# Nachos: Not Another Completely Heuristic Operating System

你的阳光学习频道 **http://study.yoursunny.com** 专题页

最后更新：2008-07-06

**Not Another Completely Heuristic Operating System**, or **Nachos**, is instructional software for teaching undergraduate, and potentially graduate level operating systems courses. It was developed at the University of California, Berkeley, designed by Thomas Anderson, and is used by numerous schools. 以上内容摘自Wikipedia

我使用的操作系统是Ubuntu Desktop ~~7.10~~ 8.04，Nachos版本是C++ 4.1版。

---

## 推荐的资源链接

- Nachos主页 http://www.cs.washington.edu/homes/tom/nachos/
- 李小勇老师提供的Nachos：介绍课件、C++ 4.1版本、Java 5.0版本
- 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……
- 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

---

## 2008-07-06

又弄了一整天，一下子就写完了！2516行的输出……

实验报告的字数统计

```
Words:      28545
Characters (no spaces):    111452
Characters (with spaces):       133315
Paragraphs:      4289
```

```
Sentences:      4379
Pages (approximate):      95
```

分享一条经验：在不知从何入手的时候，可以先写出一个个与实验目标有关的功能块，然后把它们组合起来。

---

## 2008-07-05 再次开始：虚拟存储

第二次作业，拖到今天才开始做；前面有考试、后面有生产实习，只有短短2天可以做这个作业。这次的作业要求是：实现虚拟存储（单线程，根据Modify、Referenced位决定换出，对NOFF文件进行lazy-load）。

与虚拟存储有关的源码文件包括

- machine/translate.h（不准修改），页表项的结构。Reference位称为use，Modify位称为dirty
- userprog/addrspace.cc，虚拟地址空间；包括pageTable初始化和管理，以及读取NOFF文件。目前是虚拟地址与物理地址一一对应，也就是“实模式”。要改成虚拟存储，肯定需要另一个类内部使用的page=>pageFrame转换表。
- userprog/exception.cc，中断处理；缺页中断就是在这里处理的
- threads/alarm.cc，时钟中断；需要定期清除Reference位

Nachos虚拟机实际上支持两种虚拟存储机制（machine/translate.cc）：

1. pageTable：单级页表，但是无快表；当页表项中valid位为0时此页表项无效（注释中If this bit is set, the translation is ignored.说valid为1时页表项无效，正好说反了），就引起缺页中断
2. TLB：快表，但是无页表；如果快表中没查到，就引起缺页中断

缺页中断处理完毕后，PC不增加，重新取指执行。

在machine/machine.cc里发现了《计算机组成》一道考题（写个程序判断LittleEndian还是BigEndian）的答案：

```
static
void CheckEndian()
{
    union checkit {
        char charword[4];
        unsigned int intword;
    } check;

    check.charword[0] = 1;
    check.charword[1] = 2;
    check.charword[2] = 3;
    check.charword[3] = 4;

#ifdef HOST_IS_BIG_ENDIAN
    ASSERT (check.intword == 0x01020304);
#else
    ASSERT (check.intword == 0x04030201);
#endif
}
```

稍改一下就是那道题的答案了；union这个方法，看起来比指针“高级”一点。

今天主要是阅读，程序也写了一些，未测试。

noff.h需要添加include guard（#ifndef……#define……#endif）才能正常编译。

---

## 2008-04-17 作业完成

我的报告写完了！完全在Linux下，用Firefox和Google Docs完成的。Google Docs对文章字数的统计和易懂性的评价：

**Counts**

```
Words:      2457
Characters (no spaces):       19004
Characters (with spaces):     21432
Paragraphs:       558
Sentences:  717
Pages (approximate):    17
```

**Readability**

Average sentences per paragraph: 1.28
Average words per sentence: 3.43
Average characters per word: 7.73
Average words per page: 144.53
Flesch Reading Ease: [?] 55.81
Flesch-Kincaid Grade Level: [?] 6.00
Automated Readability Index: [?] 17.00

---

## 2008-04-16 撰写实验报告

在不抓紧做作业的孩子们的强烈要求下，老师把作业截止日期延长了三天，19日清晨06:00交。对于我这个做的差不多的作业来说，只不过有了更加充裕的时间撰写实验报告。

程序可能还有问题，但是不打算再作什么修改。开始写报告了。

上pic吧（其实是stderr的输出文本~）

```
Thread main fork 588
Forking thread: main588 f(a): 134595880 0x24c
Putting thread on ready list: main588 priority=128
```

Newly-ready thread main588(priority128) < running thread main(129), yield and switch 新线程优先级较高，马上中断当前线程

```
Yielding thread: main
Putting thread on ready list: main priority=129
Scheduler readyList contents:
Thread main448400 priority=129 status=READY
Thread main priority=129 status=READY
Thread main716 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main716 priority=255 status=READY
Thread main448400 priority=129 status=READY
Thread main588 priority=128 status=READY
Switching from: main to: main588
Beginning thread: main588
NoffThread main588 start at 588
```

Thread main588 used up its time, inc priority to 129 and yield 时间片用完了，降低优先级

```
Yielding thread: main588
Putting thread on ready list: main588 priority=129
Scheduler readyList contents:
```

Thread main priority=129 status=READY　为什么没有main448400？因为Yielding main588得时候已经从readyList里取出了

```
Thread main588 priority=129 status=READY
Thread main716 priority=255 status=READY
Scheduler allThreadsList contents:
Thread postal worker priority=128 status=BLOCKED
Thread main priority=129 status=READY
Thread main716 priority=255 status=READY
Thread main448400 priority=129 status=READY
Thread main588 priority=129 status=READY
```

Switching from: main588 to: main448400　总是选择优先级最高的READY线程，main716优先级255为最低，不会被选中

```
Now in thread: main448400
```

---

## 2008-04-13 初步成形

我的作业，上午解决了大部分问题，在一个简单test case的调试输出看起来，我的scheduler已经能正常运行了。剩下的问题是，如何让助教相信这个scheduler能正常运行并符合需求呢？我是SDE的料，不是SDET的料啊（SDE=Software Development Engineer，SDET=Software Development Engineer in Test）。

声明：对Nachos C++、Linux平台运行Nachos、作业有问题可以来信探讨，但是我手头项目太多，不能保证回信；我不提供作业代码和实验报告，请勿索取。

---

## 2008-04-12 地址空间，Yield，alarm

10日，我自认为完成了Exec系统调用，实际上并不正确：我Fork出的UNIX函数是NoffProcess，它创建了一个新的AddrSpace；然而，AddrSpace的构造函数会把整个machine->mainMemory清零，这样就破坏了所有原来的MIPS程序。应该实现ThreadFork这个系统调用，才能正常创建新线程。

很快就把ThreadFork系统调用（以及相应的UNIX函数NoffThread）写好了，但是发现除了我的scheduler以外还发生了其他线程切换，从Interrupt那里一路查过去，发现是threads/alarm作怪（是一个随机时隙的round-robin），禁用它。不过这个alarm的代码还是很有参考价值的，它告诉我：在中断服务函数中如果要Yield“当前线程”（被中断掉的线程），应该调用kernel->interrupt->YieldOnReturn()而不是kernel->currentThread->Yield()。

我的作业总算会调用定时中断了，不过还是有问题：已经Yield的线程仍然会被降低优先级，而且SortedList::Apply似乎没有正确工作。

---

## 2008-04-10 线程切换原理，系统调用参数的取得，新线程启动

中文输入法问题上次解决了，但是Terminal里又显示不出中文了，gcc输出的却是中文（显示为问号）。把gcc输出改成英文的方法是在Makefile里加一行
LANG=en_US.UTF_8，就可以了。
Nachos的线程切换并不是在MIPS虚拟机里进行的，而是把“执行MIPS虚拟机的UNIX用户态线程”一起切换掉。

要使Nachos能同时启动多个线程，必须实现系统调用SC_Exec或SC_ExecV，然后在test/中的某个程序里执行Exec(程序名)。那么，userprog/exception.cc里如何取得这个程序名呢？
test/start.s指出，第一个参数（即程序名参数）在MIPS的r4寄存器里。那我就这样写：

```
char* exec_name=(char*)kernel->machine->ReadRegister(4);
DEBUG(dbgSys,"Exec "<<exec_name);
```

取数字参数这样一点也没问题，字符串不行。运行时出现错误信息：“段错误”——也就是内存访问异常，缓冲区溢出实验就是报这个错。
看看exec_name究竟是什么？把上面的两个char*改成int，运行时出现：Exec 512。
这下明白了，r4寄存器里是MIPS虚拟机内存的指针，而不是主机内存。知道怎么办了吧？用kernel->machine->ReadMem逐字节copy出来就可以了，不过要当心缓冲区溢出。

要启动一个新的线程并执行另一个noff程序（就像Windows的CreateProcess那样），仅仅

创建Thread和Fork是不行的。Thread::Fork只能开始一个UNIX用户态线程，而不是noff程序。解决方法是Fork出一个UNIX函数，这个函数创建一个AddrSpace然后载入一个noff并运行（noff的文件名可以作为参数传递，都在主机内存里，不用再次copy）。参考：threads/thread.cc的Thread::SelfTest，threads/main.cc的run an initial user program。#这个方法不可行，见2008-04-12

至于作业——优先级Round-Robin调度，准备做成动态调整的。今天做的，中断时间不对，且线程无法终止，就不截图了。

---

## 2008-04-03 系统调用

一不小心把Ubuntu里的中文输入法弄坏了，反复安装多次才修复，而且弄出了传说中的fcitx，还有微软雅黑字体……

尝试用nathos调用执行“用户程序”（MIPS平台的程序，用./nachos -x 程序名.noff执行），看到了很多Unexcepted system call，怎么回事？这篇文章介绍，nathos本来只支持很少几种系统调用——SC_Halt（程序终止）、SC_Add（很无聊的加法），其他要自己做的（修改userprog/exception）。

2008-03-20提出的问题“哪些代码运行于MIPS虚拟机？”，现在大致清楚了：

- test/内的代码运行于MIPS虚拟机，需要使用交叉编译器。
  （注意test/Makefile.dep里的CPP、GCCDIR两个定义要修改成实际安装的版本）
- 其他代码均直接运行于Linux，使用x86编译器、不是交叉编译器。
  （2008-03-18提到修改build.linux/Makefile里的路径，这种做法不正确、Makefile注释里要求不要这么做，所以还是手动cd build.linux再make depend和make吧）

出于好奇还看了看lib/copyright.h版权说明：版权归加州大学，授权任何人使用、修改、发布等，条件是加州大学不承担任何责任。

今天的成果：（命令行启动参数见threads/main.cc开头注释，“-x 文件名.noff”执行用户程序、“-d +”显示调试信息）

```
sunny@sunny:~/study/IS206/nachos-4.1/code/build.linux$ ./nachos -x ../test/sunny
.noff


tests summary: ok:0
Machine halting!

Ticks: total 2253, idle 0, system 240, user 2013
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
```

**sunny.noff程序主体是for (i=0;i<1000;++i);这个空循环，Ticks: user 2013数值较大，证明noff被运行了。因为没有系统调用，所以noff无法调用I/O；下次创建一个带DEBUG的系统调用，就可以观察noff的运行了。**

---

2008-03-20 阅读Road Map

今天阅读了A Road Map Through Nachos的1～3章节，了解了很多问题。有一点还没完全搞清：哪些代码是运行于UNIX，哪些代码是运行于MIPS虚拟机的？
作业一是增加优先级和时间片调度，只要修改code/threads/scheduler.cc就可以了。最大的问题是，如何知道线程的优先级？类对外的接口是不可以改的，所以只能用自适应的算法，自动配置优先级。而从原来的非抢占式调度改为抢占式调度，必须修改时钟中断，以便在合适的时机剥夺线程的CPU使用权——这样thread.cc也需要改……

---

## 2008-03-18 重新安装新版本，全部可以编译

我放弃了2月20日弄的3.4版本，今天用的是老师给的C++ 4.1版。只需要修改很少的地方，就可以全部编译通过：

- `lib/list.cc`
  开头加上#ifdef LIST_H，末尾加上#endif
  引用本类的成员变量、成员函数时要写上this->

- `lib/hash.cc`

  开头加上#ifdef HASH_H，末尾加上#endif

  引用本类的成员变量、成员函数时要写上this->

- build.linux/Makefile与Makefile.dep

  去除那个--writeable-strings；如果用IDE的话，可能还需要把../替换成合适的路径（我用Code::Blocks，在code目录建立的工程，所以要把../替换成./）

---

## 2008-02-20 Nachos的第一次安装，能编译/threads了

今天，我从ftp://202.120.2.148/nachos/finalversion/code/下载到一份大致可用的Nachos源码(C++ 3.4版)。不过，整个程序无法编译通过（FileSystem类有几个函数没有定义，导致/userprog无法编译通过；可能本该如此，那些函数需要自己完成？），而/threads目录已经可以编译通过、且能够运行了。运行截图：

```
sunny@sunny:~/study/IS206/nachos/code/threads$ ./nachos
Machine halting!

Ticks: total 10, idle 0, system 10, 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

Cleaning up...
sunny@sunny:~/study/IS206/nachos/code/threads$
```

我修改了如下文件：

- `/Makefile`

  把MAKE=gmake改成MAKE=make，因为Ubuntu中没有gmake；当然sudo ln /usr/bin/gmake /usr/bin/make也可以。

- `/Makefile.common`

  去掉里面的两个-writable-strings，因为gcc不再支持这个命令行参数。

- `/threads/thread.cc`

  去掉以下三个static关键字：

  ```
  static int Thread::CurID=0;
  static Thread* Thread::GetThread(int id)
  ```

`static void Thread::InitThread()`

- `/machine/sysdep.h`

  注释掉文件末尾atoi、atof、abs这三行定义，因为它们的throw与现在的C库冲突。

- `/threads/system.h`

  增加一个#include <stdlib.h>，否则会找不到atoi等函数。

- `/machine/sysdep.h`

  增加一个#include <errno.h>，否则会出现non-TLS云云。

  srand、rand、sleep、abort、exit、socket等函数全部注释掉，否则会与stdlib.h冲突。

  ReadFromSocket函数中，&size改成(socklen_t *)&size，否则会提示类型不匹配。

参考资料：基于Nachos平台的操作系统上机实践指南 by 南开大学机器智能研究所

[via]

---