Playing with fire

I feel like this is a little obvious, but I was thinking about synthesizers lately, and wondered about writting a minimal music program. The synthesizers could be dynamic, shared libraries, and you could load them on the fly (provided that the API is quite well defined).

I remmembered some things about dlopen, and wanted to do a quick test to see if you could have some sort of blackbox that could load libraries on the fly, and pass values. For simplicity, we can think of such a thing as a function that takes an integer, and returns an integer:

```
int execute(int);
```

We can have two libraries that implement this interface, and we can then compile them with the following incatations:

```
$CC -fPIC -c plugin1.c -o plugin1.o
$CC -shared -o plugin1.so plugin1.o

$CC -fPIC -c plugin2.c -o plugin2.o
$CC -shared -o plugin2.so plugin2.o
```

You could probably get away without `-fPIC`, but I'm the kind of person that swears is being followed by shadowy figures on a daily basis.

Finally we compile the program that is supposed to load the libraries.

```
CC=gcc
CFLAGS="-Wall -Werror"

$CC $CFLAGS main.c -o plugintest -ldl
```

For simplicity's sake, we don't go too crazy on the test program either. We expect the name of the `so` file to be passed as an arg. The first important function to notice is `dlopen`. You pass the name of the library you want to open as a string (and yes, null terminated):

```
  if (argc != 2) {
    usage(argv[0]);
    exit(1);
  }

  char *error = NULL;
  void *handle = dlopen(argv[1], RTLD_LAZY);
  int (*pluginfunc)(int);
```

Next important tidbit is here:

```
  /* name of function you want to act as a pluggable interface */
  pluginfunc = dlsym(handle, "execute");
```

We use dlsym to find the address of the function called `execute` in the library, and finally invoke:

```
  printf("pluginfunc result: %d\n", pluginfunc(10));
```

Considering the two implementations:

```
/* plugin 1 multiplies by 2 */
int execute(int val) {
  return val * 2;
}

/* plugin 2 divides by 2*/
int execute(int val) {
  return val / 2;
}
```

We, as expected, get the following output on the invokation of both plugins:

```
psyomn@minixaos ~/programming/notes/languages/c/dlopen $ ./plugintest ./plugin1.so
pluginfunc result: 20
psyomn@minixaos ~/programming/notes/languages/c/dlopen $ ./plugintest ./plugin2.so
pluginfunc result: 5
```

That was kind of fun :).

Edit: added some corrections, thanks to @RAttab (github.com/RAttab)