**Report for Lab06**

**Stu：金泽文**

**No.PB15111604**

**实验目的：**

通过实现myTCB数据结构，createTsk、destroyTsk原语，以及上下文切换，学习并掌握操作系统任务的概念，简单的任务调度，为后续的进一步实现操作系统做准备。

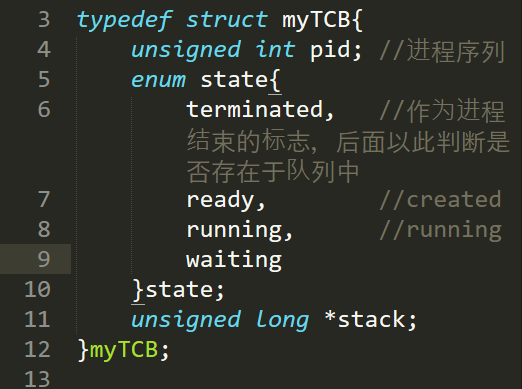
**实验内容：**

1. 设计任务数据结构，命名为myTCB
2. 实现任务创建原语（接口命名为createTsk()）和销毁原语（接口命名为destroyTsk()）功能
3. 实现任务上下文切换（接口命名为CTX\_SW()）功能
4. 实现FCFS调度算法，调度接口命名为schedule()
5. 实现简单的就绪队列和入列/出列原语
6. 实现任务启动原语，接口命名为tskStart()
7. 实现任务终止原语，接口命名为tskEnd()
8. 创建idle任务，idle任务的主体是一个死循环，在循环体中，调用schedule
9. 创建init任务，init任务的主体（接口命名为initTskBody()）由测试用例提供。
10. 实现osStart原语

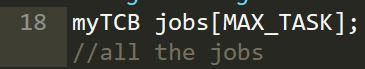
**实验分析与实验过程：**

本次实验重点在于实现。

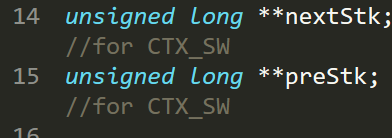
**数据结构**的设计：

**myTCB**:

需要说明的是，每个myTCB对应的stack只是一个指针，而不是一个数组。这里的实现是在内存中分配相应的空间给每个进程的stack。

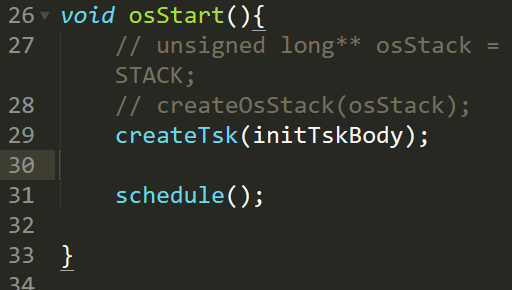
**jobs[MAX\_TASK]：**

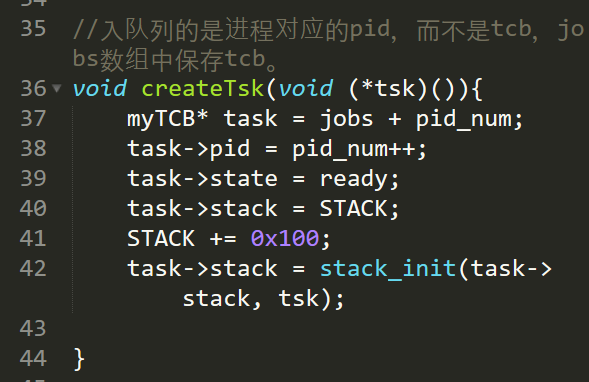
我的进程“队列”使用数组实现的。所有的create，end都通过操作数组的元素实现。

**nextStk，preStk:**

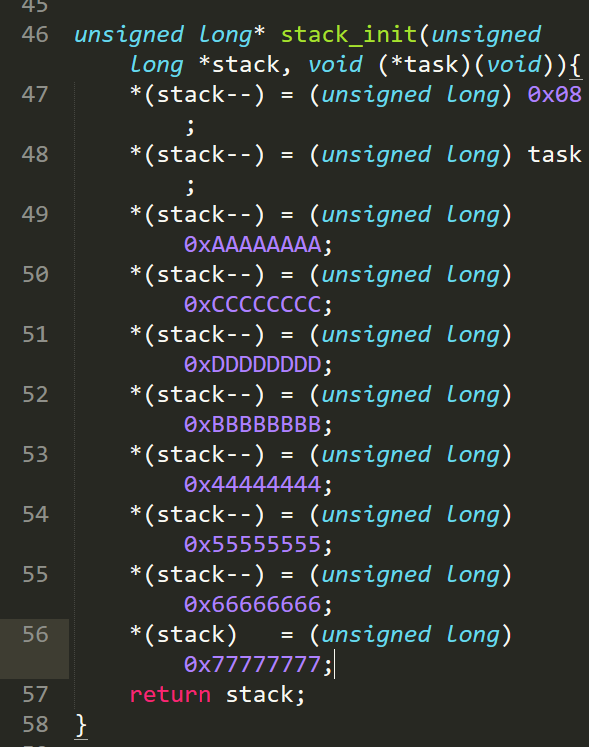
一方面，为了在汇编代码中直接使用nextStk和preStk变量，所以声明为全局变量。另一方面，由于之后initStack操作需要的是“指针的指针”类型，所以如此声明。

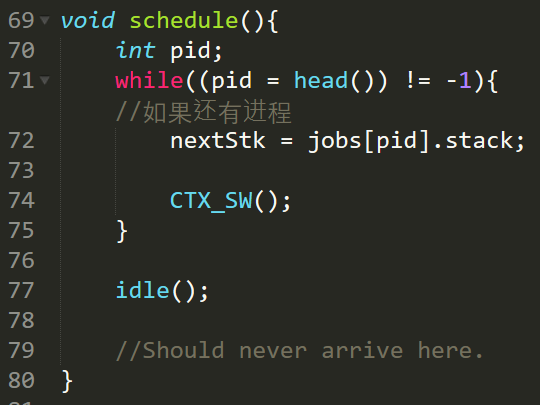
**函数原语**的设计：

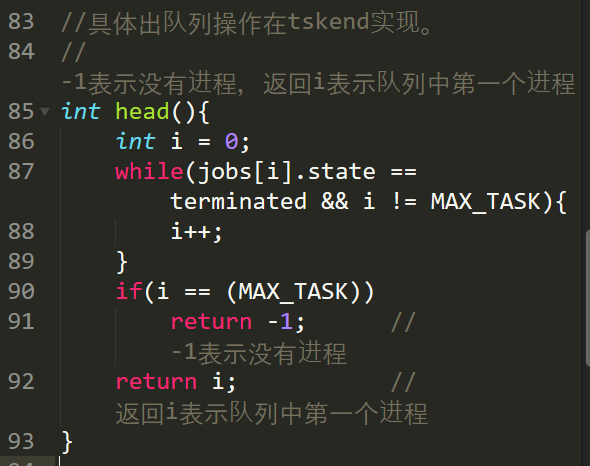
**osStart()**：作为操作系统的入口，create initTskBody，之后调用schedule（）

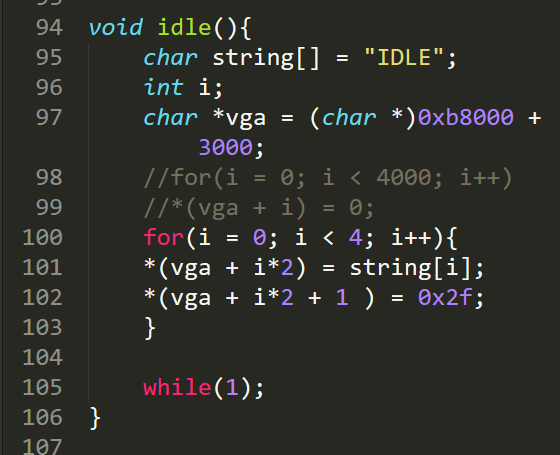
createTsk()：作为创建任务的原语，参数为所调用的函数地址。创建任务时在jobs数组中赋值，通过stack\_init初始化栈。在jobs中辨别任务的标志是进程对应的pid，而不是tcb本身。pid作为jobs数组的下标，分辨进程。创建时要将state置为ready，全局栈需要加0x100.

**stack\_init()**:作为初始化栈的原语。代码基本与ppt中相同。不同之处在于函数返回值，与stack栈顶类型，我这里是unsigned long \*，而ppt中是unsigned long \*\*，由于我对栈的操作的实现不同，所以不需要设置\*\*，而只需要\*。\*\*的目的是为了直接改变\*stack，而我的返回值为被修改的stack本身（作为形参），而在调用时赋值给了对应的stack，所以可以不通过\*\*实现。

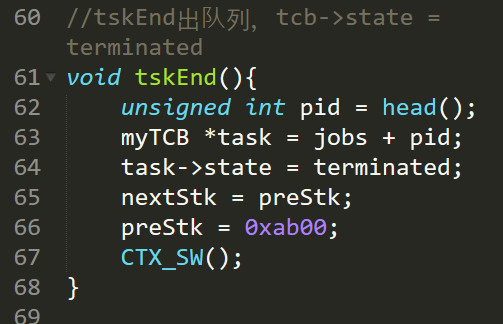


**schedule()**：作为调度函数，在create initTaskBody之后被执行。找到jobs队列中的第一个ready进程之后通过CTX\_SW切换到对应进程。该进程结束之后回到while里面。

**head()**：用来检查并返回jobs队列中的ready进程。通过检查jobs数组中元素的state变量是否为ready或者termintated来判断进程。返回-1表示没有ready进程，否则返回对应ready进程号。

**Idle()**：作为空闲函数，输出“idle”字符串之后便进入死循环。（按理说应该循环检查队

列，但本次实验没有必要，所以直接死循环。）

**tskEnd()**：作为销毁进程原语。本次实验是被进程自己调用（2333）。任务是将对应进程state改为termintated，并且通过CTX\_SW切换到调用该进程的schedule中的对应while位置。

**流程：**

而对于schedule中的while部分：