Implementing Priority Scheduler with Round-Robin on Nachos

作者：F0503602 石君霄 5050369043 2008-03-18 ~ 2008-04-17

一、实验目的

1. 了解操作系统内核、虚拟机原理
2. 加深对操作系统线程管理、线程调度的理解
3. 练习C、C++、32位汇编语言编程

二、实验环境

- 计算机硬件：DELL INSPRION 600m
- 主机操作系统：Ubuntu Desktop 7.10 gutsy，gnome
- 编译器：gcc 4.1.3 for i486-linux-gnu，gcc 2.95.2 for decstation-ultrix，coff2noff 1.0
- 集成开发环境：Code::Blocks 8.02
- Nachos C++ 4.1版本

三、简要步骤

1. 准备实验环境，下载和安装必要的软件
2. 阅读资料，了解Nachos原理
3. 修正Nachos源码，使之能编译通过
4. 设计并实现抢占式优先级调度算法
5. 增加一些系统调用，以便测试
6. 设计和制作测试程序
7. 输出并分析测试结果

四、抢占式优先级调度算法

与JAVA版本相比，C++版本的Nachos制作时间较早，自带功能也比较少。首先对Nachos源码及Makefile进行了一定的修正，使它本身能正常编译通过和运行；然后入手实现抢占式优先级调度算法。

㈠编译选项

为了方便的看出哪些代码是修改过的，我使用了#ifdef、#ifndef预处理分支，并在Makefile的DEFINES末尾加上-DPRIORITY_SCHED。
例如：

```
  private:
    #ifndef PRIORITY_SCHED
    List<Thread *> *readyList;  // queue of threads that are ready to run,
but not running
    #endif
    #ifdef PRIORITY_SCHED
    SchedulerRoundRobin* schedulerRoundRobin;
    Timer* timerRoundRobin;

    #endif
```

㈡主要数据结构

```
class Thread {
    int priority;
    bool just_run;
}
```

- priority：优先级数值，0表示优先级最高，255表示优先级最低
- just_run：线程是否刚刚启动（在刚才的时间片内开始执行），如果是，暂不剥夺CPU

```
class Scheduler {
    SortedList<Thread*>* readyList;
    List<Thread*>* allThreadsList;
    SchedulerRoundRobin* schedulerRoundRobin;
    Timer* timerRoundRobin;
}
```

- readyList：从原来的List改成了SortedList，排序依据为priority升序排列；这样readyList中最前面的就是优先级最高的READY线程
- allThreadsList：所有线程的一个列表
- schedulerRoundRobin：定时执行，用于时间片调度
- timerRoundRobin：调用schedulerRoundRobin的定时器

```
class SchedulerRoundRobin : public CallBackObj;
```

- 用于时间片调度的类

㈢优先级分配策略

1. 在Thread::Fork中，线程的priority置初值128
2. 当线程用完一个完整的时间片时（由just_run确保），优先级降低（priority加1）
   ⑴如果有优先级相等或更高的线程，马上剥夺当前线程的CPU
   ⑵如果所有其他线程优先级都比当前线程低，继续执行当前线程
3. 线程用完时间片之前主动放弃CPU（Yield）或阻塞，priority不变
4. 有线程READY时，如果该线程比当前线程优先级高，立即剥夺当前线程CPU、

priority不变

㈣优先级分配策略的函数实现

(1)

首先禁用可能干扰优先级调度的其他调度算法。具体的说，就是在Alarm::CallBack函数开头直接return。Alarm类是原先自带的随机时间片轮转调度算法。

(2)

线程的priority等变量必须初始化，创建的线程也必须插入kernel->scheduler->allThreadsList链表中，应当选择一个合适的时机进行这个操作。这个操作看起来可以在Thread::Thread构造函数中执行，实际上是不行的，因为Kernel::Initialize先创建main线程然后才会创建scheduler，所以这两个操作应当写在Thread::Fork中；那样又有一个问题，就是main线程是没有调用Fork的，main函数的初始化操作就由Kernel::Initialize负责。

```
void
Thread::Fork(VoidFunctionPtr func, void *arg)
{
    Interrupt *interrupt = kernel->interrupt;
    Scheduler *scheduler = kernel->scheduler;
    IntStatus oldLevel;
    DEBUG(dbgThread, "Forking thread: " << name << " f(a): " << (int) func <<
" " << arg);
    StackAllocate(func, arg);
    #ifdef PRIORITY_SCHED
    kernel->scheduler->allThreadsList->Append(this);
    this->priority=128;
    #endif
    oldLevel = interrupt->SetLevel(IntOff);
    scheduler->ReadyToRun(this);    // ReadyToRun assumes that interrupts are
disabled!
    (void) interrupt->SetLevel(oldLevel);
}
```

```
void
Kernel::Initialize()
{
//some code omitted here
    #ifdef PRIORITY_SCHED
    currentThread->priority=128;
    kernel->scheduler->allThreadsList->Append(currentThread);
    #endif
    interrupt->Enable();
}
```

(3)

Scheduler类新增的几个成员变量，在Scheduler::Scheduler构造函数中初始化，在Scheduler::~Scheduler析构函数中删除。

值得一提的是Scheduler::Scheduler对readyList的初始化：

```
Scheduler::Scheduler()
{
    #ifndef PRIORITY_SCHED
    readyList = new List<Thread *>;
    #endif
    #ifdef PRIORITY_SCHED
    this->readyList=new SortedList<Thread *>(ThreadPriorityCompare);
    this->allThreadsList = new List<Thread *>;
    this->schedulerRoundRobin=new SchedulerRoundRobin();
    this->timerRoundRobin=new Timer(false,this->schedulerRoundRobin);
    #endif
    toBeDestroyed = NULL;
}
```

SortedList构造函数的参数是用于比较的回调函数，根据这里的需要，应该使优先级高（priority小）的线程排在前面，比较函数为：

```
static int ThreadPriorityCompare(Thread* x,Thread* y) {
    return (x->priority) - (y->priority);
}
```

写得很简洁，只需要做个减法，x优先级高是就返回小于0的值了。

(4)

定时器回调，这个比较复杂，下面详细介绍：

```
class SchedulerRoundRobin : public CallBackObj {
  public:
    void CallBack();
  private:
    int counter;
};
void SchedulerRoundRobin::CallBack() {
```

如果当前处于空闲状态，什么也不做；本来要输出调试信息的，但是这个信息在所有用户线程退出、只剩postal worker时不断出现，所以注释掉了

```
    if (kernel->interrupt->getStatus()==IdleMode) {
        //DEBUG(dbgThread,"Machine is IDLE now, SchdulerRoundRobin do
nothing.");
        return;
    }
```

默认情况下，Timer中断的频率较高，导致线程切换过于频繁；设置一个计数器，把频率

降低20倍

```
    ++this->counter;
    if (this->counter>=20) this->counter=0;
    else return;//don't do round-robin every time
```

当前线程没有执行满一整个时间片，再给一个时间片

```
    if (kernel->currentThread->just_run) {//current thread just run
        kernel->currentThread->just_run=false;//not just run next time
        DEBUG(dbgThread,"Thread "<<kernel->currentThread->getName()<<" just
run, no round-robin");
        return;//give it a chance
    }
```

当前线程用完了时间片，需要增大priority

为了防止溢出，在priority已经达到255时，把所有线程的priority都乘以0.75

```
    if (kernel->currentThread->priority>=255) {//fade all priority numbers to
avoid overflow
        kernel->scheduler->allThreadsList->Apply(ThreadPriorityFade);
        DEBUG(dbgThread,"Priority reach 255, fade to 3/4 of original");
    }
```

现在增加priority值，即降低优先级

```
    ++kernel->currentThread->priority;//current thread used up its time
    DEBUG(dbgThread,"Thread "<<kernel->currentThread->getName()<<" used up
its time, inc priority to "<<kernel->currentThread->priority);
```

判断是否其他线程优先级都低于当前线程

```
    if (kernel->currentThread->priority < kernel->scheduler->readyList-
>Front()->priority) {
```

其他线程优先级仍然都低于当前线程，让当前线程继续执行

```
        DEBUG(dbgThread,"Current thread has highest priority, go on.");
    } else {
```

出现了优先级较高或相等的其他线程，剥夺CPU

```
        DEBUG(dbgThread,"Current thread does not have highest priority,
yield.");
        kernel->interrupt->YieldOnReturn();//pass chance to another thread
    }
}
```

(5)

Scheduler::ReadyToRun中readyList->Append改为readyList->Insert，因为SortedList::Append是私有的。

ReadyToRun在插入后还需检查新加入线程的优先级是否高于当前线程，如果新线程优先

级高，马上Yield当前线程。

(6)

Scheduler::Run中，在每次开始运行一个线程时，置just_run为true

(7)

更新Scheduler::Print，打印线程表时要打印readyList和allThreadsList，并需要打印出priority，以便调试用。

㈤正确性验证

在现在看，由于还没有有效的创建新线程的手段，算法的正确性只能停留在理论上。

五、增加系统调用

可以用nachos -x xxx.noff启动noff程序，但是noff程序（test/目录内）是不能直接调用nachos内核的函数的。如果需要调用nachos内核提供的功能，唯一的办法是产生MIPS CPU异常，异常被nachos捕获加以处理。产生异常的代码在test/start.s内，给$2置系统调用号然后syscall即可；捕获异常后执行userprog/exception.cc中的ExceptionHandler函数。

㈠设计系统调用类型

在userprog/syscall.h里声明了若干种系统调用，本来只实现了SC_Halt和SC_Add两个。我选择实现了以下几种，就足够用以测试优先调度器：

1. SC_Exit：退出程序，终止nachos
2. SC_Exec：创建用户态进程，在新的地址空间中运行（测试发现AddrSpace功能不足，此系统调用会破坏现有MIPS虚拟机内存，无法正常工作）
3. SC_ThreadFork：创建线程，在同一个地址空间运行
4. SC_ThreadYield：主动放弃CPU
5. SC_ThreadExit：终止线程
6. SC_Clock：取得当前系统时钟tick值
7. SC_SetPriority：（这是新增的）直接设置当前线程的priority值

下面详细介绍两个系统调用的实现，其他的没有太多看点，不做介绍。

㈡SC_Exec的实现

```
SpaceId Exec(char* exec_name);
```

SC_Exec的难点在于如何取得要调用的noff文件名，它是一个字符串类型的参数。

很容易看到已经实现的SC_Add中获取参数的方法：

```
    result = SysAdd((int)kernel->machine->ReadRegister(4),(int)kernel-
>machine->ReadRegister(5));
```

直观的反应是用(char*)kernel->machine->ReadRegister(4)就可以取得这个exec_name字符串参数，这样做的结果就是报Segment fault错误，然后nachos的Linux进程骤然而止。

什么原因呢？$4寄存器里确实存有指向exec_name的指针，也就是字符串的内存地址；但是，这个内存地址是MIPS虚拟机里的，而不是Linux进程的地址空间！因此，直接转换为char*类型，不可能指向正确的内存地址。

解决方法是：用kernel->machine->ReadMem函数逐字节读出内容虚拟机内存中的字符串，直到遇到\0为止。

```
            int SC_Exec_name_addr=(int)kernel->machine-
>ReadRegister(4);//exec_name
            //note: exec_name is stored in MIPS mem, not host mem, must copy
from MIPS to host
            char* SC_Exec_name=new char[256];
            int SC_Exec_ch=-1;
            int SC_Exec_name_i;
            for
(SC_Exec_name_i=0;SC_Exec_ch!=0&&SC_Exec_name_i<256;++SC_Exec_name_i) {
                kernel->machine-
>ReadMem(SC_Exec_name_addr+SC_Exec_name_i,1,&SC_Exec_ch);
                SC_Exec_name[SC_Exec_name_i]=(char)SC_Exec_ch;
            }
            ASSERT(SC_Exec_name_i<256);//exec_name should be less than 256
chars
```

这个系统调用由于AddrSpace类的问题没能正常工作，所以其他细节就不再讲述了。

㈢SC_ThreadFork的实现

nachos的Thread::Fork接受的两个参数是一个Linux函数和给它的参数，并不能直接调用noff中的函数，所以必须用一个Linux函数来启动noff函数：

```
#ifdef PATCH_SYSCALLS1
void NoffThread(int func)
{
    DEBUG(dbgThread,"NoffThread "<<kernel->currentThread->getName()<<" start
at "<<func);
    kernel->machine->WriteRegister(PrevPCReg,func-4);
    kernel->machine->WriteRegister(PCReg,func);
    kernel->machine->WriteRegister(NextPCReg,func+4);
    kernel->machine->Run();
}
#endif
```

SC_ThreadFork就是创建一个新Thread（名称是原Thread名称连接上起始地址），然后用

上述函数调用Fork。

六、程序运行结果

现在我们已经有了一个抢占式优先级调度器，也有了可以创建线程、修改优先级的系统调用，可以测试这个程序了。

㈠测试用例：sunny.c

一个没有实际意义(dummy)的小程序，总共会有4个线程，还有很多纯粹浪费(消耗)时间的for循环。

```
#include "syscall.h"

void t3() {
    int i;
    SetPriority(60);
    for (i=0;i<3000;++i);
    SetPriority(200);
    for (i=0;i<3000;++i);
    ThreadExit(233);
}

void t1() {
    int i;
    ThreadFork((void*)t3);
    for (i=0;i<600;++i) {
        if (i%50==0) ThreadYield();
    }
    ThreadExit(111);
}

void t2() {
    int i;
    for (i=0;i<800;++i) if (i%100==0) Clock();
    ThreadExit(22);
}

void last() {
    int i;
    SetPriority(255);
    ThreadYield();
    for (i=0;i<10000;++i);
    Halt();
}

void main() {
    int i;
    ThreadFork((void*)last);
    for (i=0;i<1000;++i);
    ThreadFork((void*)t1);
    for (i=0;i<1000;++i);
    ThreadFork((void*)t2);
    for (i=0;i<100;++i);
```

```
    ThreadExit(64);
}
```

为了能看到sunny.c对应的汇编指令，还在test/Makefile中增加了一条规则：

```
sunny.a: sunny.c
    $(CC) -S -c $(INCDIR) -c $< -o $@
```

这样只需执行make sunny.a就可以在sunny.a文件中看到汇编指令了，调试时可以作参考。

㈡执行测试用例

为了方便的编译和执行测试用例，创建了几个shell脚本。
工作时只需先cd到test/目录，然后：

- ./build，这个脚本会cd到build.linux目录，执行make，然后cd回来
- ./run，这个脚本根据预设参数执行sunny.noff
- ./runlog，这个脚本根据预设参数执行sunny.noff，把stdout和stderr重定向到output.txt文件里
- make，是一个标准命令，编译测试用例

其中runlog的代码如下：

```
../build.linux/nachos -d tu -x sunny.noff > output.txt 2>&1
```

这是以带调试信息的方式启动nachos并执行sunny.noff测试用例，调试信息选择dbgThread和dbgSys两种

执行一次./runlog，就可以获得输出结果了！

㈢结果分析

查看output.txt中的输出，真的非常……长……长……长啊（请读者注意看我标了颜色的部分，其他一扫而过）：

```
Entering main


tests summary: ok:0
Forking thread: postal worker f(a): 134617820 0x8070690
Putting thread on ready list: postal worker priority=128
########################################
Scheduler readyList contents:
Thread postal worker priority=128 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=READY
########################################
Received Exception 1 type: 12

Thread main fork 748
main函数正在fork线程t1，线程名称是main748
```

```
Forking thread: main748 f(a): 134596188 0x2ec
Putting thread on ready list: main748 priority=128
新线程的优先级总是128
########################################
Scheduler readyList contents:
Thread postal worker priority=128 status=READY
Thread main748 priority=128 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=READY
Thread main priority=128 status=RUNNING
Thread main748 priority=128 status=READY
########################################
Thread main used up its time, inc priority to 129
线程main在循环中用完了它的时间片，优先级降低了
Current thread does not have highest priority, yield.
现在main的优先级129低于main748的优先级128，所以main被剥夺
Yielding thread: main
Putting thread on ready list: main priority=129
########################################
Scheduler readyList contents:
Thread main748 priority=128 status=READY
Thread main priority=129 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=READY
Thread main priority=129 status=READY
Thread main748 priority=128 status=READY
########################################
Switching from: main to: postal worker
Beginning thread: postal worker
Sleeping thread: postal worker
线程被阻塞，优先级不会改变
Switching from: postal worker to: main748
Beginning thread: main748
NoffThread main748 start at 748
Received Exception 1 type: 51

main748 sets its priority from 128 to 255, takes effect on next
scheduler operation
main748使用SC_SetPriority系统调用调整了自己的优先级
Received Exception 1 type: 13

Thread main748 yield
Yielding thread: main748
Putting thread on ready list: main748 priority=255
main748使用SC_ThreadYield主动放弃CPU，优先级不变（仍然是它自己设定的255，没有
增加）
 ########################################
Scheduler readyList contents:
```

```
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
########################################
Switching from: main748 to: main
Now in thread: main
Received Exception 1 type: 12

Thread main fork 480
Forking thread: main480 f(a): 134596188 0x1e0
Putting thread on ready list: main480 priority=128
Newly-ready thread main480(priority128) < running thread
main(129), yield and switch
```
创建的新线程main480优先级128，高于当前main线程优先级129，立即执行调度
```
Yielding thread: main
Putting thread on ready list: main priority=129
########################################
Scheduler readyList contents:
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
```
把main（优先级129）放入readyList列表时，总是排在优先级最低的main748之前
```
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
Thread main480 priority=128 status=READY
########################################
Switching from: main to: main480
Beginning thread: main480
NoffThread main480 start at 480
```
NoffThread就是用Linux函数启动MIPS线程
```
Received Exception 1 type: 12

Thread main480 fork 400
Forking thread: main480400 f(a): 134596188 0x190
Putting thread on ready list: main480400 priority=128
```
当前线程main480的优先级128与新线程main480400的优先级128相同，所以继续执行当前线程
```
########################################
Scheduler readyList contents:
Thread main480400 priority=128 status=READY
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
```

```
Thread main480 priority=128 status=RUNNING
Thread main480400 priority=128 status=READY
########################################
Received Exception 1 type: 13

Thread main480 yield
Yielding thread: main480
Putting thread on ready list: main480 priority=128
########################################
Scheduler readyList contents:
Thread main480 priority=128 status=READY
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
Thread main480 priority=128 status=READY
Thread main480400 priority=128 status=READY
########################################
Switching from: main480 to: main480400
Beginning thread: main480400
NoffThread main480400 start at 400
Received Exception 1 type: 51

main480400 sets its priority from 128 to 60, takes effect on
next scheduler operation
Thread main480400 just run, no round-robin
刚刚开始执行，可能没有用完一整个时间片，所以继续执行
Thread main480400 used up its time, inc priority to 61
Current thread has highest priority, go on.
时间片用完了，降低优先级；但是优先级仍然是最高的，所以继续执行
 Thread main480400 used up its time, inc priority to 62
Current thread has highest priority, go on.
Received Exception 1 type: 51

main480400 sets its priority from 62 to 200, takes effect on
next scheduler operation
Thread main480400 used up its time, inc priority to 201
Current thread does not have highest priority, yield.
Yielding thread: main480400
Putting thread on ready list: main480400 priority=201
########################################
Scheduler readyList contents:
Thread main priority=129 status=READY
Thread main480400 priority=201 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
```

```
Thread main480 priority=128 status=READY
Thread main480400 priority=201 status=READY
########################################
Switching from: main480400 to: main480
Now in thread: main480
Received Exception 1 type: 15

Thread main480 exit with code 233
Finishing thread: main480
Sleeping thread: main480
Switching from: main480 to: main
Now in thread: main
Deleting thread: main480
```
main480执行完毕了（通过SC_ThreadExit系统调用退出），被删除，bye-bye & wave
```
########################################
Scheduler readyList contents:
Thread main480400 priority=201 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=RUNNING
Thread main748 priority=255 status=READY
Thread main480400 priority=201 status=READY
########################################
########################################
Scheduler readyList contents:
Thread main480400 priority=201 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=RUNNING
Thread main748 priority=255 status=READY
Thread main480400 priority=201 status=READY
########################################
Thread main just run, no round-robin
Received Exception 1 type: 12

Thread main fork 620
Forking thread: main620 f(a): 134596188 0x26c
Putting thread on ready list: main620 priority=128
Newly-ready thread main620(priority128) < running thread
main(129), yield and switch
```
这部分和刚才一样的
```
Yielding thread: main
Putting thread on ready list: main priority=129
########################################
Scheduler readyList contents:
Thread main priority=129 status=READY
Thread main480400 priority=201 status=READY
Thread main748 priority=255 status=READY
```
为什么没有后面执行的main620？因为已经被FindNextToRun取走了

```
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
Thread main480400 priority=201 status=READY
Thread main620 priority=128 status=READY
########################################
Switching from: main to: main620
Beginning thread: main620
NoffThread main620 start at 620
Received Exception 1 type: 20

main620 asks current clock 12066
询问时钟的系统调用
Received Exception 1 type: 20

main620 asks current clock 13572
Thread main620 just run, no round-robin
Received Exception 1 type: 20

main620 asks current clock 15078
Thread main620 used up its time, inc priority to 129
Current thread does not have highest priority, yield.
Yielding thread: main620
Putting thread on ready list: main620 priority=129
########################################
Scheduler readyList contents:
Thread main620 priority=129 status=READY
Thread main480400 priority=201 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main748 priority=255 status=READY
Thread main480400 priority=201 status=READY
Thread main620 priority=129 status=READY
########################################
Switching from: main620 to: main
Now in thread: main
########################################
Scheduler readyList contents:
Thread main620 priority=129 status=READY
Thread main480400 priority=201 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=RUNNING
Thread main748 priority=255 status=READY
Thread main480400 priority=201 status=READY
Thread main620 priority=129 status=READY
########################################
Received Exception 1 type: 15

Thread main exit with code 64
Finishing thread: main
```

```
Sleeping thread: main
Switching from: main to: main620
Now in thread: main620
Deleting thread: main
```
又一位朋友离开了我们，wave~
```
########################################
Scheduler readyList contents:
Thread main480400 priority=201 status=READY
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main748 priority=255 status=READY
Thread main480400 priority=201 status=READY
Thread main620 priority=129 status=RUNNING
########################################
Received Exception 1 type: 15

Thread main620 exit with code 64
Finishing thread: main620
Sleeping thread: main620
Switching from: main620 to: main480400
Now in thread: main480400
Deleting thread: main620
########################################
Scheduler readyList contents:
Thread main748 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main748 priority=255 status=READY
Thread main480400 priority=201 status=RUNNING
########################################
Received Exception 1 type: 15

Thread main480400 exit with code 64
Finishing thread: main480400
Sleeping thread: main480400
Switching from: main480400 to: main748
Now in thread: main748
Deleting thread: main480400
########################################
Scheduler readyList contents:
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main748 priority=255 status=RUNNING
########################################
Received Exception 1 type: 1

Thread main748 call Exit with code 0
Machine halting!
```
main748线程要求关闭整个nachos，曲终人尽，see you next time!
```
Ticks: total 16258, idle 0, system 150, user 16108
```
nachos的统计信息，据说system ticks不准的

```
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

上述测试用例的输出结果展现了大部分设计的功能，输出及注释已经明确的体现了本实验设计的正确性。
本测试用例及输出没能体现的特性有：

1. 某个priority达到255并需继续增加时，所有线程的priority乘以0.75
2. 有大量线程时调度器的时间效率
3. 调度器的空间效率

七、开发小结

- nachos是一个简单的“虚拟机+操作系统”，虽然实质上它并不是操作系统，但是很多原理都是相通的
- nachos设计时间较早，有一些兼容性问题，但是很容易解决，也不需要特意去弄什么RedHat9（据说里面的GEdit没有代码高亮~）或gcc2.95
- 网上的中文文档很少，基本上是看英文的，事实上英文文档质量远好于中文文档。认真读读Roadmap和Guide to reading the NACHOS source，然后认真阅读源码（Code::Blocks的搜索功能很有用），从main开始一路跟踪进去，理解了就容易做了

八、参考资料

1. A Road Map Through Nachos 概要性介绍了Nachos源码的各个部分：机器、线程、用户进程、文件系统……
2. Guide to reading the NACHOS source 深入阅读线程调度与启动、系统调用、地址转换、用户态地址空间部分源码