# 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：lab5_Task Manager & FCFS

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020 年 6 月 6 日

计算机实验教学中心制

2019 年 09 月

# 实验目的

1. 实现任务管理器。
2. 将 shell 封装成 task 来运行。
3. 将 shell 命令封装成 task 来运行。

# 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。

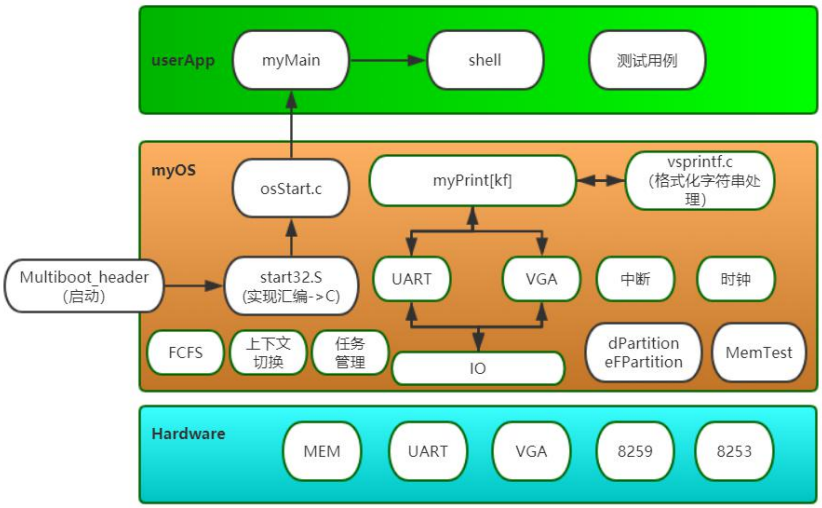


图 1

# 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，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
{
    unsigned long *stkTop;  /* 栈顶指针 */
    unsigned long tcbIndex; /* 任务 ID */
    unsigned long *stack;   /* 栈底指针 */
    unsigned long state;    /* 任务状态 */
    struct myTCB *next;     /* 下一任务 */
} myTCB;
```

任务池结构如下：


图 3

**任务创建/销毁及任务启动/终止**

该功能模块用于创建，销毁，启动，终止，通过对 FIFO 队列的控制及静态任务池实现，主要目的是实现 createTsk(),destroyTsk(),tskStart()以及 tskEnd()函数。

createTsk()实现 TCB 分配，初始化栈，对下一空闲 TCB 进行修改，最后调用 tskStart()，启动任务。

destroyTsk()实现 TCB 回收，将 TCB 链接到空闲 TCB 链表头，修改下一空闲 TCB。

tskStart()实现将任务状态置为 ready，将 TCB 入队。

tskEnd()实现任务 TCB 出队，调用 destroyTsk()销毁任务，调用 schedule()，调度下一任务。

**上下文切换**

该功能模块用于任务调度时的上下文切换，主要目的是将两个任务的栈顶指针找到，传递给 CTX_SW()函数。

具体实现如下：

```
void context_switch(myTCB *prevTsk, myTCB *nextTsk)
{
    //上下文切换
    prevTSK_StackPtr = &(prevTsk->stkTop);
    nextTSK_StackPtr = nextTsk->stkTop;
    CTX_SW(prevTSK_StackPtr, nextTSK_StackPtr);
}
```

**调度器 FCFS 的实现**

该模块的主要功能是实现任务的就绪队列，与 FIFO 队列基本相同，具体实现详见代码。

**Shell 及 Shell 命令封装的实现**

实现方式为将 startShell()修改为 createTsk(startShell)。

而 Shell 命令的封装如下：

```
tskid = createTsk(tmpCmd->func(argc, argv));
destroyTsk(tskid);
```

# 源代码说明

## 目录组织

目录组织如下图所示，主目录下

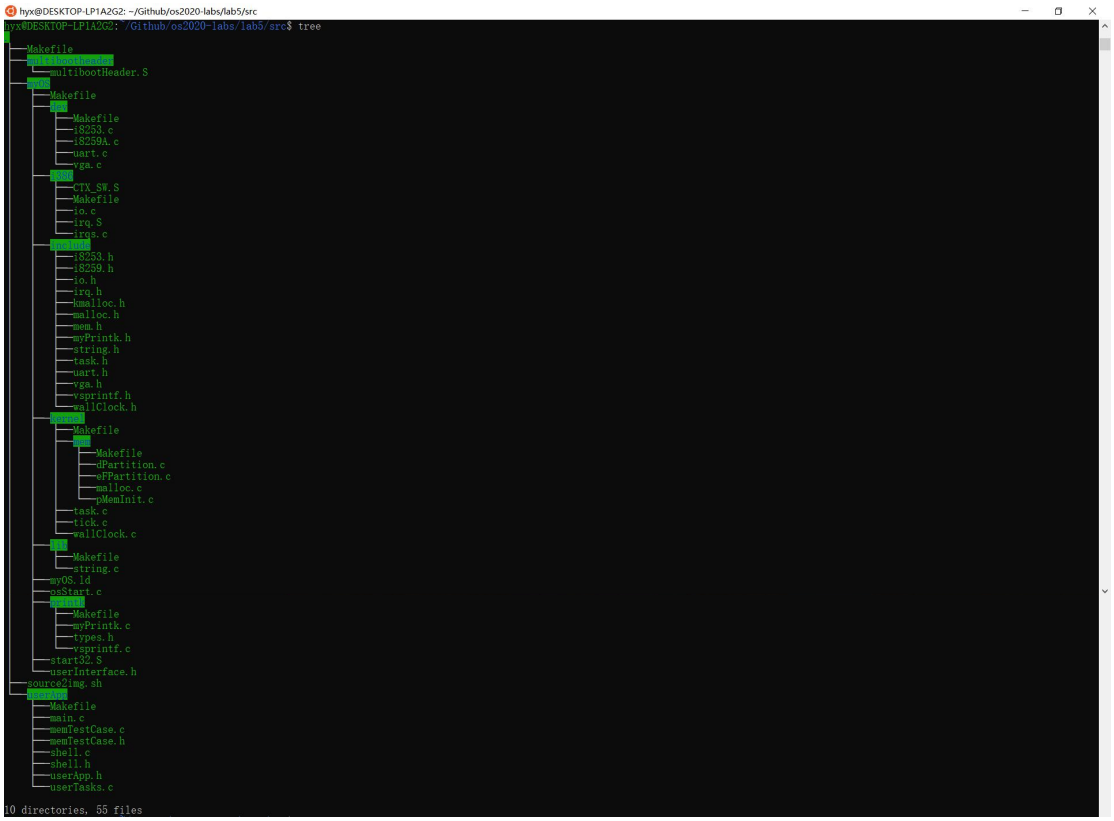


图 6

## Makefile 组织

Makefile 组织结构如下：

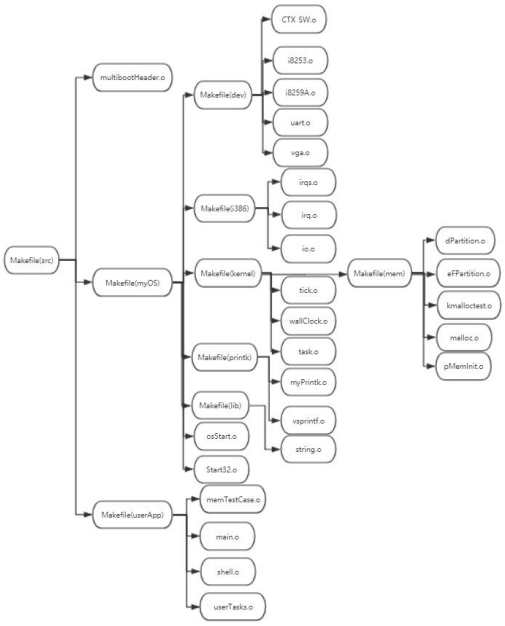

图 7

**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。

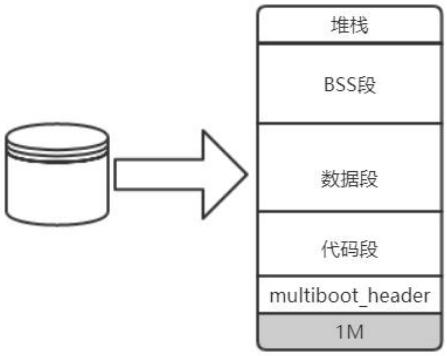


图 8

# 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

输入./source2run.sh 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：

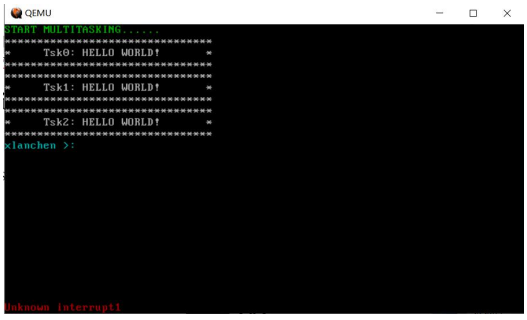

图 9

输入 cmd，进行测试。

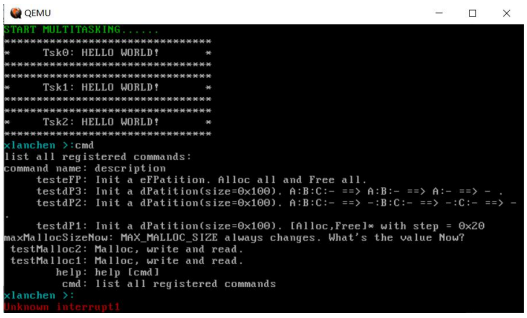


图 10

输入 testeFP，进行测试。

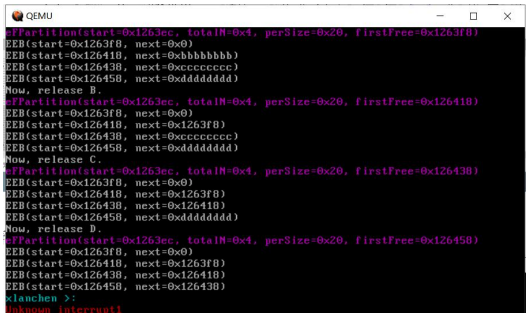


图 11

# 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