Introduction to Computer System

PA 2

实验报告

姓名: 郑奘巍

学号: 171860658

邮箱: zzw@smail.nju.edu.cn

院系: 计算机科学与技术系

2018 年 10 月 10 日

第一部分　实验进度

我完成了所有内容。

第二部分　必答题

问题 0.1 编译与链接: 关于 *static* 与 *inline*

解答 选择 *static inline void interpret_rtl_li* （其它尝试也是相同问题）

去掉 *inline :* 正常编译，运行
去掉 *static :* 正常编译，运行
去掉两者: 编译不通过，出现 *multiple definition* 的错误

由于函数的定义直接在.*h* 文件中，该文件又在多个文件中被引用，如果删掉这两者，则同一个函数在多处定义，符号相同导致出错。

当存在 *static* 修饰时，定义的 *rtl* 函数对外不可见，故可以重复被多个 *c* 文件引用。

当存在 *inline* 时，由于 *rtl* 语句简短，都成功嵌入，此时不存在函数定义，故不会产生问题。这样做还能提升程序性能。

验证如下：在 *a.h* 中定义函数 *fun()*，分别在 *b.c,c.c* 中引入 *a.h*。当 *fun()* 不添加任何限定符时，编译不通过。添加 *static*，利用 *gcc -E* 进行宏扩展, 观察到 *fun* 函数都添加了 *static* 限定符。

再添加 *inline*，利用 *objdump* 反汇编，发现函数功能果然直接在 *main* 中实现了。

问题 0.2 编译与链接

解答 *1.* 在 *common.h* 中增加后 *nemu* 中有 *30* 个 *dummy* 变量。
得到的方法是：在 *Makefile* 中新建命令 *tmp*, 定义（和一些其它变量）如下:

```
DIRS := $(shell find src -type d)
FILES := $(foreach dir, $(DIRS), $(wildcard $(dir)/*.c))
tmp:
    $(CC) -E $(CFLAGS) -c $(FILES) > tmp

```

接着打开 *tmp* 文件用 *vim* 命令:*%s/int dummy//gn* 得到 *dummy* 的实体变量个数

2. 继续在 *debug.h* 中增加后 *nemu* 中有 *60* 个 *dummy* 变量。这是上一题的两倍，这是因为在 *common.h* 中 *include debug.h* 并且在 *debug.h* 中 *include common.h,* 两者必然同时出现。

3. gcc 默认编译*-std=gnu11*

此时发生了重定义错误。原因是，在 *C* 标准中，*static* 修饰的 *File scope declaration* 属于暂定的变量定义 (*tentative declaration*)，即如果有初始化 *(initializer)*，则此处算定义；否则，编译器暂定它是变量定义：当程序中有该变量的定义时，使用其它定义，否则使用这个暂定的定义并赋初值 *0*。

在 *2* 中，我们进行了两次暂定的变量定义，编译器会认为其中一个是声明，另一个是定义。其实，任意多个没有初始化的 *static* 定义都是可以的。而赋初值后，两处都成为了定义导致重定义错误。带初值的 *static* 定义最多有一处。

问题 0.3 了解 *Makefile*

解答 *make* 的工作方式：

make 中定义了缺省目标为 *app* 故执行的是 *app* 的命令。这里是变量 *$(BINARY),* 全部展开后为*./build/nemu* (此处不考虑 *$(SHARE)* 为 *1*)。*Makefile* 会查看该文件的依赖。

但在此之前，*-include $(OBJ:.c=.d)* 会先被执行来更新编译所有*.c* 文件。*SRCS* 变量是 *src/*下所有文件夹的列表，通过 *$(SRC:src/%.c=$(OBJ_DIR)/%.o) (*这里用了符号替换和模式匹配*)* 来得到 *src/*下所有*.c* 文件名对应的*.o* 文件的列表。于是，*include* 试图引入这些*.d* 文件但不存在 (由于有-，错误忽略)。但同时，*include* 的文件同样会进行模式规则匹配，这便符合了 *Complication patterns* 注释下的模式规则。

该规则首先输出 *gcc +* 改变过的文件 (即 *$<*) 信息，将 *cc* 一行变量全部展开得 *gcc -O2 -MMD -Wall -Werror -ggdb3 -I./include -fomit-frame-pointer -DDIFF_TEST_QEMU -c* 编译到相应*.o* 文件中。我们按下不表。

接着才到 *app* 命令。根据 *nemu* 的依赖规则，任意一个 *src* 下*.c* 的改动都使其重新生成。先执行 *git_commit* 函数 (在 *Makefile.git* 下)，参数为 *compile*。接着输出 *LD* 信息。最后一句展开变量为：*gcc -O2 -rdynamic -o ./build/nemu (*所有*.o* 文件*) -lSDL2 -lreadline -ldl*

编译链接过程：

编译：采用 *O2* 优化，开启所有警告且视为错误，将*./include* 作为 *#include* 默认搜索地址之一，自动查找依赖，添加 *DIFF_TEST_QEMU* 宏，打开 *omit-frame-point* 优化，允许 *gdb* 调试，

链接：采用 *O2* 优化，链接 *SDL2,readline,dl* 库，将所有符号添加到动态符号表中，将所有*.o* 文件链接生成 *nemu*

第三部分　致谢

宋劲杉 Linux C 编程一站式学习 https://docs.huihoo.com/c/linux-c-programming/index.html 网站给了我很大的帮助

感谢肖江同学为我 video.c 中的一个 bug 提供了重要线索。