# 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**:

```
3  typedef struct myTCB{
4      unsigned int pid; //进程序列
5      enum state{
6          terminated,   //作为进程
           结束的标志，后面以此判断是
           否存在于队列中
7          ready,        //created
8          running,      //running
9          waiting
10     }state;
11     unsigned long *stack;
12 }myTCB;
13
```

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]：**

```
18 myTCB jobs[MAX_TASK];
   //all the jobs
```

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:**

```
14 unsigned long **nextStk;
   //for CTX_SW
15 unsigned long **preStk;
   //for CTX_SW
16
```

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（ ）

```
26 void osStart(){
27     // unsigned long** osStack =
       STACK;
28     // createOsStack(osStack);
29     createTsk(initTskBody);
30
31     schedule();
32
33 }
34
```

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.

```
35 //入队列的是进程对应的pid，而不是tcb，jo
   bs数组中保存tcb。
36 void createTsk(void (*tsk)()){
37     myTCB* task = jobs + pid_num;
38     task->pid = pid_num++;
39     task->state = ready;
40     task->stack = STACK;
41     STACK += 0x100;
42     task->stack = stack_init(task->
           stack, tsk);
43
44 }
```

**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。

```
45
46  unsigned long* stack_init(unsigned
        long *stack, void (*task)(void)){
47      *(stack--) = (unsigned long) 0x08
            ;
48      *(stack--) = (unsigned long) task
            ;
49      *(stack--) = (unsigned long)
            0xAAAAAAAA;
50      *(stack--) = (unsigned long)
            0xCCCCCCCC;
51      *(stack--) = (unsigned long)
            0xDDDDDDDD;
52      *(stack--) = (unsigned long)
            0xBBBBBBBB;
53      *(stack--) = (unsigned long)
            0x44444444;
54      *(stack--) = (unsigned long)
            0x55555555;
55      *(stack--) = (unsigned long)
            0x66666666;
56      *(stack)   = (unsigned long)
            0x77777777;
57      return stack;
58  }
```

**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。

```
void schedule(){
    int pid;
    while((pid = head()) != -1){
    //如果还有进程
        nextStk = jobs[pid].stack;

        CTX_SW();
    }

    idle();

    //Should never arrive here.
}
```

**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。

```
//具体出队列操作在tskend实现。
//
-1表示没有进程，返回i表示队列中第一个进程
int head(){
    int i = 0;
    while(jobs[i].state ==
        terminated && i != MAX_TASK){
        i++;
    }
    if(i == (MAX_TASK))
        return -1;      //
        -1表示没有进程
    return i;           //
    返回i表示队列中第一个进程
}
```

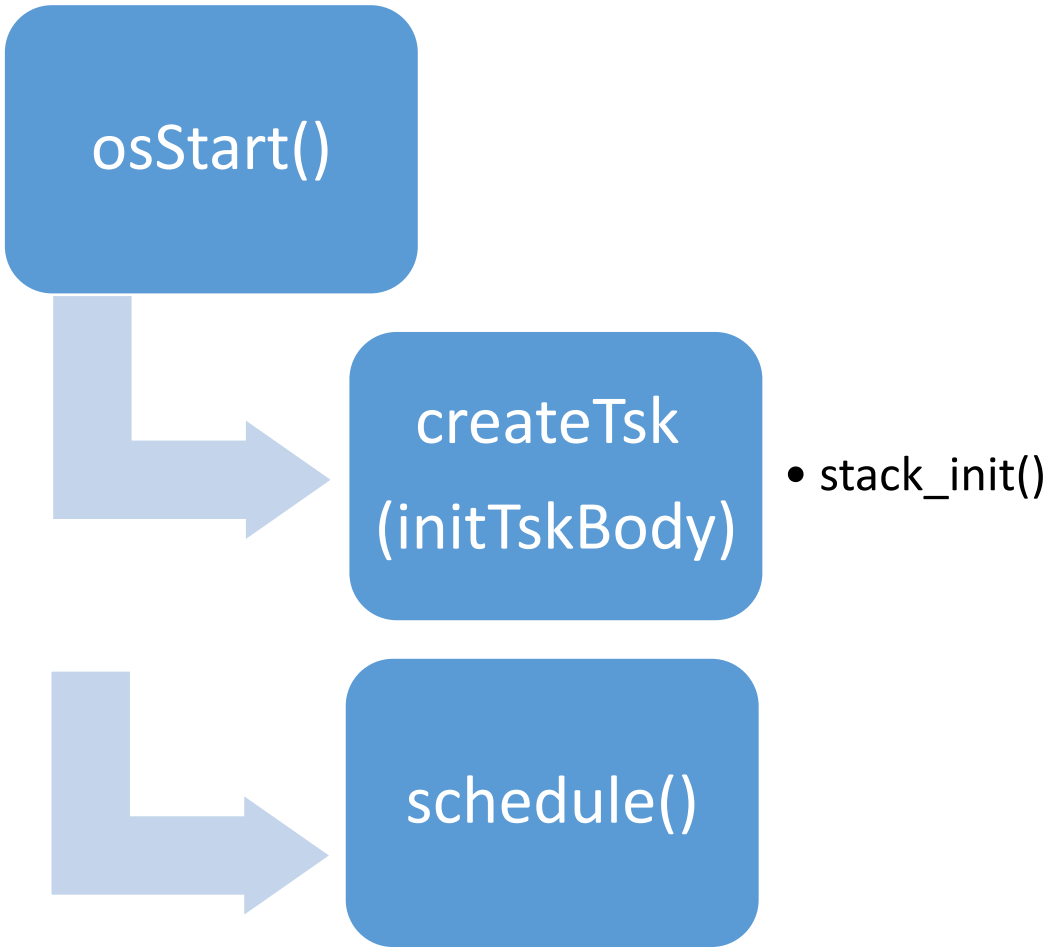
**Idle()**：作为空闲函数，输出“idle”字符串之后便进入死循环。（按理说应该循环检查队列，但本次实验没有必要，所以直接死循环。）

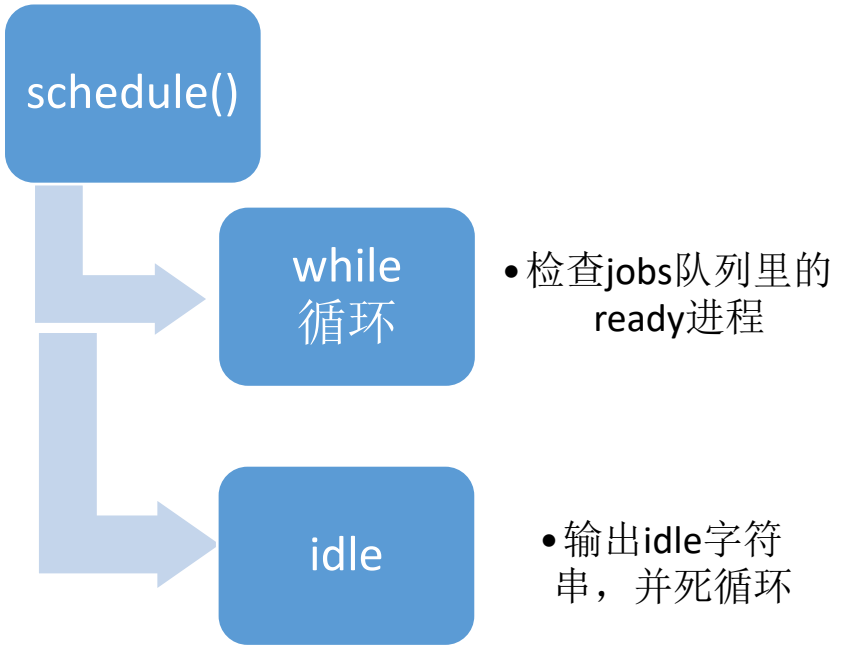
```
94  void idle(){
95      char string[] = "IDLE";
96      int i;
97      char *vga = (char *)0xb8000 +
            3000;
98      //for(i = 0; i < 4000; i++)
99      //*(vga + i) = 0;
100     for(i = 0; i < 4; i++){
101     *(vga + i*2) = string[i];
102     *(vga + i*2 + 1 ) = 0x2f;
103     }
104
105     while(1);
106 }
107
```

**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。

```
60  //tskEnd出队列, tcb->state =
    terminated
61  void tskEnd(){
62      unsigned int pid = head();
63      myTCB *task = jobs + pid;
64      task->state = terminated;
65      nextStk = preStk;
66      preStk = 0xab00;
67      CTX_SW();
68  }
69
```

**流程：**


osStart()
createTsk
(initTskBody)
• stack_init()
schedule()


schedule()
while
循环
• 检查jobs队列里的
ready进程
idle
• 输出idle字符
串，并死循环

而对于 schedule 中的 while 部分：