Lab 5. Layer Programming

Hu Zhengdong 517370910249
Oct. 28, 2020

I. Layer programming

- The program can be divided into three layers, what are they?
  - Interface layer, API layer, Kernel layer

- Split the program into files according to the defined layers.
  - Interface layer: interface.c/h
  - API layer: api.c/h
  - Kernel layer: kernel.c/h

- Create the appropriate corresponding header files.
  - As shown above

- If necessary, rewrite functions such that no call is emitted from lower level functions to upper level functions.
- The initial program implements a command line interface, write a “Menu interface” which (i) welcomes the user, (ii) prompts him for some task to perform, and (iii) runs it. When a task is completed the user should (i) be informed if it was successful and then (ii) be displayed the menu. From the menu he should be able to exit the program.
  - Menu interface: Menuinterface function in interface.c/h
  - Command interface: Cmdinterface function in interface.c/h

- Write two main functions, one which will “dispatch” the work to another function which will run the command line user interface and asecond one which will “dispatch” the work to the Menu user interface.
  - Main1: Cmd.c
  - Main2: Menu.c

II. Libraries

- What are the three stages performed when compiling a file?
  - Pre-processing, Compilation, Linking
- Briefly describe each of them
  - Pre-processing: The preprocessor does some initial processing. This includes joining continued lines (lines ending with a \) and stripping comments, interpreting definitions.
  - Compilation: The preprocessed code is translated to assembly instructions specific to the target processor architecture. Then, an assembler is used to translate the assembly instructions to object code.
  - Linking: To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in. This process is called linking.

- Explain the difference between static and dynamic libraries?
  - A Static library or statically-linked library is a set of routines, external functions and variables which are resolved in a caller at compile-time and copied into a target application by a compiler, linker, or binder, producing an object file and a stand-alone executable.
  - Dynamic libraries are linked dynamically simply including the address of the library (whereas static linking is a waste of space). Dynamic linking links the libraries at the run-time. Thus, all the functions are in a special place in memory space, and every program can access them, without having multiple copies of them.

- How to create static libraries?

```
## Add in cmakelist.txt
add_library(kernel_static STATIC kernel.c)
add_library(api_static STATIC api.c )

add_executable(Cmdlist Cmd.c interface.c interface.h)
target_link_libraries(Cmdlist api_static kernel_static)
```

- How to create dynamic libraries?

```
##Add in cmakelist.txt
add_library(kernel_dynamic SHARED kernel.c)
add_library(api_dynamic SHARED api.c)
target_link_libraries(api_dynamic kernel_dynamic)

add_executable(Cmdlist Cmd.c interface.c interface.h kernel.c api.c)
add_executable(Menulist Menu.c interface.c interface.h kernel.c api.c)

add_executable(Cmdlist Cmd.c interface.c interface.h)
add_executable(Menulist Menu.c interface.c interface.h)
target_link_libraries(Cmdlist api_dynamic kernel_dynamic)
target_link_libraries(Menulist api_dynamic kernel_dynamic)
```

- What's the difference between API and libraries?
  - A library refers to a chunk of running codes that are designed to be reused during development, an API is the method or means by which these running codes interact with each other.
  - An API can be thought of as the logical representation of what is in the library, or the consistent format that explains what a developer can do with the library.
  - It is the part of the code that is accessible to programmers. The main difference is that the library refers to the code itself, while API refers to the interface.

- Implement the API for the two libraries
  - Reference /src