操作系统实习Lab1实习报告

Exercise 1

- Linux中的进程控制块定义在/usr/src/..../sched.h头文件中的结构体`task_struct{}`,其中定义了大量进程相关的变量，其中比较典型的变量包括例如，进程是否在CPU上的`on_cpu`，代表CPU编号的`cpu`,栈指针`stack`,结束状态`exit_code`,标识进程的`pid,tgid`等等，也保存了进程相关的创建链关系，打开文件表的指针`files`,以及相关的虚拟内存页表信息，保存信息非常全面，
- 在windows中，进程控制块以结构体EPROCESS表示。
- 在Nachos系统中，最基础的进程控制块以 thread.h 文件中的`Thread`类来表示。其中饱含了相对基本的进程管理信息，例如栈指针，进程寄存器上下文，以及进程名称。

Exercise 2

Nachos现有的线程机制和linux的线程机制是有极大的不同的，其线程机制的运行方式是在`main`函数中，调用`ThreadTest()`函数，函数会调用`ThreadTest1()`生成线程并且传入一个函数地址`SimpleThread()`，线程初始化完毕之后，会调用`SimpleTread()`函数作为运行入口。相较于liux的装载操作，Nachos系统由于没有虚拟内存机制，只能够将需要运行的进程编译到ThreadTest.cc中，这样才能够使用其入口函数的地址。同时，其线程切换机制其本质是将Nachos主线程当作CPU，用户线程轮流占用CPU（即当前主线程），在为实现中断机制的时候，CPU只能由用户线程主动放弃。

Exercise 3 & Exercise 4

增加两个变量`userID, tid`定义在头文件`thread.h`中的`Thread`结构体中，并添加了访问函数`get`，如下：

```
    int getTid(){ return tid;}
    int getUserID(){ return userID;}
/*  id管理  */
    int tid;
    int userID;
```

- 其中变量`UserID`由于当前Nachos没有用户管理机制，因此统一初始化为0，即为root用户，但是同样提供了设置`userID`的构造函数`Thread(char*`

`debugName, int userID);`

- 变量`tid`代表线程id，期值由线程创建时申请，操作系统分配，因此其对用户应该是只读属性（此处不考虑用户可以指定线程id的情况）。同时，考虑到操作系统对线程数量上限的限制，因此设立线程池，所有生命周期内的线程均在线程池内，线程池满，则不可在分配线程id，当线程被销毁时，从线程池移除，并返还可用的tid。
- 考虑到在线程调度过程中对线程池的频繁访问，因此在`scheduler.h`中定义线程池为可变数组`threadPool`，并且使用队列`tids`保存可用tid。在`scheduler`实例化时，初始化所有tid均为可用，这一步在`Schedule()`构造函数中实现。
- 同时在`scheduler.cc`中实现函数`acquireTid(Thread* t),releaseTid(Thread* t)`两个函数，用于在线程创建和销毁时维护线程池和tid队列

```
int Scheduler::aquireTid(Thread* t){

    if(tids.size() == 0){
        return -1;
    }
    int tid = tids.front();
    tids.pop();
    threadPool.push_back(t);
    return tid;
}

/**/
void
Scheduler::releaseTid(Thread* t){
    if(t->getTid() == -1){
        return ;
    }
    tids.push(t->getTid());
    for(std::vector<Thread*>::iterator t0 = threadPool.begin(); t0
!= threadPool.end(); t0 ++){
        if(*t0 == t){
            threadPool.erase(t0);
        }
    }
}
```

- 为了配合`scheduler`对于线程池的维护，没次创建线程时须向`scheduler`申请`tid`，如果`tid`返回值为`-1`，则代表线程池已满，线程没有创建成功，除了主线程创建时不需要做如下检查，剩余用户线程创建时均需要做检查以保证线程创建成功，否则没有创建成功的线程不会被加入线程池，也不会被调度

```
Thread(){
+ tid = scheduler.aquireTid(this);
}
~Thread(){
+ scheduler.releaseTid(this);
}
```

- 打印命令TS则比较简单，只需要遍历整个线程池，打印每个线程的状态，用户，id， 等相关信息即可，但是由于当前没有涉及到shell的部分，因此为对其进行运行测试，但是其可以友简单的遍历完成。