to be the versioning/repository of our choice and Peter offered to set up and administrate the repo. All members had used git before. Initially, the work was rather individual until the libraries became usable. Then, 'electrotest' and the makefile were written in a truly collaborative way. Some testing was conducted in the end until we finally all agreed that the specifications from the assignment were met.

3.1 Account on Current Collaborative Work

I happened to choose the simplest library, 'libpower'. To make up for that, I wrote large parts of the makefile. Collaborating with the other members worked fine. After an initial

chat on Skype, we communicated mostly by email reply's to all. In a couple of cases we also used the commenting function on github. Working with the repository was without any problem. I experienced maybe twice source code conflicts that needed to be resolved, which also worked well. All in all, this lab was a really nice and positive experience.

3.2 Large Collaborative Projects

In larger project, it would be necessary to define code style standards to simplify reading each others code. Also it would make sense to use more advanced versioning features such as branching and pull request. Some sort of task tracker would certainly also be nice.