# 实验 进程调度

5090369081 蔡韬

popacai@gmail.com

## 实验内容：

修改Nachos代码，实现以下功能来实现进程调度：

1. 系统调用函数
2. Onetick时间片处理
3. 进程调度算法

## 环境搭建

根据[1]中的文档，修改相关源代码，即可运行。

## 源代码分析

参考[2]中的其中相关调度的内容

Thread提供线程相关函数，例如创建，运行，阻塞等

Schedule提供调度所需要的相关函数，例如等待列表。

Interrupt提供相关中断函数

Userproc提供用户态程序所需要的函数和系统调用

Lib中提供了相关的一些数据结构，例如链表。

## 调度算法说明

在Unix系统中的调度的优先级计算是：

p\_pri = p\_cpu / 2 + p\_nice +PUSER + NZERO

其中p\_pri是动态优先级，p\_cpu是CPU运行的tick，p\_nice是用户之前设置的相关优先级，PUSER和NZERO用于区分不同的优先级。p\_nice越低优先级越高。

在这里，我简化了算法，成为p\_pri = p\_cpu / 2 + p\_nice

其中当前thread的p\_cpu每执行一次则p\_cpu++。50个时钟tick后，全体ready线程p\_cpu减半。每次tick时，则调度其p\_pri最小的线程。而每个程序运行时间片以同比例与优先级区间衍射的方式进行选取。

通过这种实现动态优先级调度。

## 代码实现

**修改userproc中的实现，增加相关的系统调用。**

默认中的系统调用

#define SC\_Halt 0

#define SC\_Exit 1

#define SC\_Exec 2

#define SC\_Join 3

#define SC\_Create 4

#define SC\_Remove 5

#define SC\_Open 6

#define SC\_Read 7

#define SC\_Write 8

#define SC\_Seek 9

#define SC\_Close 10

#define SC\_Delete 11

#define SC\_ThreadFork 12

#define SC\_ThreadYield 13

#define SC\_ExecV 14

#define SC\_ThreadExit 15

#define SC\_ThreadJoin 16

#define SC\_getSpaceID 17

#define SC\_getThreadID 18

#define SC\_Ipc 19

#define SC\_Clock 20

实现其中的Fork函数

case SC\_ThreadFork:

threadname = new char[32];

cout << "pid:" << ++pid ;

cout << " priority?" << endl;

cin >> priority;

sprintf(threadname,"pid=%d nice=%d",pid,priority);

thread = new Thread(threadname);

thread->nice = priority;

thread->cpu = 0;

thread->space= kernel->currentThread->space;

kernel->currentThread->SaveUserState();

kernel->machine->WriteRegister(RetAddrReg, kernel->machine->ReadRegister(PCReg)+4);

kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(4));

kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);

thread->SaveUserState();

kernel->currentThread->RestoreUserState();

kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));

kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);

kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg)+4);

kernel->currentThread->SaveUserState();

thread->Fork(runFunc, NULL);

return;

ASSERTNOTREACHED();

break;

**Thead类中增加相关优先级并修改Yield函数：**

int time; //time\_slice

int pri; //pri

int nice; //nice value static

int cpu; //cpu

int ticks;

其中time用于存储这个线程已经使用的时间TICK，用于计量时间片。当时间片用尽后，进行调度，决定是否继续运行还是切换

Pri是动态优先级参数，越低优先级越高

Nice是静态优先级参数，决定时间片长度和优先级

Cpu为时间片，运行时间越长，数值越大。

Ticks用于计量程序已经运行的tick时间数，主要用于测试数据的现实，与调度无关。

Yield函数修改：

void

Thread::Yield ()

{

Thread \*nextThread;

IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);

ASSERT(this == kernel->currentThread);

DEBUG(dbgThread, "Yielding thread: " << name);

kernel->scheduler->ReadyToRun(this);

nextThread = kernel->scheduler->FindNextToRun();

if (nextThread != NULL) {

kernel->scheduler->Run(nextThread, FALSE);

}

(void) kernel->interrupt->SetLevel(oldLevel);

}

将现有进程也加入到readyList中，从而达到动态优先级下，优先级高的进程在时间片用尽后可能可以继续获得执行权。

**修改OneTick函数，修改调度方式，并增加衰减函数。**

int b\_time\_slice\_out() //check for current threading's time\_slice is out or not

{

int t;

t = kernel->currentThread->time > (30 - kernel->currentThread->nice);

return t; //true for time slice is out

}

void

Interrupt::OneTick()

{

MachineStatus oldStatus = status;

Statistics \*stats = kernel->stats;

ListIterator<Thread\*>\*it = new ListIterator<Thread\*>(kernel->scheduler->readyList);

Thread\* th;

if (!(stats->userTicks % 50))

{

for (;!it->IsDone();it->Next())

{

th = (Thread\*)it->Item();

th->cpu = th->cpu / 2;

cout << th->name << " tick=" << th->ticks << " cpu:" << th->cpu<< endl;

}

cout << "---------------" << endl;

}

if (b\_time\_slice\_out())// start to switch

{

kernel->currentThread->time = 0; //reset time slice

kernel->currentThread->Yield();

}

if (kernel->currentThread->cpu < 80)

{

kernel->currentThread->cpu+= UserTick;

}

kernel->currentThread->time+= UserTick;

kernel->currentThread->ticks++;

OneTick是Nachos中最小的时钟中断，在这里我们

检查是否当前进程的时间片已经用尽，如果用尽，则进行切换。

当前进程Cpu时间片参数的增加

当50次tick后，对所有的进程的cpu进行减半衰减，从而达到增加其他非运行状态，等待状态的进程优先级

对当前进程的tick进行计数，用于演示。

**Schedular类中修改FindNextToRun函数实现调度算法，满足动态优先级调度**

Thread \*

Scheduler::FindNextToRun ()

{

ASSERT(kernel->interrupt->getLevel() == IntOff);

Thread\* th;

Thread\* th\_top;

ListIterator<Thread\*> \*it = new ListIterator<Thread\*>(readyList);

unsigned int min = -1;

if (readyList->IsEmpty()) {

return NULL;

} else {

for (;!it->IsDone(); it->Next())

{

th = (Thread\*) it->Item();

th->pri = th->nice + th->cpu / 2;

if (min > th->pri)

{

min = th->pri;

th\_top = th;

}

}

}

readyList->Remove(th\_top);

return th\_top;

}

计算每一个就绪进程的动态优先级。并将其优先级最高的返回。

## 算法测试

编写测试的用户态程序

#include "syscall.h"

int foo()

{

while (1)

return 0;

}

int

main()

{

int result;

int t = ThreadFork(foo);

t = ThreadFork(foo);

t = ThreadFork(foo);

t = ThreadFork(foo);

while (1)

Halt();

/\* not reached \*/

}

这个程序尝试开启4个线程，从而用于测试，在开始时会调用在用户态程序中的系统响应，完成其优先级设置

测试：

使用nachos –x add.noff进行运行

按ctrl+c中断程序，来观察每个线程所占用CPU时间数，从而来测试优先级。

以下为运行结果输出的一部分。

---------------

pid=2 nice=10 tick=403504 cpu:31

pid=1 nice=5 tick=469846 cpu:39

pid=4 nice=20 tick=273174 cpu:22

pid=3 nice=15 tick=341610 cpu:30

---------------

pid=4 nice=20 tick=273195 cpu:21

pid=1 nice=5 tick=469882 cpu:37

pid=3 nice=15 tick=341632 cpu:26

pid=2 nice=10 tick=403535 cpu:31

---------------

pid=4 nice=20 tick=273217 cpu:21

pid=2 nice=10 tick=403567 cpu:31

pid=1 nice=5 tick=469922 cpu:38

pid=3 nice=15 tick=341662 cpu:28

---------------

pid=4 nice=20 tick=273239 cpu:21

pid=1 nice=5 tick=469960 cpu:38

pid=3 nice=15 tick=341690 cpu:28

pid=2 nice=10 tick=403599 cpu:31

---------------

pid=2 nice=10 tick=403630 cpu:31

pid=1 nice=5 tick=469996 cpu:37

pid=4 nice=20 tick=273259 cpu:20

main tick=657913 cpu:40

**---------------**

**pid=3 nice=15 tick=341727 cpu:32**

**pid=1 nice=5 tick=470022 cpu:31**

**pid=2 nice=10 tick=403651 cpu:26**

**pid=4 nice=20 tick=273272 cpu:16**

---------------

pid=1 nice=5 tick=470064 cpu:36

pid=3 nice=15 tick=341754 cpu:29

pid=4 nice=20 tick=273304 cpu:24

pid=2 nice=10 tick=403693 cpu:34

---------------

其中，main的NICE为0。所以MAIN的运行时间最长。其他依次递增。

加粗部分可以观察到，1,2,3,4的TICK呈递增关系，1的运行TICK数量最多。说明1的拥有更高的CPU占有权限。

## 其他OneTick的运行方式

在SelfTest中Fork一个进程，用于推动OneTick。

如下：

void

Thread::SelfTest()

{

DEBUG(dbgThread, "Entering Thread::SelfTest");

Thread \*t;

pid\_t f = fork();

if (f){

while (1)

{

sleep(1);

kernel->interrupt->OneTick

}

}else{

t = new Thread("forked thread");

}

//pid\_t p = fork();

t->Fork((VoidFunctionPtr) SimpleThread, (void \*) 1);

kernel->currentThread->Yield();

SimpleThread(0);

}

## 其他的算法的想法

在这里，我采用了不同的动态优先级思路。思路如下：

所有线程有基础优先级p\_s和动态优先级p\_d。

每经过一个tick，所有线程的p\_d+1，当p\_d>100时则不增加，当线程被执行时，则p\_d恢复成p\_s。

通过这种方式，当p\_s大于100时则是静态优先级，p\_s<100时是动态优先级。

其测试结果参考附录。

## 参考

[1] Nachos 4.1 安装文档

[2] Nachos Road Map

## 感想

这次学习NACHOS的过程中，发现NACHOS在系统态，中断无法响应。所以在这里，我采用了用户态的测试。来使用ONETICK，来组织进程来放弃所有权。而不是在由进程本身的代码来执行ONETICK。

Nachos作为操作系统的学习还是相当不错。里面有相当多可以供实现的部分。而相对真实的操作系统，这个操作系统显得更加的简单。而且代码采用OO思想编写。所以非常读懂。

最后感谢老师在操作系统学习过程中的解惑。

## 附录：

另外一种实现方式：

**测试数据**

在这里我测试3个线程的运行在不同优先级下的运行状况：

**相同优先级的线程运行情况**

popacai@popacai-thinkpad:~/nachos/NachOS-4.1/code/build.linux$ nachos -K

tests summary: ok:0

Here is the Nice Value Control

100 Priority will work as Real Time Threading

number of threads is going to be tested: ? 3

1 priority[0~100]: ? 50

2 priority[0~100]: ? 50

3 priority[0~100]: ? 50

starting...

thread 1 looped 0 times

thread 2 looped 0 times

thread 3 looped 0 times

thread 1 looped 10 times

thread 2 looped 10 times

thread 3 looped 10 times

thread 2 looped 20 times

thread 1 looped 20 times

thread 3 looped 20 times

thread 2 looped 30 times

thread 1 looped 30 times

thread 3 looped 30 times

thread 2 looped 40 times

thread 1 looped 40 times

thread 3 looped 40 times

thread 1 looped 50 times

thread 2 looped 50 times

thread 3 looped 50 times

thread 1 looped 60 times

thread 2 looped 60 times

thread 3 looped 60 times

thread 2 looped 70 times

thread 1 looped 70 times

thread 3 looped 70 times

thread 2 looped 80 times

thread 1 looped 80 times

thread 3 looped 80 times

thread 2 looped 90 times

thread 1 looped 90 times

thread 3 looped 90 times

thread 2 looped 100 times

thread 3 looped 100 times

thread 1 looped 100 times

thread 2 looped 110 times

thread 3 looped 110 times

thread 1 looped 110 times

thread 2 looped 120 times

thread 3 looped 120 times

thread 1 looped 120 times

thread 2 looped 130 times

thread 1 looped 130 times

thread 3 looped 130 times

thread 2 looped 140 times

thread 1 looped 140 times

thread 3 looped 140 times

thread 2 looped 150 times

thread 1 looped 150 times

thread 3 looped 150 times

thread 1 looped 160 times

thread 2 looped 160 times

thread 3 looped 160 times

thread 2 looped 170 times

thread 1 looped 170 times

thread 3 looped 170 times

thread 2 looped 180 times

thread 1 looped 180 times

thread 3 looped 180 times

thread 2 looped 190 times

thread 1 looped 190 times

thread 3 looped 190 times

\*\*\* thread 2 finish !~~~~~~~~~~~~~~~~~~~

\*\*\* thread 1 finish !~~~~~~~~~~~~~~~~~~~

\*\*\* thread 3 finish !~~~~~~~~~~~~~~~~~~~

\*\*\* thread 0 finish !~~~~~~~~~~~~~~~~~~~

Machine halting!

Ticks: total 11070, idle 0, system 11070, user 0

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

**相似优先级的运行情况**

popacai@popacai-thinkpad:~/nachos/NachOS-4.1/code/build.linux$ nachos -K

tests summary: ok:0

Here is the Nice Value Control

100 Priority will work as Real Time Threading

number of threads is going to be tested: ? 3

1 priority[0~100]: ? 10

2 priority[0~100]: ? 15

3 priority[0~100]: ? 20

starting...

thread 3 looped 0 times

thread 1 looped 0 times

thread 2 looped 0 times

thread 3 looped 10 times

thread 3 looped 20 times

thread 3 looped 30 times

thread 3 looped 40 times

thread 3 looped 50 times

thread 2 looped 10 times

thread 3 looped 60 times

thread 3 looped 70 times

thread 3 looped 80 times

thread 1 looped 10 times

thread 3 looped 90 times

thread 2 looped 20 times

thread 3 looped 100 times

thread 3 looped 110 times

thread 3 looped 120 times

thread 3 looped 130 times

thread 2 looped 30 times

thread 3 looped 140 times

thread 3 looped 150 times

thread 3 looped 160 times

thread 3 looped 170 times

thread 1 looped 20 times

thread 2 looped 40 times

thread 3 looped 180 times

thread 3 looped 190 times

\*\*\* thread 3 finish !~~~~~~~~~~~~~~~~~~~

thread 2 looped 50 times

thread 2 looped 60 times

thread 2 looped 70 times

thread 2 looped 80 times

thread 1 looped 30 times

thread 2 looped 90 times

thread 2 looped 100 times

thread 2 looped 110 times

thread 2 looped 120 times

thread 1 looped 40 times

thread 2 looped 130 times

thread 2 looped 140 times

thread 2 looped 150 times

thread 2 looped 160 times

thread 2 looped 170 times

thread 1 looped 50 times

thread 2 looped 180 times

thread 2 looped 190 times

\*\*\* thread 2 finish !~~~~~~~~~~~~~~~~~~~

thread 1 looped 60 times

thread 1 looped 70 times

thread 1 looped 80 times

thread 1 looped 90 times

thread 1 looped 100 times

thread 1 looped 110 times

thread 1 looped 120 times

thread 1 looped 130 times

thread 1 looped 140 times

thread 1 looped 150 times

thread 1 looped 160 times

thread 1 looped 170 times

thread 1 looped 180 times

thread 1 looped 190 times

\*\*\* thread 1 finish !~~~~~~~~~~~~~~~~~~~

\*\*\* thread 0 finish !~~~~~~~~~~~~~~~~~~~

Machine halting!

Ticks: total 11070, idle 0, system 11070, user 0

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

**较大优先级差别的运行情况**

popacai@popacai-thinkpad:~/nachos/NachOS-4.1/code/build.linux$ nachos -K

tests summary: ok:0

Here is the Nice Value Control

100 Priority will work as Real Time Threading

number of threads is going to be tested: ? 3

1 priority[0~100]: ? 80

2 priority[0~100]: ? 50

3 priority[0~100]: ? 20

starting...

thread 1 looped 0 times

thread 1 looped 10 times

thread 1 looped 20 times

thread 2 looped 0 times

thread 1 looped 30 times

thread 1 looped 40 times

thread 1 looped 50 times

thread 3 looped 0 times

thread 1 looped 60 times

thread 1 looped 70 times

thread 1 looped 80 times

thread 1 looped 90 times

thread 1 looped 100 times

thread 1 looped 110 times

thread 1 looped 120 times

thread 1 looped 130 times

thread 1 looped 140 times

thread 1 looped 150 times

thread 1 looped 160 times

thread 1 looped 170 times

thread 1 looped 180 times

thread 1 looped 190 times

\*\*\* thread 1 finish !~~~~~~~~~~~~~~~~~~~

thread 2 looped 10 times

thread 2 looped 20 times

thread 2 looped 30 times

thread 2 looped 40 times

thread 2 looped 50 times

thread 2 looped 60 times

thread 2 looped 70 times

thread 2 looped 80 times

thread 2 looped 90 times

thread 2 looped 100 times

thread 2 looped 110 times

thread 2 looped 120 times

thread 2 looped 130 times

thread 2 looped 140 times

thread 2 looped 150 times

thread 2 looped 160 times

thread 2 looped 170 times

thread 2 looped 180 times

thread 2 looped 190 times

\*\*\* thread 2 finish !~~~~~~~~~~~~~~~~~~~

thread 3 looped 10 times

thread 3 looped 20 times

thread 3 looped 30 times

thread 3 looped 40 times

thread 3 looped 50 times

thread 3 looped 60 times

thread 3 looped 70 times

thread 3 looped 80 times

thread 3 looped 90 times

thread 3 looped 100 times

thread 3 looped 110 times

thread 3 looped 120 times

thread 3 looped 130 times

thread 3 looped 140 times

thread 3 looped 150 times

thread 3 looped 160 times

thread 3 looped 170 times

thread 3 looped 180 times

thread 3 looped 190 times

\*\*\* thread 3 finish !~~~~~~~~~~~~~~~~~~~

\*\*\* thread 0 finish !~~~~~~~~~~~~~~~~~~~

Machine halting!

Ticks: total 11070, idle 0, system 11070, user 0

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

**实时优先级运行情况**

popacai@popacai-thinkpad:~/nachos/NachOS-4.1/code/build.linux$ nachos -K

tests summary: ok:0

Here is the Nice Value Control

100 Priority will work as Real Time Threading

number of threads is going to be tested: ? 3

1 priority[0~100]: ? 50

2 priority[0~100]: ? 55

3 priority[0~100]: ? 100

starting...

thread 3 looped 0 times

thread 3 looped 10 times

thread 3 looped 20 times

thread 3 looped 30 times

thread 3 looped 40 times

thread 3 looped 50 times

thread 3 looped 60 times

thread 3 looped 70 times

thread 3 looped 80 times

thread 3 looped 90 times

thread 3 looped 100 times

thread 3 looped 110 times

thread 3 looped 120 times

thread 3 looped 130 times

thread 3 looped 140 times

thread 3 looped 150 times

thread 3 looped 160 times

thread 3 looped 170 times

thread 3 looped 180 times

thread 3 looped 190 times

\*\*\* thread 3 finish !~~~~~~~~~~~~~~~~~~~

thread 1 looped 0 times

thread 2 looped 0 times

thread 2 looped 10 times

thread 2 looped 20 times

thread 2 looped 30 times

thread 2 looped 40 times

thread 1 looped 10 times

thread 2 looped 50 times

thread 2 looped 60 times

thread 2 looped 70 times

thread 2 looped 80 times

thread 2 looped 90 times

thread 1 looped 20 times

thread 2 looped 100 times

thread 2 looped 110 times

thread 2 looped 120 times

thread 2 looped 130 times

thread 2 looped 140 times

thread 1 looped 30 times

thread 2 looped 150 times

thread 2 looped 160 times

thread 2 looped 170 times

thread 2 looped 180 times

thread 2 looped 190 times

\*\*\* thread 2 finish !~~~~~~~~~~~~~~~~~~~

thread 1 looped 40 times

thread 1 looped 50 times

thread 1 looped 60 times

thread 1 looped 70 times

thread 1 looped 80 times

thread 1 looped 90 times

thread 1 looped 100 times

thread 1 looped 110 times

thread 1 looped 120 times

thread 1 looped 130 times

thread 1 looped 140 times

thread 1 looped 150 times

thread 1 looped 160 times

thread 1 looped 170 times

thread 1 looped 180 times

thread 1 looped 190 times

\*\*\* thread 1 finish !~~~~~~~~~~~~~~~~~~~

\*\*\* thread 0 finish !~~~~~~~~~~~~~~~~~~~

Machine halting!

Ticks: total 11070, idle 0, system 11070, user 0

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