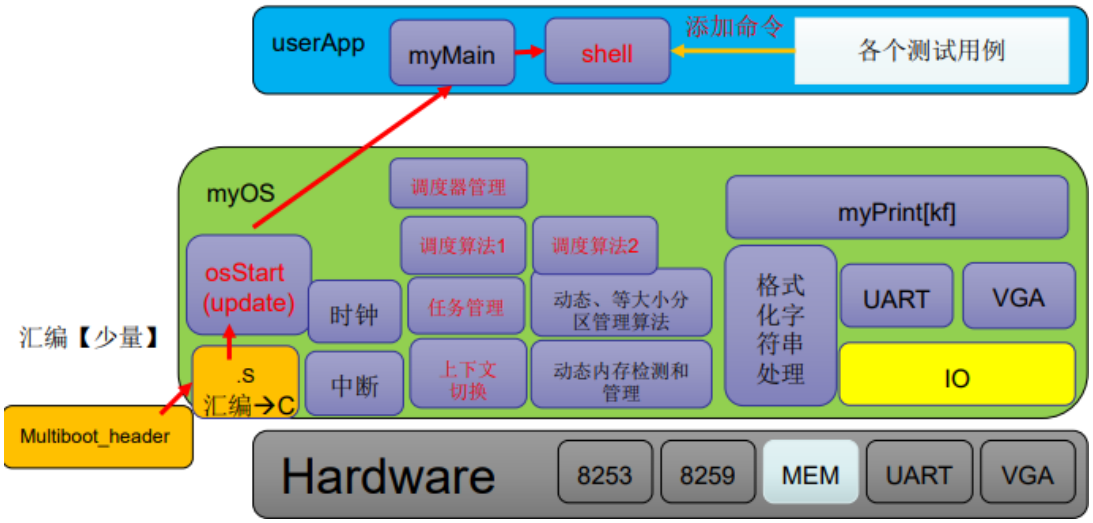
Lab 6

I. 程序框图



与上一个实验相比，本次实验主要拓展了进程调度算法，除了上一个实验中的FCFS算法，本次实验实现了非抢占的优先级调度算法和基于FCFS的时间片轮转算法（使用了时间片方法的调度一定是抢占式调度）。

本次实验中多调度算法的实现是静态的，即OS运行前需要选择使用哪一种调度算法，输入相应数字后运行多进程系统。我的设置中，0对应FCFS调度，1对应非抢占的Priority调度，2对应基于FCFS顺序的RR调度。所谓基于FCFS顺序的RR调度，是指在进程由于时间片用完被抢占时，下一个进程的选择按照FCFS策略。

II. 代码实现

定义调度器结构体并声明一个全局变量：

```
typedef struct scheduler {
	//调度模式: 0-FCFS，1-非抢占Priority，2-(FCFS式的)RR
	int arrangeModel;
	myTCB* (*nextTsk_func)(void);//取下一个进程
	int params[2];//RR模式下分别为时间片长度和当前时间片已经过的长度
	void (*tick_hook)(int);
}scheduler;

scheduler currSch;//当前调度器
```

调度模式的选择实现在启动程序`osStart()`中，它将输入整数作为`model`参数调用`TaskManagerInit()`函数。在准备进入多任务调度模式(`TaskManagerInit()`函数)时，需要调用`setScheduler()`函数根据输入值来初始化调度器。即设置调度器`currSch`的各个字段。之后即可进入多任务模式。

调度时，调度算法使用统一的接口，根据调度器情况来选择不同的算法进行调度。

1.非抢占的Priority调度

该策略需要在进程控制块`myTCB`中增加一个字段`int prority`来表示各个进程的优先级，这里默认每个进程的`prority`各不相同，并且值越小，优先级越高。

在不考虑性能等情况下，简单的实现是直接通过之前的FCFS队列`rqFCFS`来组织就绪队列，每次创建新进程时，它的TCB仍然置于就绪队列末尾，但每次进程执行完需要调度下一个进程时，遍历整个就绪队列，找到优先级最高的那个进程。将该进程设置为当前进程，从就绪队列中删除，再进行上下文切换即可完成一次调度。调度函数如下：

```
//非抢占式优先级调度算法
void schedulePrority(void) {
	myTCB* prevTsk = currentTsk;
	destroyTsk(prevTsk->TSK_ID);//old进程TCB加入空闲TCB链表

	currentTsk = nextProrityTsk();
	tskDequeue(currentTsk);//当前进程离开就绪队列，准备进入CPU

	context_switch(prevTsk);//上下文切换
}
```

2. 使用FCFS的RR调度

时间片轮转需要使用hook机制来实现。在时钟中断的处理程序中使用hook机制：`if(tick_hook) tick_hook(1)`，每次中断都会触发`tick_hook()`函数。在初始化调度器时，非RR策略会将`tick_hook()`置空，而RR策略下则将它设为调度函数，其中参数1指示调度的原因是时间片耗尽，以区分参数为0的调度（进程执行完后的调度）。

调度器结构体中可以设置时间片长度，tick的频率为100Hz，我使用的时间片长为100ms，因此该字段置为10；对应的另一个字段是为了计数，只有计数值到达10才需要进行调度。

另外由于是使用FCFS方法，所以选择下一个进程时也使用FCFS的函数，即直接从就绪队列的开头处取进程控制块。

III. 实验结果

编译运行程序，将终端

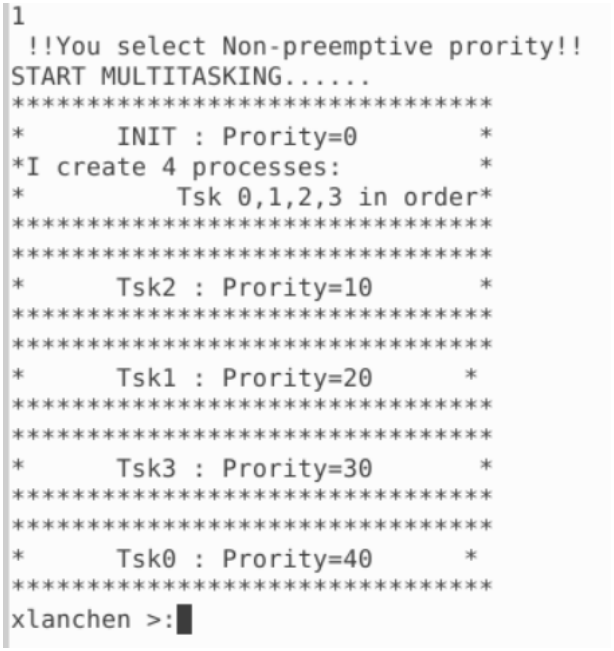
1. 测试FCFS调度

输入0，进入FCFS模式，初始用户进程`myMain()`分别创建4个进程task 0-3，最后创建shell进程，运行结果如下：

可以看到，FCFS策略下，进程严格按照到达顺序来执行，而与进程的优先级无关。

2. 测试非抢占的Priority调度

输入1，进入非抢占的Priority调度模式。进程调度结果如下图：



用户主进程分别创建task 0-3和shell进程（Priority设为100，优先级最低），由上图可以看到进程按照优先级从高到低的顺序执行（task2->task1->task3->0->shell），并且每个进程进入CPU后都是在执行完后才进行调度，这符合非抢占的Priority的调度方法。

3. 测试FCFS的RR调度

输入2，进入FCFS的RR调度。用户主进程分别创建task 0-3，它们执行类似的迭代程序，task 0-3分别迭代10000000，20000000，10000000和10000000次。通过在该调度函数中使用hook机制，注册函数来监测这4个进程的调度情况；在函数内部，通过过程性的输出显示进程执行的进度。进程调度情况如下图：

```
2
 !!You select RR with FCFS!!
START MULTITASKING......
Tast0 get time slice.
       Task0(Priority=40) counts to 0
Tast1 get time slice.
       Task1(Priority=20) counts to 0
       Task1(Priority=20) counts to 5000000
Tast2 get time slice.
       Task2(Priority=10) counts to 0
       Task2(Priority=10) counts to 5000000
Tast3 get time slice.
       Task3(Priority=30) counts to 0
       Task3(Priority=30) counts to 5000000
Tast0 get time slice.
       Task0(Priority=40) counts to 5000000
Tast1 get time slice.
Tast2 get time slice.
Tast3 get time slice.
Tast0 get time slice.
Tast1 get time slice.
       Task1(Priority=20) counts to 10000000
Tast2 get time slice.
       Task2(Priority=10) counts to 10000000
       Task2 finishes.
Tast3 get time slice.
       Task3(Priority=30) counts to 10000000
       Task3 finishes.
Tast0 get time slice.
       Task0(Priority=40) counts to 10000000
       Task0 finishes.
Tast1 get time slice.
       Task1(Priority=20) counts to 15000000
       Task1(Priority=20) counts to 20000000
       Task1 finishes.
```

上述图片显示Task 0-3按照顺序分别获取时间片，交替运行，时间片用完后切换到下一个进程，如此往复，直到进程执行完。最后由于只剩下Task 1，它就一直持有时间片直到完成执行。