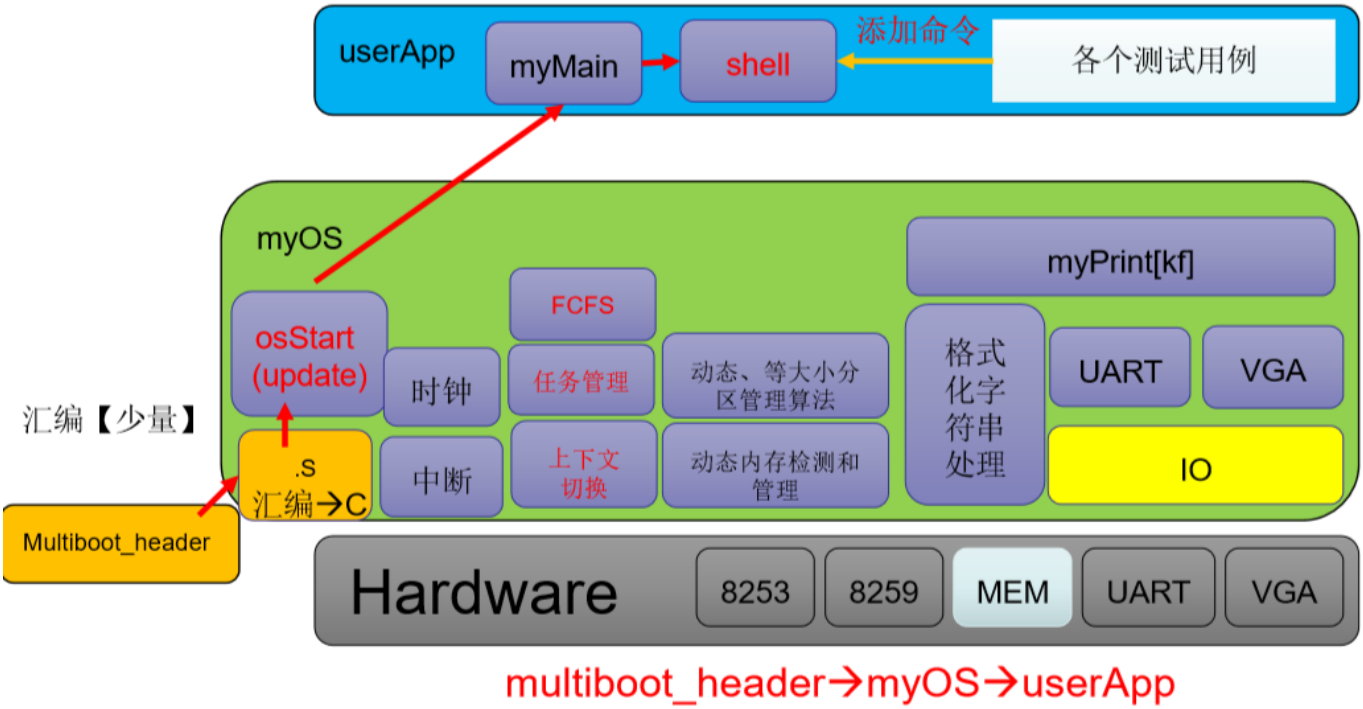
操作系统实验报告

实验六　Scheduler

学号：PB18111683
姓名：童俊雄
完成时间：2020-06-23

一、　软件框图



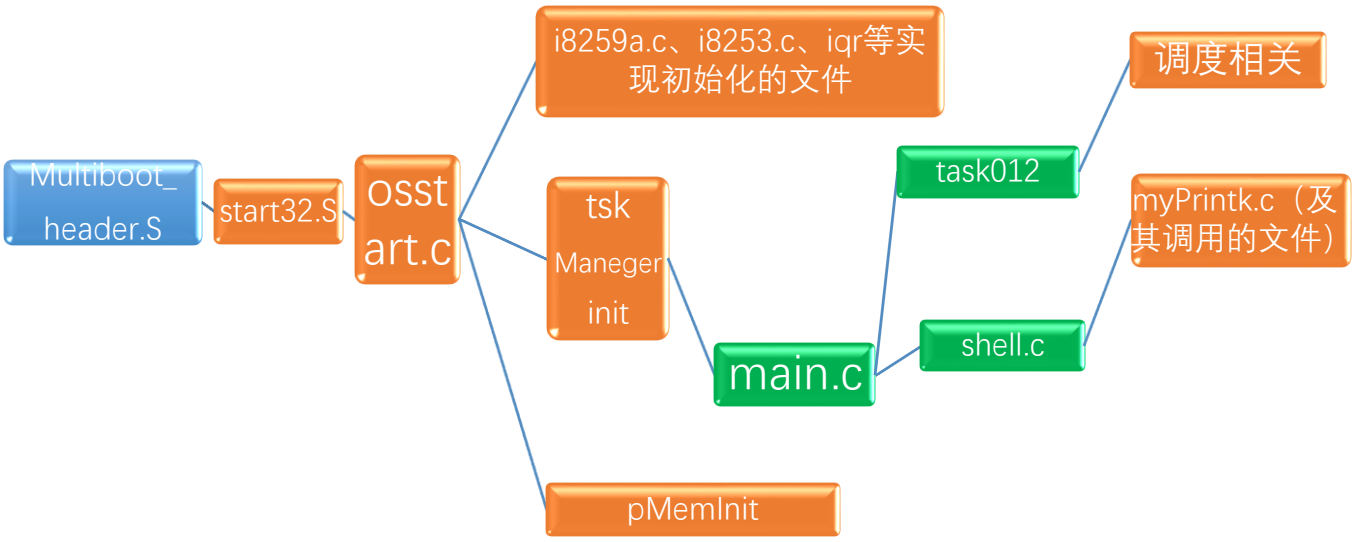
概述：软件大体上可以分为两个层次：用户程序和操作系统，其中操作系统又可以分为与用户程序的接口、IO设备的驱动程序、中断控制程序、时钟功能程序、调度程序和内存管理程序等各功能模块，在每块中又可以进一步划分更细的层次（越上方的层次越高），如图所示。

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）；
7. 通过调度算法执行任务

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. 通用的任务管理模块

TCB&para结构体

```
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;

typedef struct tskPara
{
    unsigned int priority;
    unsigned int exeTime;//为了简化起见,SJF直接将其作为剩余执行时间
    unsigned int arrTime;
    unsigned int schedPolicy;
} tskPara;
```

Para各个函数原理较为简单，在此略去

双向链表入队函数

```c
void ArrListEnqueue(myTCB* tsk)
{
    arrNode* arrnode = tcb2Arr(tsk);
    arrnode->arrTime = tsk->para.arrTime;
    arrnode->theTCB = tsk;
    dLinkedList* head = &arrList;
    dLinkedList* findnode = &arrList;
    arrNode* arr;
    findnode = findnode->next;
    arr = (arrNode*)findnode;
    while (findnode != head)//若是链表的队尾则结束循环
    {
        if (arrnode->arrTime <= arr->arrTime)
        {
            dLinkInsertBefore(head, findnode, (dLinkedList*)arrnode);
            return;
        }
        findnode = findnode->next;//指针后移
        arr = (arrNode*)findnode;
    }
    dLinkInsertBefore(head, findnode, (dLinkedList*)arrnode);
}
```

schedule模块：

```c
void schedule(void)
{
    static int idle_times = 0;
    myTCB* prevTsk, * nextTsk;
    disable_interrupt();

    prevTsk = currentTsk;
    nextTsk = sysScheduler->nextTsk_func();
    currentTsk = nextTsk;
    context_switch(prevTsk, nextTsk);

    enable_interrupt();
}
```

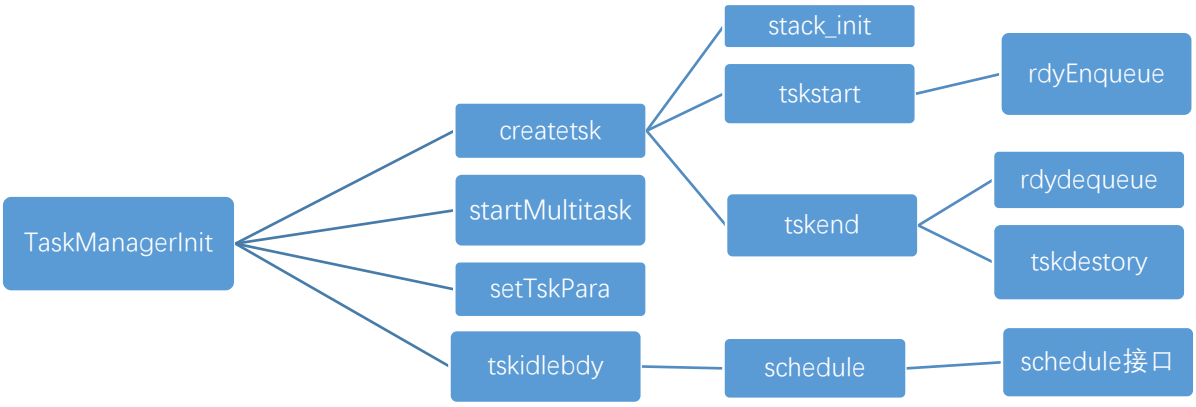
nexttsk等各个函数实现方法类似，故只给出前两项

```c
//实现以下的nextTsk enqueueTsk dequeueTsk schedulerInit scheduler_tick
myTCB* nextTsk(void)
{
    if (sysScheduler->nextTsk_func)
        return sysScheduler->nextTsk_func();
    else
        return (myTCB*)NULL;
}

void enqueueTsk(myTCB* tsk)
{
    if (sysScheduler->enqueueTsk_func)
        sysScheduler->enqueueTsk_func(tsk);
}
```

任务的各个处理函数上次已经给出，详情参见task.c

流程图：



2. SJF调度

（此处的“短作业”做了简化，指的是exe时间较短的作业）

```
myTCB* nextSJFTsk(void)
{
    dLinkedList* head = (dLinkedList*)rqSJF;
    dLinkedList* findnode = head;
    myTCB* nexttsk = (myTCB*)head;
    int i = 0;
    while ((findnode != head) || (i == 0))//到链表尾部停止
    {
        findnode = findnode->next;//链表指针移动
        if ((((myTCB*)findnode)->para.exeTime < nexttsk->para.exeTime) || (i == 0))//剩余时间变化
        {
            nexttsk = (myTCB*)findnode;
            i++;
        }
    }
    return nexttsk;
}
```

其余类似FCFS

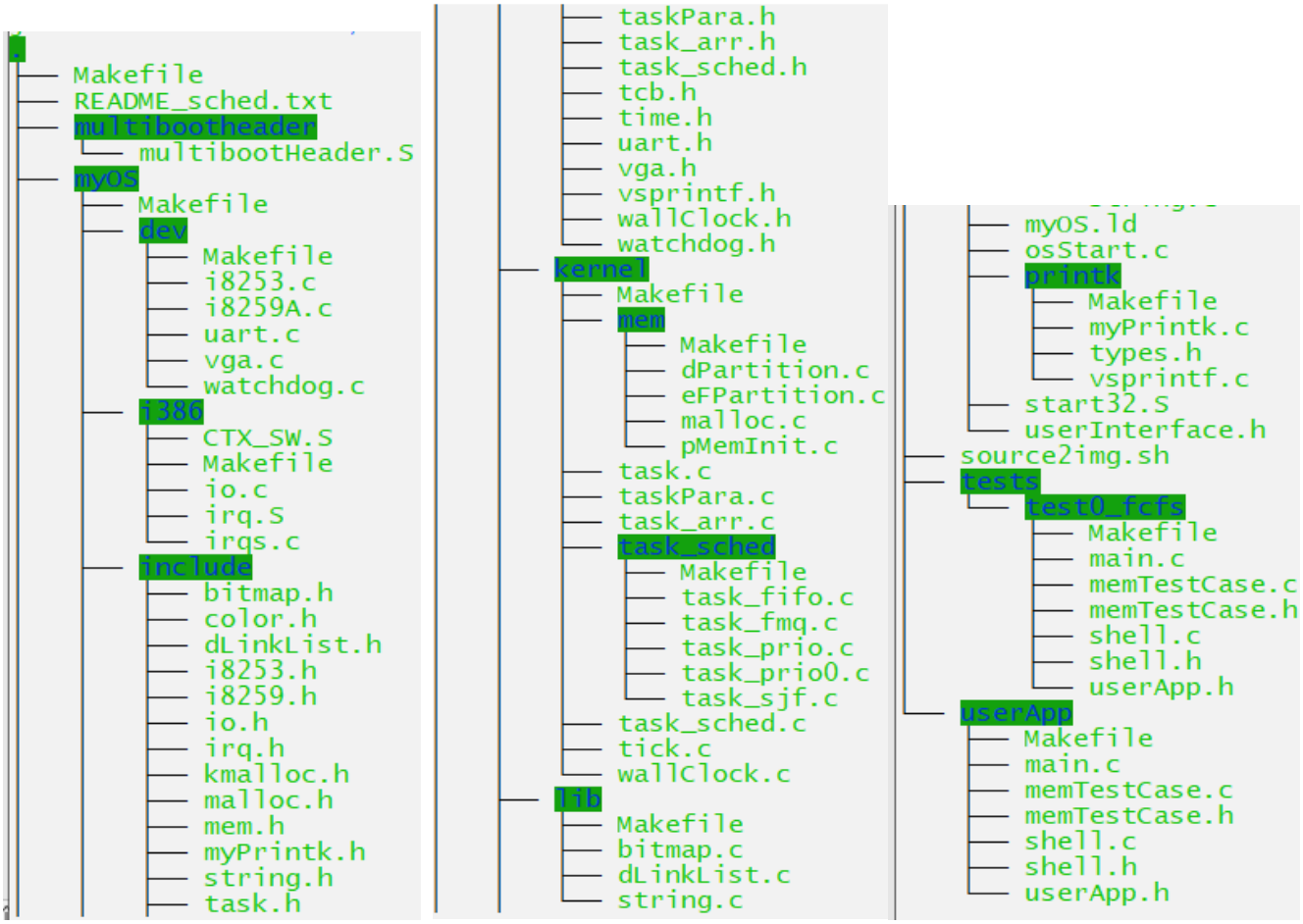
3. PRIO调度

(优先级数字越大，表示优先级越高)

```
myTCB* nextPRIOTsk(void)
{
    dLinkedList* head = (dLinkedList*)rqPRIO;
    dLinkedList* findnode = head;
    myTCB* nexttsk = (myTCB*)head;
    int i = 0;
    while ((findnode != head)||(i==0))//到链表尾部停止
    {
        findnode = findnode->next;//链表指针移动
        if ((((myTCB*)findnode)->para.priority > nexttsk->para.priority) || (i == 0))//指针指向优先级比nexttsk高
        {
            nexttsk = (myTCB*)findnode;//替换
            i++;
        }
    }
    return nexttsk;
}
```

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：

```
rm -rf output
ld -n -T myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start32.o output
/myOS/osStart.o output/myOS/dev/uart.o output/myOS/dev/vga.o output/myOS/dev/i8253.o output
/myOS/dev/i8259A.o output/myOS/i386/io.o output/myOS/i386/irq.o output/myOS/i386/irqs.o out
put/myOS/i386/CTX_SW.o output/myOS/printk/myPrintk.o output/myOS/lib/string.o output/myOS/l
ib/dLinkList.o output/myOS/lib/bitmap.o output/myOS/kernel/tick.o output/myOS/kernel/wallCl
ock.o output/myOS/kernel/task.o output/myOS/kernel/task_arr.o output/myOS/kernel/taskPara.o
 output/myOS/kernel/task_sched.o output/myOS/kernel/mem/pMemInit.o output/myOS/kernel/mem/d
Partition.o output/myOS/kernel/mem/eFPartition.o output/myOS/kernel/mem/malloc.o output/myO
S/kernel/task_sched/task_fifo.o output/myOS/kernel/task_sched/task_fmq.o output/myOS/kernel
/task_sched/task_prio.o output/myOS/kernel/task_sched/task_prio0.o output/myOS/kernel/task_
sched/task_sjf.o output/userApp/main.o output/userApp/shell.o output/userApp/memTestCase.o
-o output/myOS.elf
make succeed
```

七、 运行和运行结果说明

运行指令./source2img.sh

运行指令sudo screen /dev/pts/0

1. 设置为SJF：（修改schedule_hook_main）

```
void scheduler_hook_main(void)
{
    //prior settings
    //setSysScheduler(SCHEDULER_FCFS);
    setSysScheduler(SCHEDULER_SJF);
    //setSysScheduler(SCHEDULER_PRIORITY);
}
```

测试数据：

```
case(SCHEDULER_SJF): {
    myPrintf(0x3, "SJF\n");

    setTskPara(EXETIME, 100, &tskParas[0]);
    createTsk(myTSK0, &tskParas[0]);

    setTskPara(EXETIME, 0, &tskParas[1]);
    createTsk(myTSK1, &tskParas[1]);

    setTskPara(EXETIME, 50, &tskParas[2]);
    createTsk(myTSK2, &tskParas[2]);

    initShell();
    memTestCaseInit();
    setTskPara(EXETIME, 120, &tskParas[3]);
    createTsk(startShell, &tskParas[3]);
}break;
```

结果：

如图，顺利按照设置的exetime顺序执行

2. 设置为PRIO：（修改schedule_hook_main）

```
void scheduler_hook_main(void)
{
    //prior settings
    //setSysScheduler(SCHEDULER_FCFS);
    //setSysScheduler(SCHEDULER_SJF);
    setSysScheduler(SCHEDULER_PRIORITY);
}
```

测试数据：

```
case(SCHEDULER_PRIORITY): {
    myPrintf(0x3, "PRIO\n");
    setTskPara(PRIORITY, 0, &tskParas[0]);
    createTsk(myTSK0, &tskParas[0]);

    setTskPara(PRIORITY, 2, &tskParas[1]);
    createTsk(myTSK1, &tskParas[1]);

    setTskPara(PRIORITY, 4, &tskParas[2]);
    createTsk(myTSK2, &tskParas[2]);

    initShell();
    memTestCaseInit();
    setTskPara(PRIORITY, 1, &tskParas[3]);
    createTsk(startShell, &tskParas[3]);
}break;
```

运行结果：

如图，顺利按照优先级顺序执行（shell结束后才能进入优先级最低的TSK0）

八、 遇到的问题和解决办法

问题：对ArrEnqueue的实现不知道如何解决

解决：请教同学得知，此处需要用到指针的强制类型转换，在dlink_node和arrNode之间进行转换，最终得以解决。