OS Project Part 1: 内核输出函数与同步原语

提交须知：本Lab包括至少8个函数填空检查点与3个问题，只有输出预期的结果并且回答完问题才可以获得全部分数。

步骤1: 构建内核与简单的GDB使用

acmOS是一个使用C语言编写的RISC-V操作系统。在开始写操作系统之前，首先需要了解如何构建内核。本实验通过QEMU模拟在RISC-V硬件上的执行，建议版本为5.0。QEMU可以配合GDB进行调试，提供包括单步调试、变量打印、内存查看等功能。在项目根目录，可以通过`./script/docker_build.sh`构建系统镜像。如果没有构建出任何错误，屏幕上应该会显示如下的结果。

```
Scanning dependencies of target acmOS_spr21-common
[ 10%] Building C object kernel/common/CMakeFiles/acmOS_spr21-common.dir/lock.c.o
[ 20%] Building C object kernel/common/CMakeFiles/acmOS_spr21-common.dir/printk.c.o
[ 30%] Building C object kernel/common/CMakeFiles/acmOS_spr21-common.dir/uart.c.o
[ 40%] Building ASM object kernel/common/CMakeFiles/acmOS_spr21-common.dir/kernelvec.S.o
[ 50%] Building C object kernel/common/CMakeFiles/acmOS_spr21-common.dir/string.c.o
[ 50%] Built target acmOS_spr21-common
Scanning dependencies of target acmOS_spr21-arch
[ 60%] Building C object kernel/CMakeFiles/acmOS_spr21-arch.dir/main.c.o
[ 60%] Built target acmOS_spr21-arch
Scanning dependencies of target acmOS_spr21-boot
[ 70%] Building C object kernel/boot/CMakeFiles/acmOS_spr21-boot.dir/start.c.o
[ 80%] Building ASM object kernel/boot/CMakeFiles/acmOS_spr21-boot.dir/kernelvec.S.o
[ 80%] Built target acmOS_spr21-boot
Scanning dependencies of target acmOS_spr21-asm
[ 90%] Building ASM object kernel/asm/CMakeFiles/acmOS_spr21-asm.dir/boot.S.o
[ 90%] Built target acmOS_spr21-asm
Scanning dependencies of target kernel.img
[100%] Linking C executable kernel.img
[100%] Built target kernel.img
after make
```

并且可以在`build`目录下找到构建好的内核镜像`kernel.img`以及对应的符号文件`kernel.sym`。符号文件的作用是查看各个符号对应的地址，对调试有非常好的帮助。

在完成内核的构建之后，可以输入`make qemu-nox`启动qemu并且模拟执行我们构建好的内核。acmOS的标准输出应该显示在QEMU中，如下图。

```


UART Finish initialization. 1013 decimal is 3f5 hex, 1111110101 binary, 1013 dec.
kernel lock address: 0x8000a818
UART String: Hello, world!
[Pass: TEST_lock_test, kernel/common/lock.c:68]: "kernel lock: try_acquire"
[Pass: TEST_lock_test, kernel/common/lock.c:69]: "kernel lock: is_locked"
[Pass: TEST_lock_test, kernel/common/lock.c:71]: "kernel lock: is_locked"
[acmOS] Reaching Suspend point.
QEMU: Terminated
```

要退出QEMU，按下Ctrl-A，再输入X退出QEMU。

在学会了如何启动qemu进行模拟之后，尝试使用`make qemu-nox-gdb`命令创建一个远程目标（Remote Target），并且在另外一个窗口中执行`gdb-multiarch -n -x .gdbinit`从而连接到这个远程目标并且开始调试。在gdb-multiarch中的命令类似于gdb，可以通过`b <TARGET>`设置一个断点等。

问题1：内核在哪里?

(1) 尝试使用gdb单步跟踪内核的启动流程，并列出从启动qemu到进入到main函数的函数调用过程及其对应地址。Tips: 可以通过objdump查看生成elf的符号表，查询qemu的手册找到入点。

(2) 指出如何使用gdb中的一条指令显示(1)所需要的内容。

问题2：启动内核!

在我们使用QEMU模拟中，内核是如何启动的？和编译脚本的关联是什么？

步骤2: 内核输出函数 printk

在日常的C++代码实践中，大家想必一定是知道printf的使用方法的。printf接收不定长的参数，第一个参数为一个字符串作为format，第二个参数起作为输出的变量。运行的时候替换%开始的标识符为目标的内容。

在用户态的printf最终需要通过系统调用进入内核态输出函数。内核中也可以有一个输出函数，直接输出目标函数。本步骤的任务是完成printk相关内核代码的填空。printk和日常在stdio.h中的printf函数签名完全相同并且作用相同。

需要填写的函数：

- **`kernel/common/answer_printk.h:static void printk_write_num(int base, unsigned long long n, int neg);`**
- **`kernel/common/answer_printk.h:static void printk_write_string(const char *str);`**

printk_write_num

输出一个base进制的数n，如果是负数，neg为1，反之为0。

printk_write_string

输出一个字符串。

步骤提示：观察`kernel/common/printk.c:34:print_format`函数中对于普通字符的输出方式。

如果你的代码是正确的，在填写完相关的代码并且编译运行之后，你应该得到这样的输出。

```
UART Finish initialization. 1013 decimal is 3f5 hex, 1111110101 binary, 1013
dec.
kernel lock address: 0x8000a818
UART String: Hello, world!
```

到这里，你已经完成了printk函数的编写。

问题3: 内核输出

内核输出函数和通过stdio中的printf函数输出到屏幕的过程差异有哪些?

步骤3: 同步原语：互斥锁

在系统课程上，大家都学会了同步原语，并且知道了其中的一种是互斥锁。在内核中，因为有多核的参与，对内核的共享数据仍然存在竞争条件与临界区问题。因此，对内核的数据以及数据结构仍然需要使用同步原语进行保护。请你填写位于`kernel/common/answer_locks.h`中的acqiure、try_acquire、release、is_locked、holding_lock五个函数。实现一个简单的锁，锁的结构已经定义在`kernel/common/lock.h`中。

需要填写的函数：

- **`kernel/common/answer_locks.h:void acquire(struct lock *lock)`**
- **`kernel/common/answer_locks.h:int try_acquire(struct lock *lock)`**
- **`kernel/common/answer_locks.h:void release(struct lock* lock)`**
- **`kernel/common/answer_locks.h:int is_locked(struct lock* lock)`**
- **`kernel/common/answer_locks.h:int holding_lock(struct lock* lock)`**

步骤提示：内核代码中并没有原子变量，可以参考编译器层面的函数`__sync_lock_release,__sync_lock_test_and_set`。参考页面：`https://gcc.gnu.org/onlinedocs/gcc/_005f_005fsync-Builtins.html`

如果你的代码是正确的，在填写完相关的代码并且编译运行之后，你应该得到这样的输出。

```
UART Finish initialization. 1013 decimal is 3f5 hex, 1111110101 binary, 1013
dec.
kernel lock address: 0x8000a818
UART String: Hello, world!
[Pass: TEST_lock_test, kernel/common/lock.c:68]: "kernel lock: try_acquire"
[Pass: TEST_lock_test, kernel/common/lock.c:69]: "kernel lock: is_locked"
[Pass: TEST_lock_test, kernel/common/lock.c:71]: "kernel lock: is_locked"
[acmOS] Reaching Suspend point.
```

到这里，你已经完成了第一部分的全部内容。

关于评分的补充事项

本次作业步骤2采用输出比对，输出正确即为正确。步骤3由于在这一阶段我们并没有提供多核的支持，因此采用输出比对+Code Review方式进行。注意你可以修改的文件在没有特殊情况下仅为`answer_***.h`。当然，如果你觉得为了实现这个功能，这样的设计是非常差劲的，你也可以联系`@peterzheng`助教咨询后自主设计。

关于代码量

以下给出助教Demo的代码量，仅供参考

File	Function	LoCs
`answer_printk.h`	`void printk_write_num(int, unsigned long long, int);`	19
`answer_printk.h`	`void printk_write_string(const char *);`	3
`answer_locks.h`	`void acquire(struct lock *);`	4
`answer_locks.h`	`int lock_init(struct lock *);`	1
`answer_locks.h`	`int try_acquire(struct lock *);`	5
`answer_locks.h`	`void release(struct lock *);`	5
`answer_locks.h`	`int holding_lock(struct lock *);`	2
	总计	39

关于迟交

迟交1天之内作业成绩为原始成绩80%，超过24小时本次作业计0分。