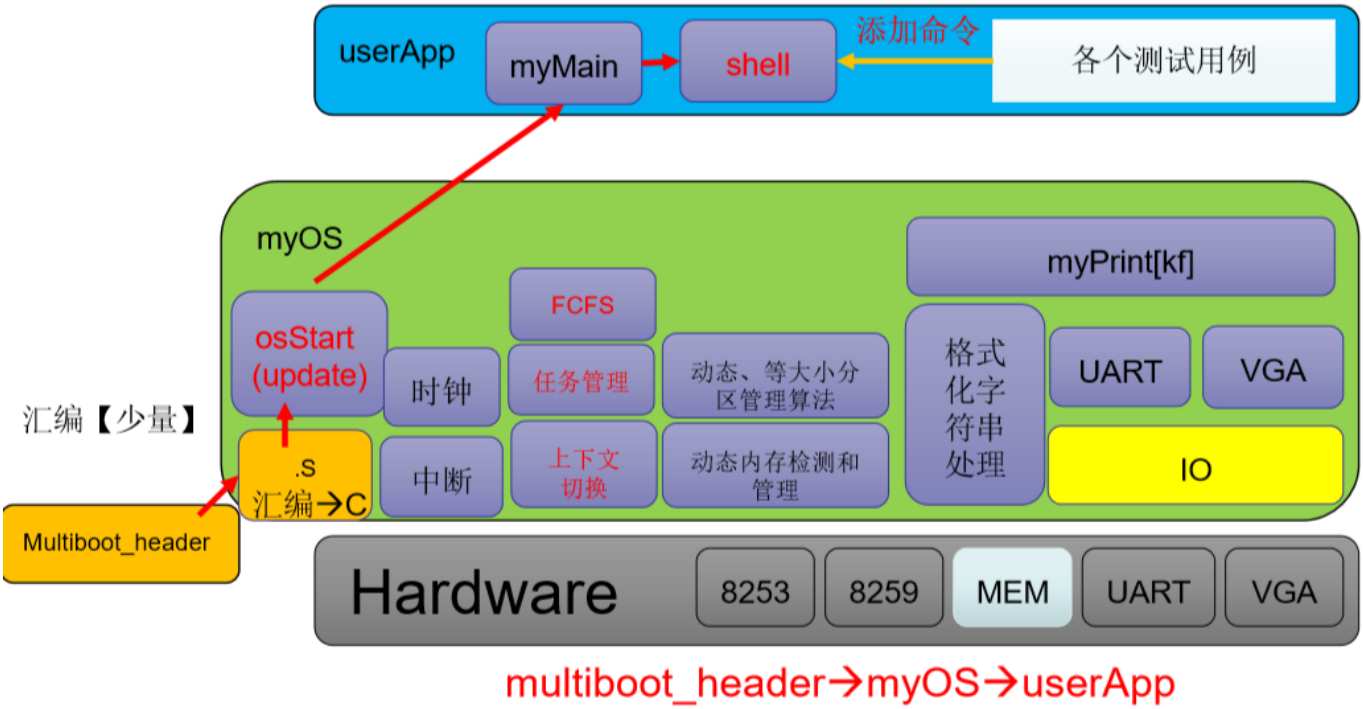
操作系统实验报告

实验五　TaskManager & FCFS

学号：PB18111683
姓名：童俊雄
完成时间：2020-05-09

一、　软件框图



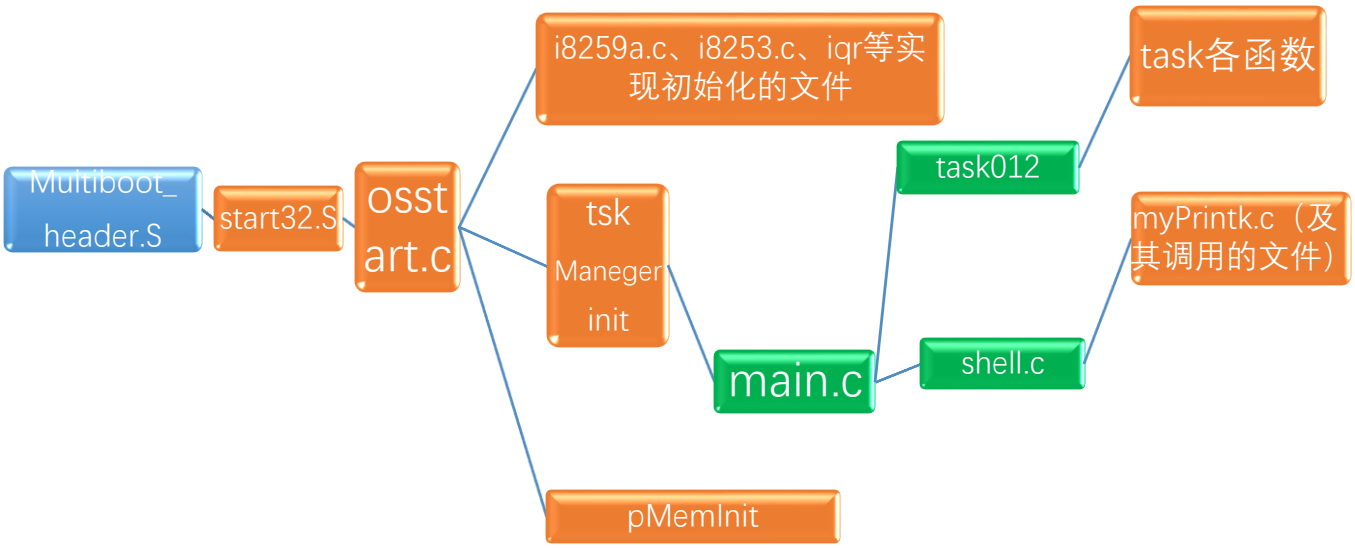
概述：软件大体上可以分为两个层次：用户程序和操作系统，其中操作系统又可以分为与用户程序的接口、IO设备的驱动程序、中断控制程序、时钟功能程序和内存管理程序等各功能模块，在每块中又可以进一步划分更细的层次（越上方的层次越高），如图所示。

二、　主流程说明

1. qemu启动header；
2. header通过调用myOS提供的_start，跳转到汇编文件start32.S；（进入myOS）
3. 从start32调用C语言入口，进入到c程序osstart.c；
4. osstart.c调用i8259A和i8253函数进行初始化，调用enable_interrupt函数启用中断；调用pMemInit接口进行内存检测；并调用TaskManagerInit的接口进入任务管理；
5. 任务管理通过inittskbdy与mymain函数对接；
6. mymain通过调用creattsk创建任务（包括shell）；

流程图：

（橙色表示myOS内的程序，绿色表示用户程序）



三、 主要功能模块说明&源代码说明

1. 任务管理模块

TCB结构体& rdyFCFS结构体：

```
typedef struct myTCB {
    unsigned long *stkTop;      /* 栈顶指针 */
    unsigned long state;
    int tcbIndex;
    struct myTCB* next;
    unsigned long stack[STACK_SIZE];
} myTCB;
```

```
typedef struct rdyQueueFCFS {
    myTCB* head;
    myTCB* tail;
    myTCB* idleTsk;
} rdyQueueFCFS;
```

FCFS队列的初始化、判断是否为空函数、next函数

```
void rqFCFSInit(myTCB* idleTsk) {
    rqFCFS.head = (void*)0;
    rqFCFS.tail = (void*)0;
    rqFCFS.idleTsk = idleTsk;
}

int rqFCFSIsEmpty(void) {
    //if empty return 1
    return (((rqFCFS.head == (void*)0)) && (rqFCFS.tail == (void*)0));
}

myTCB* nextFCFSTsk(void) {
    if (rqFCFSIsEmpty()) return rqFCFS.idleTsk;
    else return rqFCFS.head;
}
```

入队与出队

```
/* tskEnqueueFCFS: insert into the tail node */
void tskEnqueueFCFS(myTCB* tsk) {
    if (rqFCFSIsEmpty()) {
        rqFCFS.head = tsk;
    }
    else rqFCFS.tail->next = tsk;
    rqFCFS.tail = tsk;
}

//*****************************************

/* tskDequeueFCFS: delete the first node */
void tskDequeueFCFS(myTCB* tsk) {
    rqFCFS.head = rqFCFS.head->next;
    if (tsk == rqFCFS.tail) rqFCFS.tail = (void*)0;
}
```

初始化栈

```
// 用于初始化新创建的 task 的栈
// 这样切换到该任务时不会 stack underflow
void stack_init(unsigned long** stk, void (*task)(void)) {
    *(*stk)-- = (unsigned long)0x08;
    *(*stk)-- = (unsigned long)task;
    *(*stk)-- = (unsigned long)0x0202;

    *(*stk)-- = (unsigned long)0xAAAAAAAA;
    *(*stk)-- = (unsigned long)0xCCCCCCCC;
    *(*stk)-- = (unsigned long)0xDDDDDDDD;
    *(*stk)-- = (unsigned long)0xBBBBBBBB;

    *(*stk)-- = (unsigned long)0x44444444;
    *(*stk)-- = (unsigned long)0x55555555;
    *(*stk)-- = (unsigned long)0x66666666;
    *(*stk) = (unsigned long)0x77777777;
}
```

任务开始与中止接口：

```
void tskStart(myTCB* tsk) {

    tsk->state = TSK_RDY;
    tskEnqueueFCFS(tsk);
}

void tskEnd(void) {

    tskDequeueFCFS(currentTsk);
    destroyTsk(currentTsk->tcbIndex);
    schedule();
}
```

创建与销毁：

```
int createTsk(void (*tskBody)(void)) {

    myTCB* tsk = firstFreeTsk;
    if (tsk == (void*)0)//没有空闲块
        return -1;
    int i;
    tsk->state = TSK_NEW;
    stack_init(&(tsk->stkTop), *tskBody);

    for (i = 1; i < TASK_NUM; i++) {
        if ((tcbPool[i].state) == TSK_FREE) {
            firstFreeTsk = &tcbPool[i];
            break;
        }
    }
    if (i == TASK_NUM)      //没有空闲块
        firstFreeTsk = (void*)0;
    tskStart(tsk);
    return tsk->tcbIndex;

}

void destroyTsk(int takIndex) {

    tcbPool[takIndex].state = TSK_FREE;       //释放对应的TCB
    int i;
    for (i = 1; i < TASK_NUM; i++) {
        if (tcbPool[i].state == TSK_FREE) {
            firstFreeTsk = &tcbPool[i];
            return;
        }
    }

}
```

调度：

```
void scheduleFCFS(void) {
    prevTSK_StackPtr = currentTsk->stkTop;           //切换到下一个任务
    currentTsk = nextFCFSTsk();
    nextTSK_StackPtr = currentTsk->stkTop;
    CTX_SW(prevTSK_StackPtr, nextTSK_StackPtr);

}

void schedule(void) {
    scheduleFCFS();
}

/**
 * idle 任务
 */
void tskIdleBdy(void) {
    while (1) {
        schedule();
    }
}
```

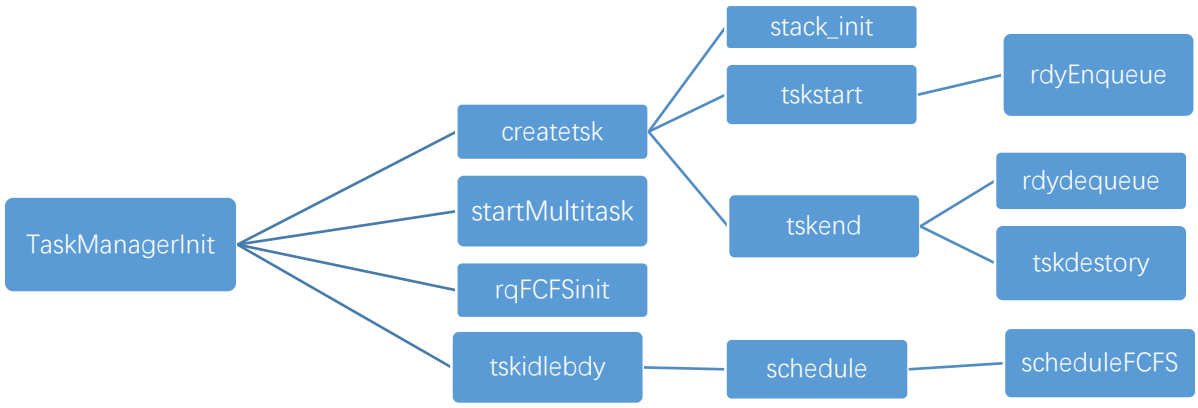
多任务：

```
 //start multitasking
void startMultitask(void) {
    BspContext = BspContextBase + STACK_SIZE - 1;
    prevTSK_StackPtr = &BspContext;
    currentTsk = nextFCFSTsk();
    nextTSK_StackPtr = currentTsk->stkTop;
    CTX_SW(prevTSK_StackPtr, nextTSK_StackPtr);

}
```

任务管理器初始化

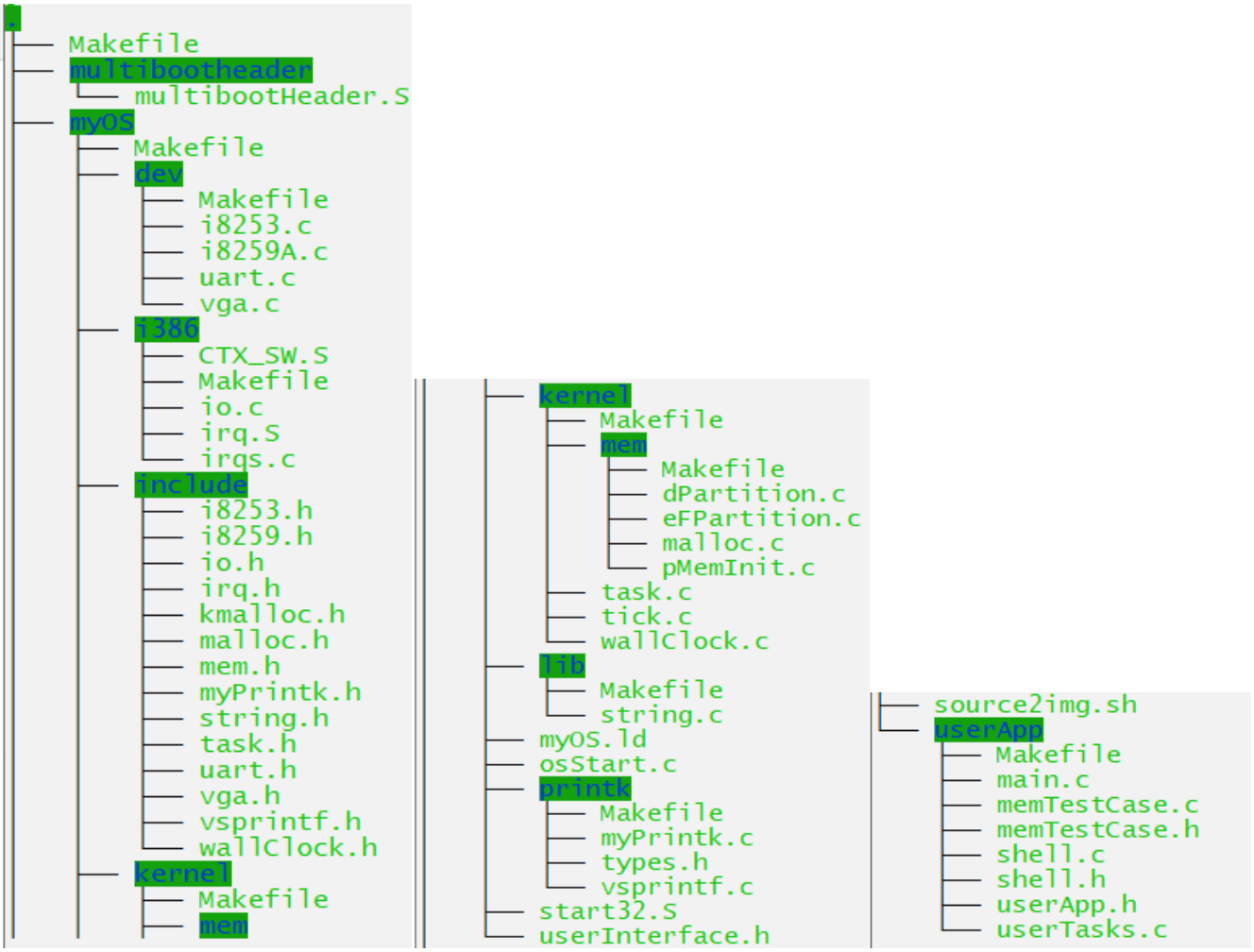
```
void TaskManagerInit(void) {

    int i;
    myTCB* thisTCB;

    for (i = 0;i < TASK_NUM; i++) {
        thisTCB = &tcbPool[i];
        thisTCB->tcbIndex = i;
        thisTCB->state = TSK_FREE;       //任务块都空闲
        if (i == TASK_NUM - 1) thisTCB->next = (myTCB*)0;
        else thisTCB->next = &tcbPool[i + 1];
        thisTCB->stkTop = thisTCB->stack + STACK_SIZE - 1;
    }
    idleTsk = &tcbPool[0];
    stack_init(&(idleTsk->stkTop), tskIdleBdy);
    rqFCFSInit(idleTsk);
    firstFreeTsk = &tcbPool[1];
    createTsk(initTskBody);
    myPrintk(0x2, "start Multitask.....");
    startMultitask();
    myPrintk(0x2, "stop Multitask.....");

}
```

流程图：



四、 目录组织

所有文件：



Makefile组织：

见上图中的各Makefile文件

五、 代码布局

由myOS.ld的代码可知，myOS.elf文件中有三个 section：

1. 第一个section为.text，位置从1M处开始，在.text内的分布为8字节对齐，前12字节为魔术，从第16字节开始是代码部分；
   代码结束后16位对齐；
2. 第二个section为.data，位置从.text结束并对齐后开始；
   末尾16位对齐；
3. 第三个section为.bss，位置从.data结末尾对齐后开始；
   末尾16位对齐；
4. .bss结束后是_end，此处是我们可以操作的内存空间的开始，512位对齐；

六、 编译过程说明

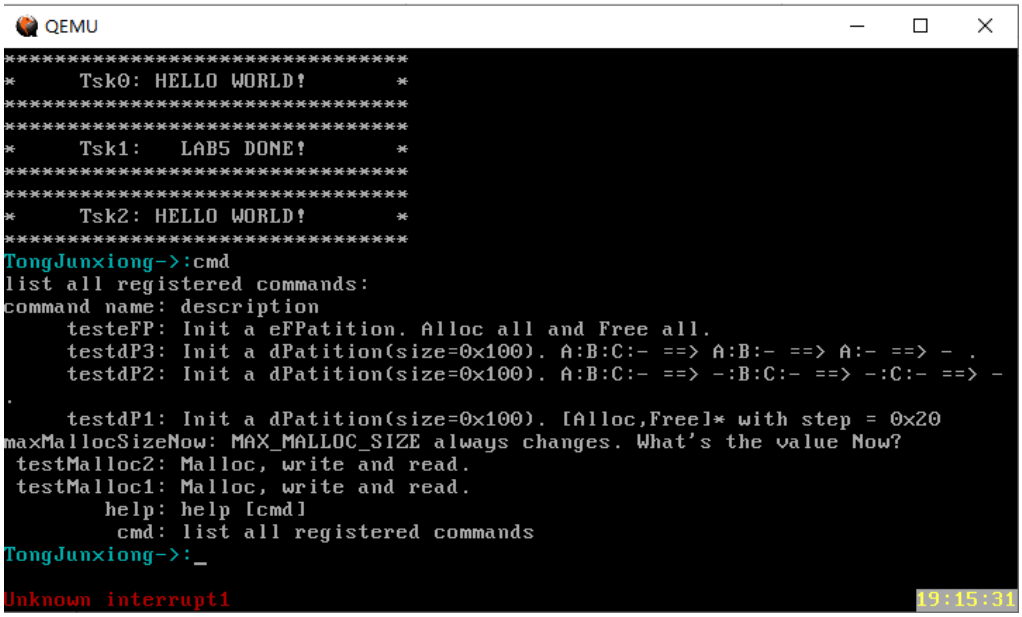
由makefile可知，编译过程有以下两步：

1. 编译汇编代码（header.S和start32.S）和C代码（osstart.c等）生成.o文件；
2. 根据myOS.ld的部署要求，把上述.o文件链接成myOS.elf文件
   如下图所示：

```
ld -n -T myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start32.o output/myOS/osSta
rt.o output/myOS/dev/uart.o output/myOS/dev/vga.o output/myOS/dev/i8253.o output/myOS/dev/i8259A.o out
put/myOS/i386/io.o output/myOS/i386/irq.o output/myOS/i386/irqs.o output/myOS/i386/CTX_SW.o output/myO
S/printk/myPrintk.o output/myOS/lib/string.o output/myOS/kernel/tick.o output/myOS/kernel/wallClock.o
output/myOS/kernel/task.o output/myOS/kernel/mem/pMemInit.o output/myOS/kernel/mem/dPartition.o output
/myOS/kernel/mem/eFPartition.o output/myOS/kernel/mem/malloc.o output/userApp/main.o output/userApp/sh
ell.o output/userApp/memTestCase.o output/userApp/userTasks.o -o output/myOS.elf
make succeed
```

七、 运行和运行结果说明

运行指令qemu-system-i386 -kernel output/myOS.elf -serial stdio
运行指令sudo screen /dev/pts/0
运行结果如下如所示：



从图中可以看出测试用的任务0、1、2均正常执行，shell作为任务也正常执行，各任务的顺序符合FCFS规则

八、 遇到的问题和解决办法

问题：对myTCB的结构里stack项的处理不太明白；
解决：仔细阅读老师的代码，反复尝试，最终明确其为stack数组的首地址。