实验项目二：Nachos 进程管理

Note: 每次修改文件后，记得在../code/build.linux 目录下运行 make clean Makefile，再 make，之后再运行./nachos -K 命令。

- **扩展 Nachos 线程管理模式，限制线程的数量为最多 128 个：**

  代码定位：

  code/threads/thread.h

  code/threads/thread.cc

1. 要实现对线程数量的限制，可以重新构造一个判断函数，也可以重载 Thread::Tread（）函数，在这里，我们选择后者。

```
int Thread::threadNum = 0;
//Thread函数重载
Thread::Thread(char* threadName, int p)
{
    if(++threadNum > 128){
        cout<<"最多只能同时创建128个线程！！"<<endl;
        ASSERT(threadNum <= 128);
    }
    cout<<"Create Thread "<<threadNum<<endl;
    name = threadName;
    //priority = p;
    stackTop = NULL;
    stack = NULL;
    status = JUST_CREATED;
   // setPriority(p);
    for (int i = 0; i < MachineStateSize; i++) {
        machineState[i] = NULL;         // not strictly necessary, since
                                        // new thread ignores contents
                                        // of machine registers
    }
    space = NULL;
}
```

2. 对 thread.cc 中的 selfTest 函数进行修改：

```
void
Thread::SelfTest()
{
    DEBUG(dbgThread, "Entering Thread::SelfTest");
//    srand(time(NULL));
    for(int i = 1; i <= 130; i++){
        //int priority = rand() % 7 + 1;
        int priority = 0;
        //cout<<"UserThread:"<<i<<" proirity:"<<priority<<endl;

        Thread *t = new Thread("UserThread", priority);
//        t->Fork((VoidFunctionPtr) SimpleThread, (void *) i);
    }
    kernel->currentThread->Yield();
    SimpleThread(0);
}
```

3. 相应的，在 thread.h 中的 Thread 类里添加变量 threadNum 和 Thread(char*, int)

```
public:
  Thread(char* debugName);              // initialize a Thread
  static int threadNum;
  Thread(char* debugName, int priority);
  ~Thread();                            // deallocate a Thread
```

4. 运行结果：

```
redhat@ubuntu:~/NachOS-4.1/code/build.linux$ ./nachos -K


tests summary: ok:0
Create Thread 1
Create Thread 2
Create Thread 3
Create Thread 4
Create Thread 5
Create Thread 6
Create Thread 7
Create Thread 8
Create Thread 9
```

```
Create Thread 118
Create Thread 119
Create Thread 120
Create Thread 121
Create Thread 122
Create Thread 123
Create Thread 124
Create Thread 125
Create Thread 126
Create Thread 127
Create Thread 128
最多只能同时创建128个线程！！
Assertion failed: line 58 file ../threads/thread.cc
Aborted (core dumped)
```

- **修改扩充 Nachos 的线程调度机制，改为“优先级调度”的抢占式调度**
- **下面仅是提示，具体样例请大家认真完成。**

1. 通过 ppt 的实验提示，我们知道所有的线程最终由 Thread::Yield（）函数终止和寻找下一个线程，让我们来看下 Yield 函数的代码。现成的处理都要调用 scheduler 代码，我们再转去 scheduler.cc 中看。
2. “优先级调度”归根究底就是将线程排序，我们要找的就是 scheduler 中的排序算法部分，其中 Scheduler 构造函数中，原本是初始化一个 List 链表类，在 ReadyToRun 函数中使用了 List 链表类中的 Append 函数，我们在 list.cc 中可以看到这个函数就是使用尾插法，这就是先进先出的调度策略。此时，我们的目的就是要改这两处代码，我们再查一下 list.cc 中代码，其实里面已经有一个 SortedList 模板类，这个类里有一个 Insert 算法，这正是我们想要的，所以我们回头将 List 类换成 SortedList 类，然后将 Append 函数换成 Insert 函数即可。但是要注意，在 Inset 中有一个 Compare 函数，这个函数就是我们应该在 scheduler.cc 自定义的实现优先级比较。如下：

```
static int Scheduler::Compare(Thread *x, Thread *y){
    if(x->getPriority() > y->getPriority())
    return -1;
    else if(x->getPriority() == y->getPriority())
    return 0;
    else
    return 1;
}
```

3. 再改一下任务一中的两个函数 Tread（）和 selfTest()，加入优先级
4. 最后别忘了修改相应的 thread.h 文件，因为又定义了优先级函数 getPriority().