PA2 实验报告

张铭方 161220169

必答题

编译与链接

问题一： 在 `nemu/include/cpu/rtl.h` 中, 你会看到由 `static inline` 开头定义的各种RTL指令函数. 选择其中一个函数, 分别尝试去掉 `static` , 去掉 `inline` 或去掉两者, 然后重新进行编译, 你会看到发生错误. 请分别解释为什么会发生这些错误? 你有办法证明你的想法吗?

答：

1. 去掉 `static` ：

   现象：

   一般的，通过各种测试，没有明显影响。

   若修改特别的某些函数：`error: ... is static but used in inline function ... which is not static`

   解释：

   一般的，`static` 将其作用对象声明为Internal Linkage的，不能给文件外使用。但是若其它文件有 `#include "rtl.h"` 则相当于复制了 rtl.h 中的代码，因此是可以使用 rtl.h 中的 `static` 内容的。

   特别的，若修改嵌套着 `static` 函数的函数则不能去掉 `static`。

2. 去掉 `inline`：

   现象：

   `error: ‘rtl_li’ defined but not used [-Werror=unused-function]`

   解释：

   将函数声明为 `inline`，编译器不把它当做是一个函数，直接嵌套汇编代码，取代函数调用，提高效率。去掉 `inline` 后，编译器还是将其当做函数处理，由于没有被使用过而 warning 又被视为 error，故出现此错误。

3. 二者同时去掉：

   现象：

`/home/Manfred/ics2017/nemu/./include/cpu/rtl.h:12: rtl_li 的多重定义`

解释:

去掉 `inline` 说明编译器将其看做函数，去掉 `static` 说明函数变成 `GLOBAL` 的函数不能重复定义。而 `#include "rtl.h"` 的文件有很多，会重复定义多次。

问题二:

1. 在 nemu/include/common.h 中添加一行 volatile static int dummy; 然后重新编译NEMU. 请问重新编译后的NEMU含有多少个 dummy 变量的实体? 你是如何得到这个结果的?

   答：用 `grep` 命令发现在30个可重定位.o文件中存在 dummy。

2. 添加上题中的代码后, 再在 nemu/include/debug.h 中添加一行 volatile static int dummy; 然后重新编译NEMU. 请问此时的NEMU含有多少个dummy 变量的实体?

   答：30个。因为 common.h 中有 `#include "debug"`

3. 修改添加的代码, 为两处 dummy 变量进行初始化: volatile static int dummy = 0;然后重新编译NEMU. 你发现了什么问题? 为什么之前没有出现这样的问题?

   现象:

   `./include/common.h:31:21: error: redefinition of ‘dummmy’`

   解释:

   之前弱符号定义可以重复，而现在强符号定义不可重复。

了解 Makefile

问题:

请描述你在 nemu 目录下敲入 make 后, make 程序如何组织.c和.h文件, 最终生成可执行文件 nemu/build/nemu .

(这个问题包括两个方面: Makefile 的工作方式和编译链接的过程.)

关于 Makefile 工作方式的提示:

- Makefile 中使用了变量, 包含文件等特性
- Makefile 运用并重写了一些implicit rules
- 在 man make 中搜索 -n 选项, 也许会对你有帮助
- RTFM

答:

Makefile 的工作方式:

1. make 会在当前目录下找名字叫“Makefile”或“makefile”的文件。
2. 如果找到，它会找文件中的第一个目标文件（target），并把这个文件作为最终的目标文件。
3. 如果目标文件不存在，或是其所依赖的后面的 .o 文件的文件修改时间要比这个文件新，那么就会执行后面所定义的命令来生成目标文件。
4. 如果目标文件所依赖的.o文件也不存在，那么make会在当前文件中找目标为.o文件的依赖性，如

果找到则再根据那一个规则生成.o文件。
5. 对于nemu的Makefile，它会寻找nemu/src目录下的所有.c文件和build/obj目录下名称相对应的.o文件。然后将.c文件使用gcc编译成同名.o文件。最后链接成可执行目标文件。

编译链接的过程：

`make -n` 指令可以输出 make 将要完成的指令。

1. 创建目录，存放源文件的编译出的可重定位目标文件。
2. `git add`, `git commit`
3. 将 .o 文件链接起来。