中国科学技术大学计算机学院
《操作系统原理与设计》实验报告

实验题目：lab6_Scheduler

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020 年 6 月 14 日

计算机实验教学中心制

2019 年 09 月

实验目的

1. 实现两种调度算法。
2. 实现调度算法对应的任务管理器。

实验环境

1. PC 一台
2. Windows 系统
3. Ubuntu
4. QEMU
5. Xserver

软件框图

本实验的软件框图如图所示。软件层次分为 multiboot_header、myOS 和 userApp 三部分。

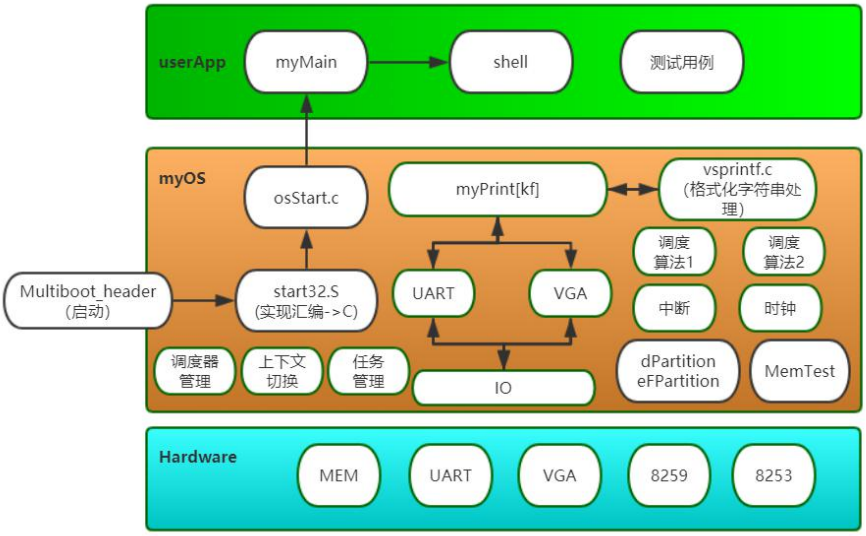

图 1

主流程

图 2

本实验的主流程如上图所示：

1. 在 multiboot_header 中完成系统的启动。
2. 在 start32.S 中准备好上下文，最后调用 osStart.c 把进入 c 程序。
3. 在 osStart.c 中完成初始化 8259A，初始化 8253，清屏，内存初始化，任务管理初始化等操作，最后切换至多任务状态。
4. 运行 myMain 中的代码，完成创建测试任务，shell 初始化，内存测试初始化等操作，启动 shell。

5. 进入 shell 程序，等待命令的输入。

主要功能模块及其实现

任务数据结构(TCB)及任务池(tcbPool)

该功能模块用于对任务的信息的管理及控制，主要包含栈顶指针，栈底指针，任务序号，任务状态。任务池则用 TCB 数组构成，用 next 指针链接成链表。

TCB 结构如下：

```
typedef struct myTCB {
    /* node should be the 1st element*/
    struct dLink_node thisNode;
    /* node body */
    unsigned long state;  // 0:rdy
    int tcbIndex;
    struct myTCB * next;
    unsigned long* stkTop;
    unsigned long stack[STACK_SIZE];
    tskPara para;
    unsigned int leftSlice; // for SCHED_RR or SCHED_RT_RR policy
} myTCB;
```

任务池结构如下：

图 3

任务创建/销毁

该功能模块用于创建，销毁，主要目的是实现 createTsk(),destroyTsk()函数。

createTsk()实现 TCB 分配，初始化调度参数，初始化栈，对下一空闲 TCB 进行修改，若此时为到达时间，调用 tskStart()，启动任务，否则调用 tskStartDelayed()函数，待到达时间到后再执行。

destroyTsk()实现 TCB 回收，将 TCB 链接到空闲 TCB 链表头，修改下一空闲 TCB，同时调度新任务执行。

任务参数

该功能模块用于任务调度时的任务参数的初始化，设置及读取，主要目的是实现 initTskPara()，setTskPara()，getTskPara()函数。

具体实现较为简单，详见 taskPara.c 文件中的代码。

统一的调度接口

该模块的主要功能是调度器的接口统一，实现方式为调用对应调度器的结构体的函数，具体实现详见代码。

调度算法 PRIORITY0、PRIORITY 的实现

这两种调度算法的实现与 FCFS 基本一致，只需在任务入队时，根据任务的优先及进行排序即可。因入队前，任务链表中的优先级是有序的，所以只需顺序查询，找到插入的节点，调用 dLinkList.c 中的函数完成插入即可。

具体实现代码如下：

```
void tskEnqueuePRIO0(myTCB *tsk)
```

```
{
    myTCB *point;
    point = rqPRIO0;
    if (point == NULL)
    {
        dLinkInsertBefore((dLinkedList *)point, (dLink_node *)point, (dLink_node *)tsk);
    }
    else
    {
        while (tsk->para.priority > point->para.priority && point->next != 0)
            point = point->next;
        if (tsk->para.priority > point->para.priority)
            dLinkInsertAfter((dLinkedList *)point, (dLink_node *)point, (dLink_node *)tsk);
        else
            dLinkInsertBefore((dLinkedList *)point, (dLink_node *)point, (dLink_node *)tsk);
    }
}
```

调度器管理的实现

该部分主要实现调度器的设置及相关参数的设定，主要目的是实现 getSysScheduler()，setSysScheduler()，getSysSchedulerPara()及 setSysSchedulerPara()。

具体实现较为简单，详见代码 task_sched.c 部分。

源代码说明

Makefile 组织

Makefile 组织结构如下：

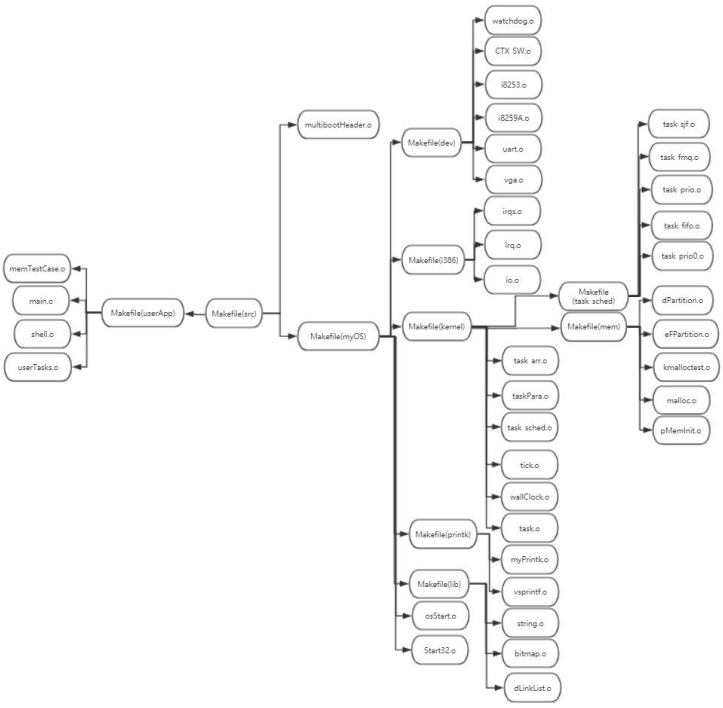

图 4

目录组织

目录组织如下图所示，主目录下

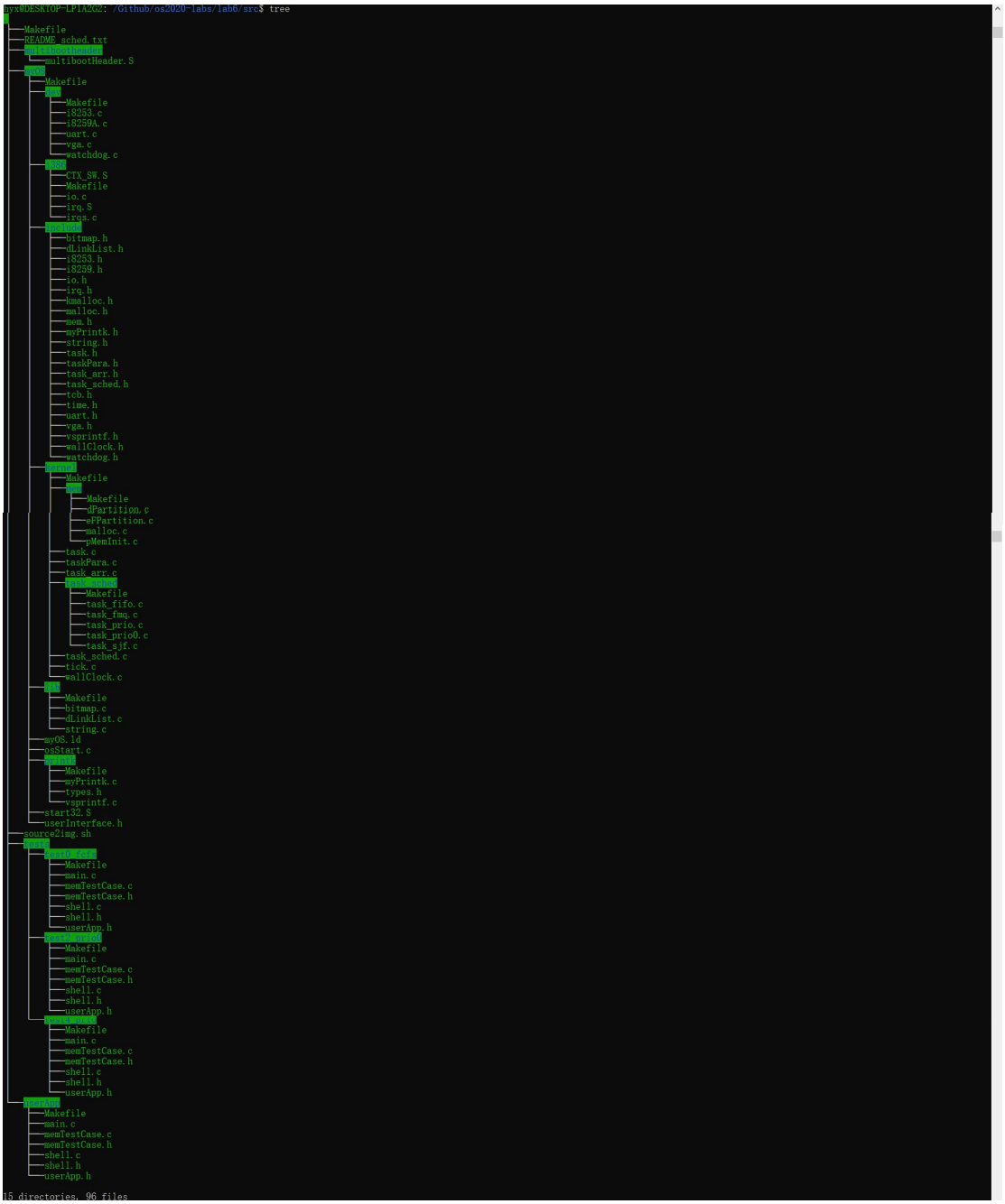

图 4

代码布局说明（地址空间）

从物理内存 1M 的位置开始放代码和数据，前面 12 个字节为 multiboot_header,向后对齐 8 个字节，放代码。再向后对齐 16 个字节，用于放初始化的数据（数据段）。在数据段之后，再向后对齐 16 个字节。之后为 BSS（Block Started by Symbol）段,用于存放程序中未初始化的全局变量和静态变量。并在 BSS 段后，再向后对齐 16 个字节。剩余部分为堆栈段。

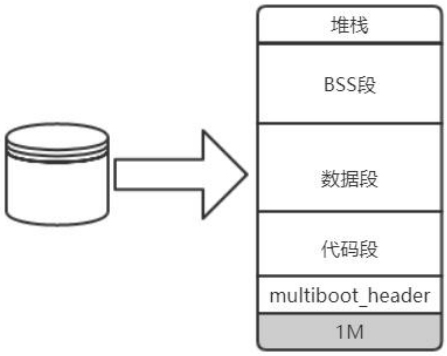

图 5

编译过程说明

```
ASM_FLAGS= -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
```

-m32 用 32 位机器的编译器来编译这个文件

--pipe 使用管道代替编译中临时文档

-Wall 打开警告选项

-fasm 识别 asm 关键字

-g 使用调试器 GDB

-O1 优化生成代码

-fno-stack-protector 停止使用 stack-protector 功能

```
C_FLAGS =  -m32 -fno-stack-protector -fno-pic -fno-builtin -g
```

-fpic 如果支持这种目标机,编译器就生成位置无关目标码.适用于共享库(shared library)

生成 multiHeader.o、osStart.o、start32.o、uart.o、vga.o、io.o、myPrintk.o、vsprintf.o 及 main.o 多个目标文件，链接生成 myOS.elf 文件。

运行和运行结果说明

PRIORITY0 的测试用例示意如下：

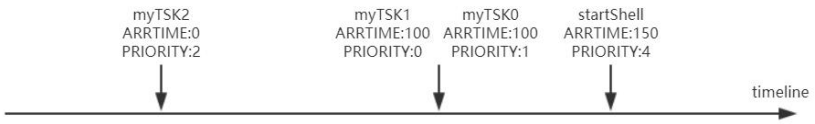

图 6

输入./source2run.sh test2_prio0 指令后，编译，链接，生成 myOS.elf 文件并运行之，输入 sudo screen /dev/pts/0，通过 Ubuntu 输入。运行结果如下图：

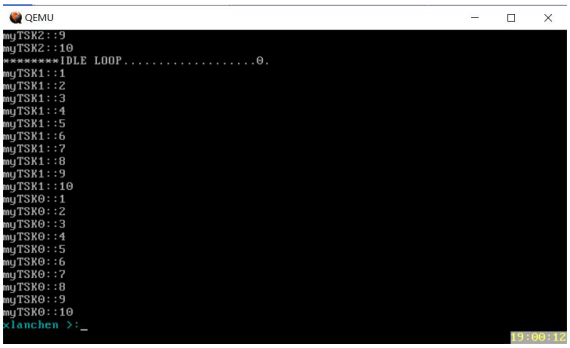

图 7

PRIORITY 的测试用例示意如下：

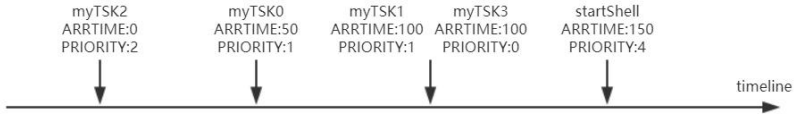

图 8

输入./source2run.sh test4_prio 指令后，编译，链接，生成 myOS.elf 文件并运行之，输入 sudo screen /dev/pts/0，通过 Ubuntu 输入。运行结果如下图：

图 9

遇到的问题和解决方案说明

1. 编译时出现报错“对‘_GLOBAL_OFFSET_TABLE_’未定义的引用”。

解决方案：在 src 目录下的 Makefile 文件的 CFLAGS 变量中添加-fno-pic.

2. 编译出现 fatal error: bits/libc-header-start.h: No such file or directory

解决方案:在 Ubuntu 中输入 apt-get install gcc-multilib，完善编译环境

3. 编译时出现 warning: assignment makes pointer from integer without a cast

解决方案：在给指针赋值前进行强制格式转化(unsigned short int*)

4. 编译时出现 warning: conflicting types for built-in function

解决方案：在 src 目录下的 Makefile 文件的 CFLAGS 变量中添加-fno-builtin