操作系统原理课程设计

实验手册

2010 年 **4** 月

目　　录

第一章 试验环境介绍

1.1 引言

操作系统上机实践环节是操作系统课程的重要组成部分，对于理解操作系统课程中的相关理论和知识点有着非常重要的作用。为帮助学生更好的完成上机实践作业，特编写此基于Nachos 平台的操作系统上机实践指南。

1.2 Nachos 平台的功能特点简介

1.2.1 什么是 Nachos

Nachos 的全称是“Not Another Completely Heuristic Operating System”，它是一个可修改和跟踪的操作系统教学软件。它给出了一个支持多线程和虚拟存储的操作系统骨架，可让学生在较短的时间内对操作系统中的基本原理和核心算法有一个全面和完整的了解。

1.2.2 Nachos 的特点功能介绍

在本科的操作系统教学中，能够提供一个展示真实操作系统是如何工作的工程环境是很重要的，但同时也要求这个工程环境便于学生的理解和修改，所以我们采用 Nachos 作为操作系统课程的教学实践平台。Nachos 是美国加州大学伯克莱分校在操作系统课程中已多次使用的操作系统课程设计平台，在美国很多大学中得到了应用，它在操作系统教学方面具有以下几个突出的优点：

- 采用通用虚拟机

Nachos 是建立在一个软件模拟的虚拟机之上的，模拟了 MIPS R2/3000 的指令集、主存、中断系统、网络以及磁盘系统等操作系统所必须的硬件系统。许多现代操作系统大多是先在用软件模拟的硬件上建立并调试，最后才在真正的硬件上运行。用软件模拟硬件的可靠性比真实硬件高得多，不会因为硬件故障而导致系统出错，便于调试。虚拟机可以在运行时报告详尽的出错信息，更重要的是采用虚拟机使 Nachos 的移植变得非常容易，在不同机器上移植 Nachos，只需对虚拟机部分作移植即可。

采用 R2/3000 指令集的原因是该指令集为 RISC 指令集，其指令数目比较少。Nachos 虚拟机模拟了其中的 63 条指令。由于 R2/3000 指令集是一个比较常用的指令集，许多现有的编译器如 gc++能够直接将 C 或 C++源程序编译成该指令集的目标代码，于是就不必编写编译器，读者就可以直接用 C/C++语言编写应用程序，使得在 Nachos 上开发大型的应用程序也成为可能。

- 使用并实现了操作系统中的一些新的概念

随着计算机技术和操作系统技术的不断发展，产生了很多新的概念。Nachos 将这些新概念融入操作系统教学中，包括网络、线程和分布式应用。而且 Nachos 以线程作为一个基本概念讲述，取代了进程在以前操作系统教学中的地位。

Nachos 的虚拟机使得网络的实现相当简单。与 MINIX 不同，Nachos 只是一个在宿主机上运行的一个进程。在同一个宿主机上可以运行多个 Nachos 进程，各个进程可以相互

通讯，作为一个全互连网络的一个节点；进程之间通过 Socket 进行通讯，模拟了一个全互连网络。

- 确定性调试比较方便，随机因素使系统运行更加真实

因为操作系统的不确定性，所以在一个实际的系统中进行多线程调试是比较困难的。由于 Nachos 是在宿主机上运行的进程，它提供了确定性调试的手段。所谓确定性调试，就是在同样的输入顺序、输入参数的情况下，Nachos 运行的结果是完全一样的。在多线程调试中，可以将注意力集中在某一个实际问题上，而不受操作系统不确定性的干扰。

另外，不确定性是操作系统所必须具有的特征，Nachos 采用了随机因子模拟了真实操作系统的不确定性。

- 简单而易于扩展

Nachos 是一个教学用操作系统平台，它必须简单而且有一定的扩展余地。Nachos 不是向读者展示一个成功的操作系统，而是让读者在一个框架下发挥自己的创造性进行扩展。例如一个完整的类似于 UNIX 的文件系统是很复杂的，但是对于文件系统来说，无非是需要实现文件的逻辑地址到物理地址的映射以及实现文件 inode、打开文件结构、线程打开文件表等重要的数据结构以及维护它们之间的关系。Nachos 中具有所有这些内容，但是在很多方面作了一定的限制，比如只有一级索引结构限制了系统中最大文件的大小。读者可以应用学到的各种知识对文件系统进行扩展，逐步消除这些限制。Nachos 在每一部分给出很多课程作业，作为读者进行系统扩展的提示和检查对系统扩展的结果。

- 面向对象性

Nachos 的主体是用 C++的一个子集来实现的。目前面向对象语言日渐流行，它能够清楚地描述操作系统各个部分的接口。Nachos 没有用到面向对象语言的所有特征，如继承性、多态性等，所以它的代码就更容易阅读和理解。

Nachos 共有五个功能模块，分别是机器模拟、线程管理、文件系统管理、用户程序和虚拟存储以及网络系统。它们的具体实现会在第二章中进行详细介绍。

1.3 Nachos 平台的搭建与配置

1.3.1 Nachos 试验环境文件列表

在我们的教学网站可以下载到搭建 nachos 试验平台的所有软件，下面就将搭建 nachos 试验环境所需的软件作一个简要的描述：

NachOS-4.1_110.gz：

nachos 的源代码压缩包，在 linux 下可以用 gzip、tar 解开，在 windows 下直接使用 winrar 就可解压开。

mips-decstation.linux-xgcc.tar.gz:

nachos 的交叉编译工具，在 linux 下可以用 gzip、tar 解开，在 windows 下直接使用

winrar 就可解压开。

Si35Setup.rar：

Windows 下的源代码编辑查看工具。

vmware-5.5.1.rar：

Vmware 虚拟机软件的安装程序，运行它就可以在我们的 Windows 安装上 vmware。Vmwar 可以在一个物理机器上模拟出虚拟的硬件机器，进而我们就可以在这些虚拟的硬件机器上安装我们的操作系统了。（详细使用参见后面的试验环境搭建）

1.3.2 Nachos 试验环境搭建步骤

Nachos 最初由美国加州大学伯克利分校为其操作系统课程的教学实习而开发。这个操作系统一开始就设计成只能在 unix like 的操作系统下运行，并没有针对 windows 平台的移植。有鉴于国内教学实习中普遍使用 windows 平台，所以我们需要通过安装虚拟机软件，创建 linux 虚拟机来最终实现我们的 nachos 代码的编译运行。下面我们将通过几个步骤来具体的描述 nachos 试验环境的搭建。

1.3.2.1 Vmware 的安装

Vmware 是一个虚拟机软件，它可以在 windows 平台上虚拟出真实机器的硬件环境的，使得我们可以在不购买新机器的情况下就可以在一个机器上运行多个操作系统。

Vmware 的安装和普通的 windows 应用程序安装没有太大的差别，是一个相当“傻瓜”的过程，只要按照提示，依次点击“下一步”就可顺利地完成 vmware 的安装了。

1.3.2.2 虚拟机的创建

这一步的主要任务就是要告诉 vmware 我们想要虚拟什么样的硬件机器。下面将以图示的方式一步步地讲解虚拟机的创建：

启动 vmware 程序点击如图 1-1 所示的菜单

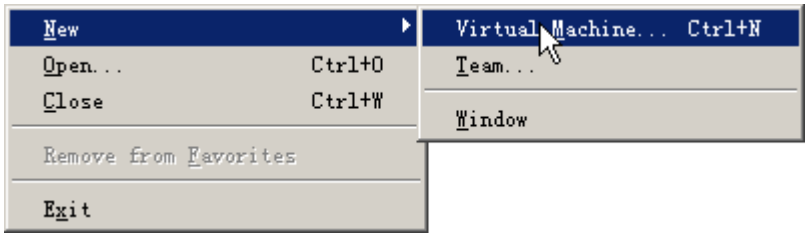

图 1-1 VMWare 程序启动

或是点击 home-tab 上的 new virtural machine（如图 1-2）就可开始虚拟机的创建过程。

图 1-2 VMWare 程序启动（二）

在接下来的对话框中选择定制创建虚拟机点击下一步按钮（如图 1-3）

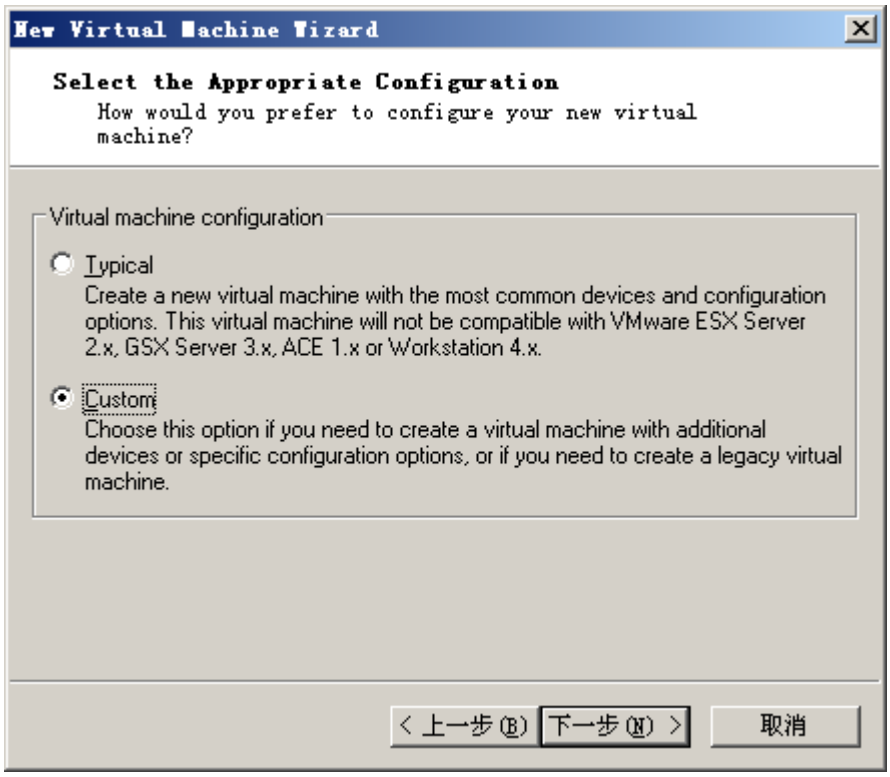

图 1-3 定制虚拟机

在接下来的虚拟机格式中选择系统默认的 new-workstation5 格式即可，这样我们可以获得一些好处比如可以抓图，但是也有一些坏处那就是创建的虚拟机文件不能在老版本的 vmware 上打开。

点击下一步后出现选择操作系统类型对话框，我们只要选择 linux－redhat linux 就行了。

点击下一步，弹出的对话框要求我们回答虚拟机名称和虚拟机文件安放的路径，只要根据自己的情况回答就行了。（如图 1-4）

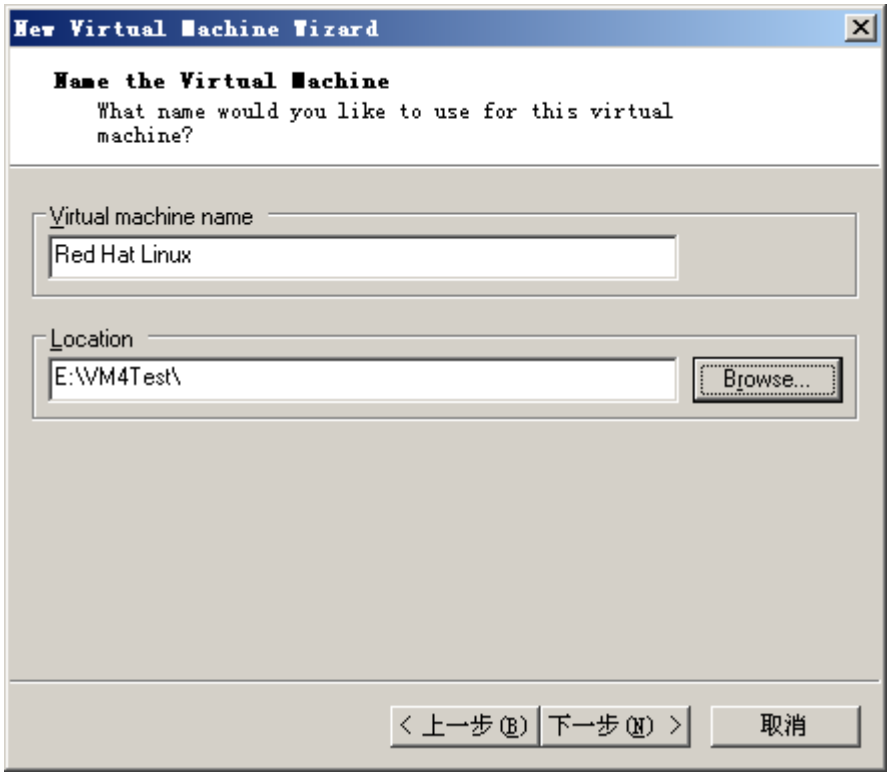

图 1-4 选择虚拟机名称和安放路径

点击下一步后弹出内存设定对话框，这个对话框是要回答 vmware 需要给我们的虚拟机器模拟多大的内存，在我们的实验环境中 256M 已经足够了。（如图 1-5）

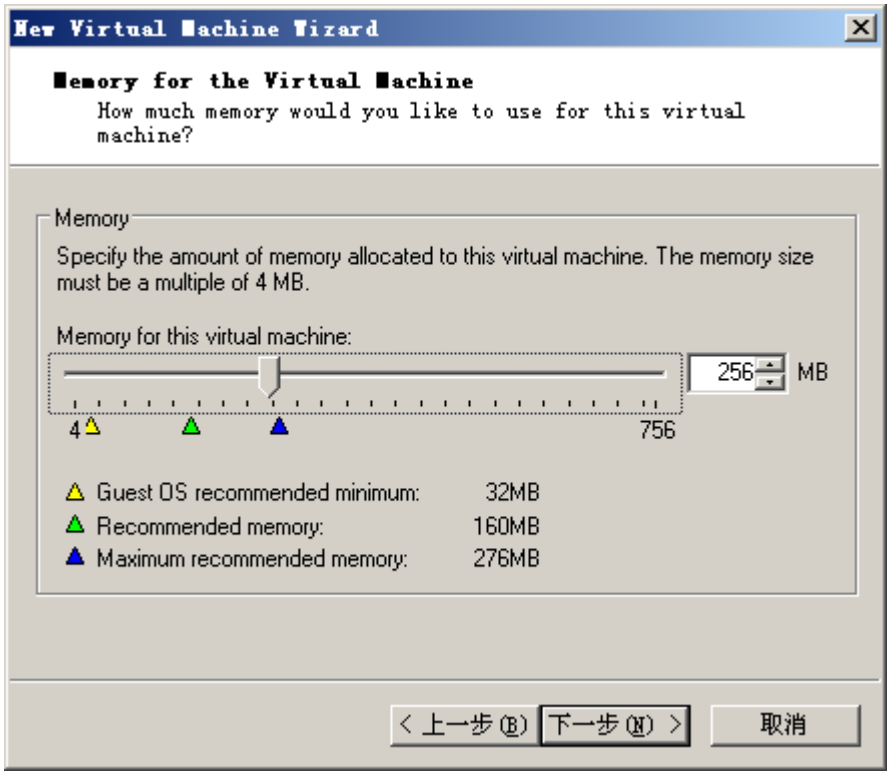

图 1-5 虚拟机定制内存

点击下一步，就进入了网络选择对话框，在我们的试验环境中选择最后一项“不使用网络”就可以了。（如图 1-6）

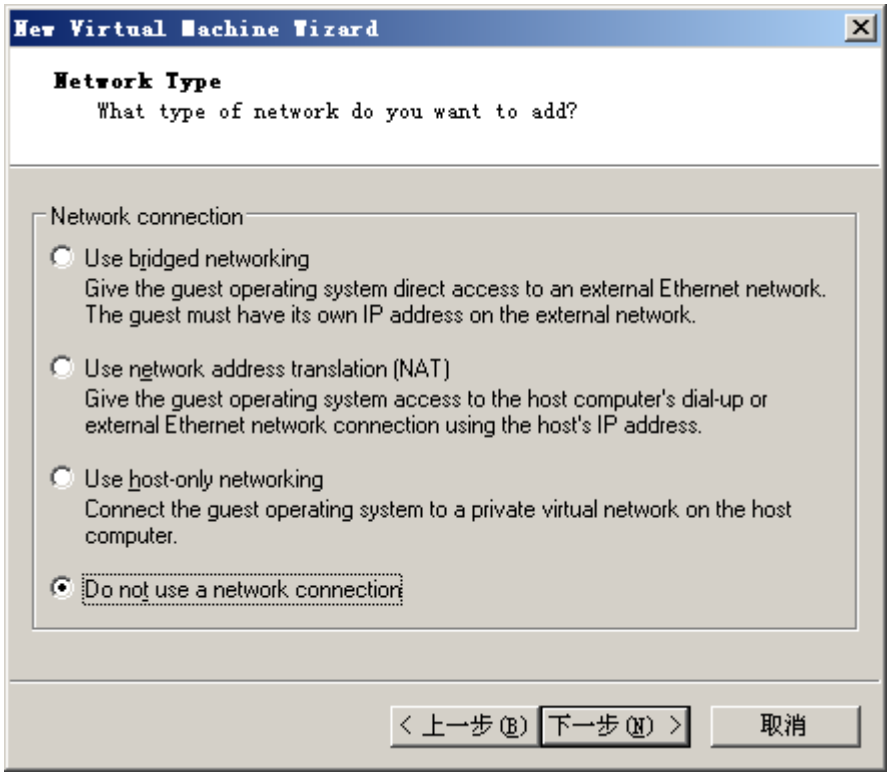

图 1-6 虚拟机网络设置

点击下一步，进入选择 I/O 适配器类型，这一步不需要我们干预接受默认值，点击下一步就可以了。（如图 1-7）

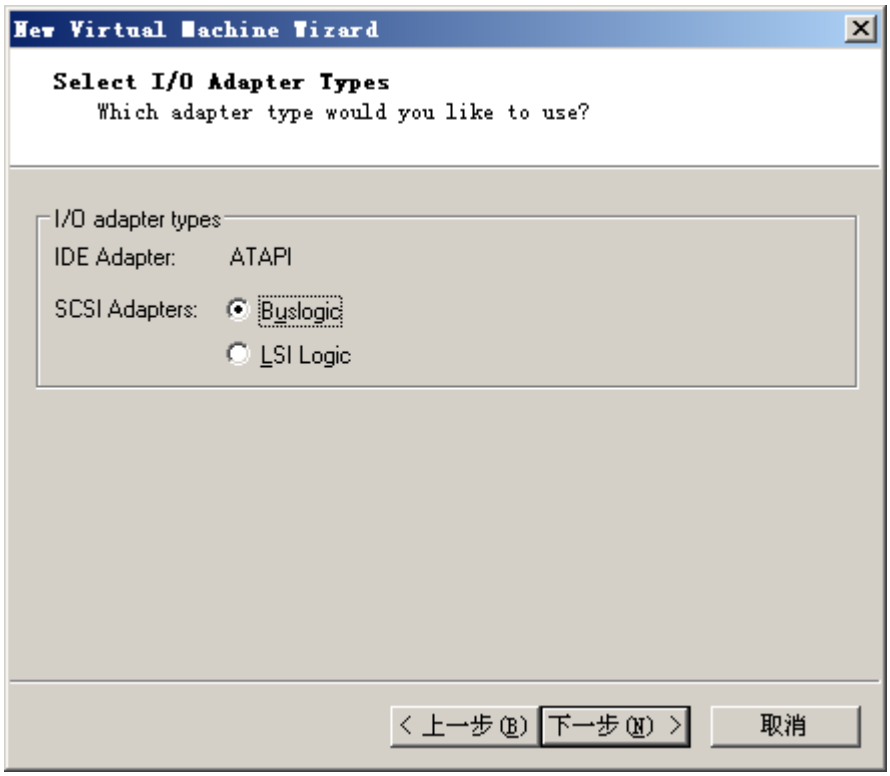

图 1-7 网络适配器设置

接下来就进入了硬盘创建的步骤了，选择创建 IDE 的硬盘，系统就会弹出对话框，要求回答硬盘的大小，在我们的试验环境中有 4G 大小就足够了。（按图 1-8，1-9，1-10）

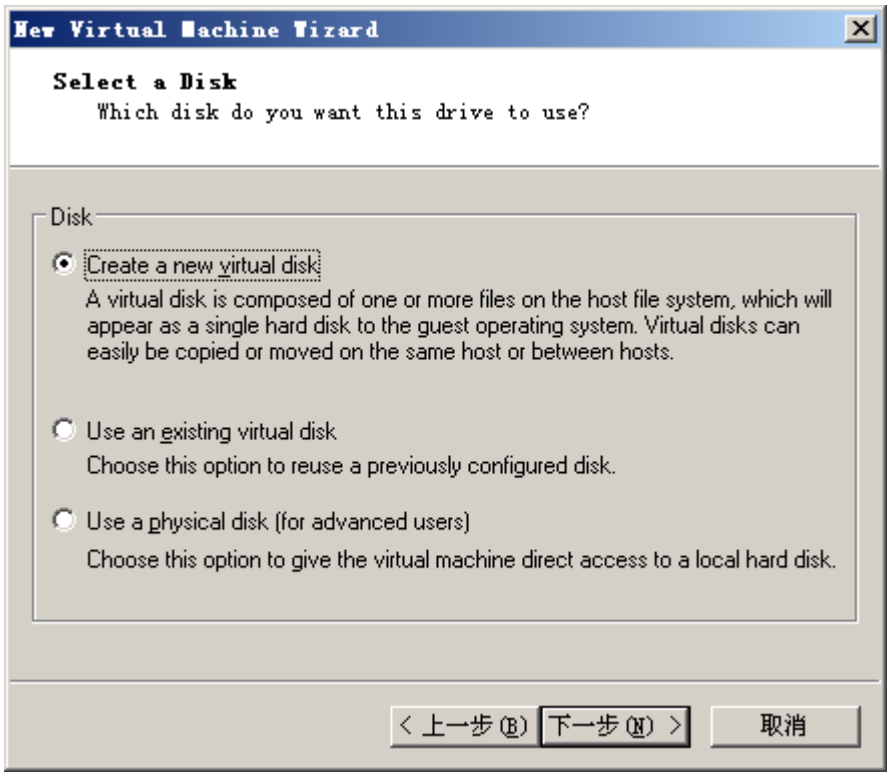

图 1-8 创建虚拟磁盘

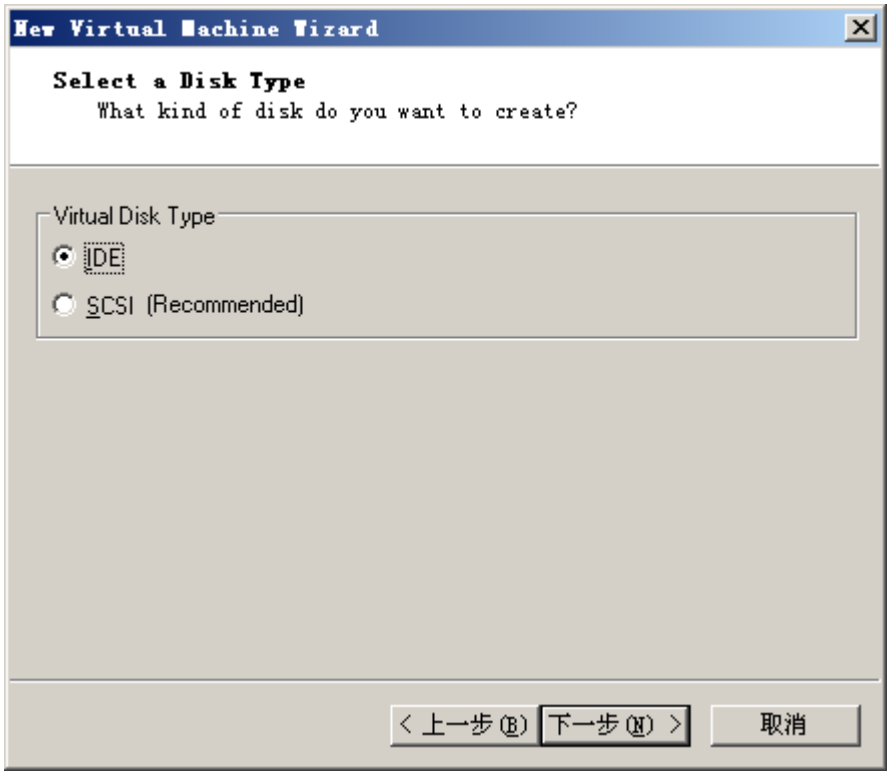

图 1-9 选择磁盘种类

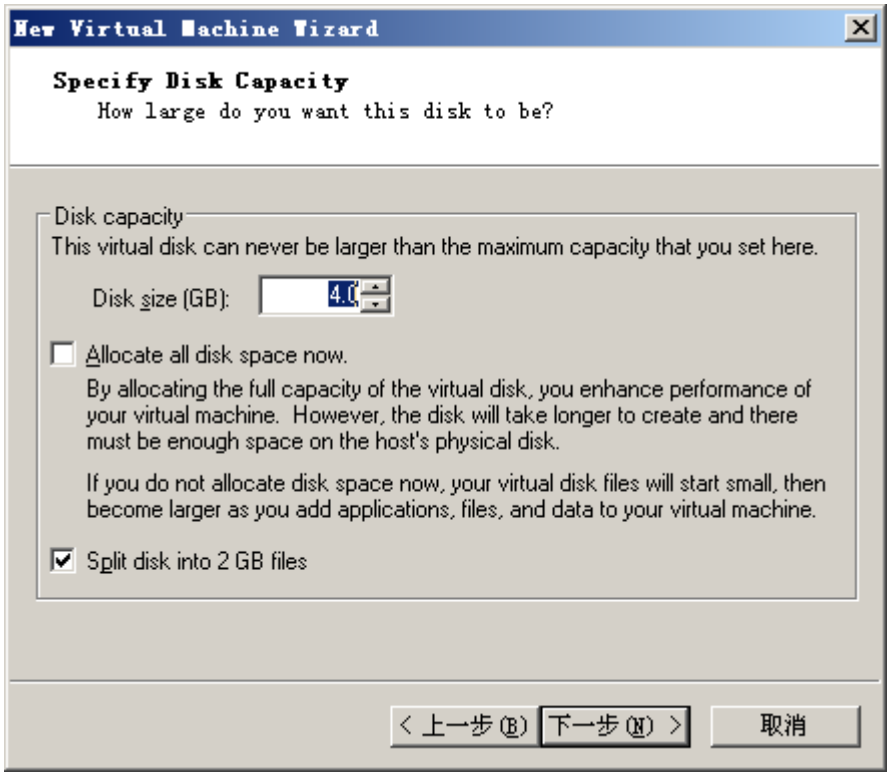

图 1-10 选择磁盘大小

点击下一步后，虚拟机的创建就到了最后一个步骤回答硬盘文件的名称，根据自己的情况回答这个名称点击完成，整个虚拟机的创建过程就结束了。（如图 1-11）

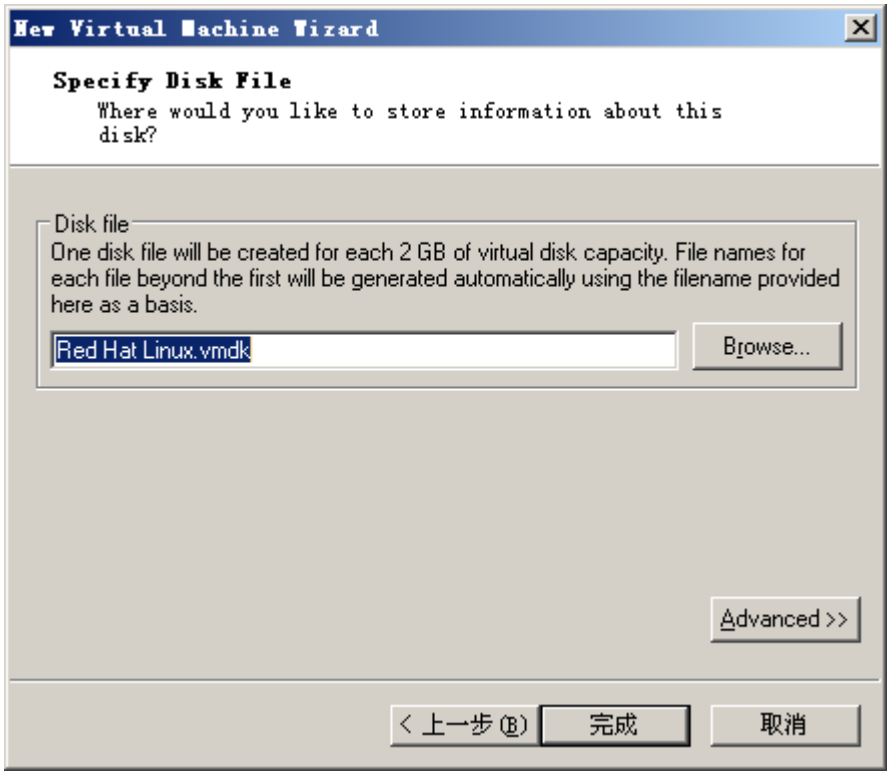

图 1-11 指定硬盘文件名称

虚拟机创建完成后 vmware 的主窗口就会多出一个标签页，这页的内容正式我们所创建的虚拟机的信息。（如图 1-12）

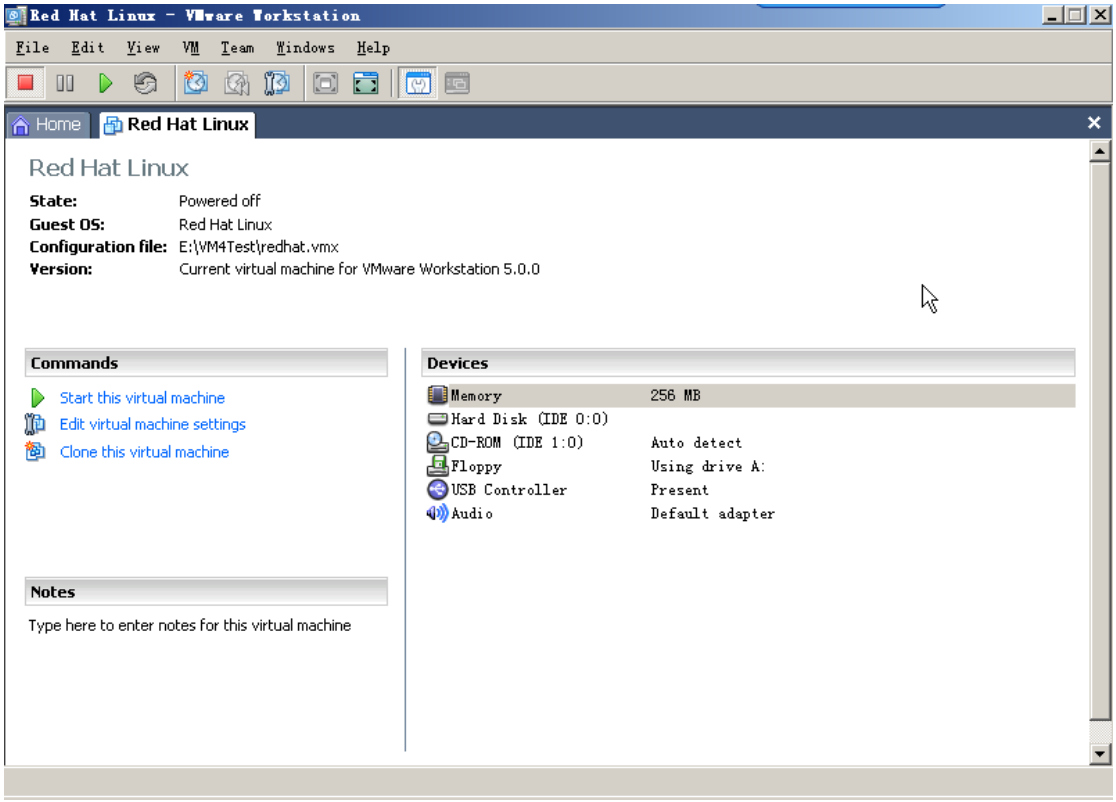

图 1-12 新建虚拟机信息

点击标签页上的“start this virtual machine”就可以启动我们刚刚创建的虚拟机，但是由于我们只是让 vmware 帮我们模拟出了一个硬件机器，我们还没有向这个机器上安装任何软件，所以我们的机器运行后除了一些 BIOS 程序的输出外别的什么也没有，也什么都不能干，所以我们需要进行下一步，向虚拟机上安装 OS 和应用程序。

1.3.2.3 虚拟机上 linux 的安装

虚拟机创建好了，vmware 只是按照我们的要求模拟出了一个硬件机器，到目前位置这个虚拟机上并没有安装任何软件，也不能做任何事情。这一步的目的就是要向虚拟的硬件机器上安装 linux 操作系统，进而安装其他应用程序。

在虚拟机上安装 linux 和在真实机器上由光驱安装 linux 是相同的过程。

首先需要把安装光盘放到光驱中，双击主界面上的 CD-ROM，弹出如图 1-13 对话框：

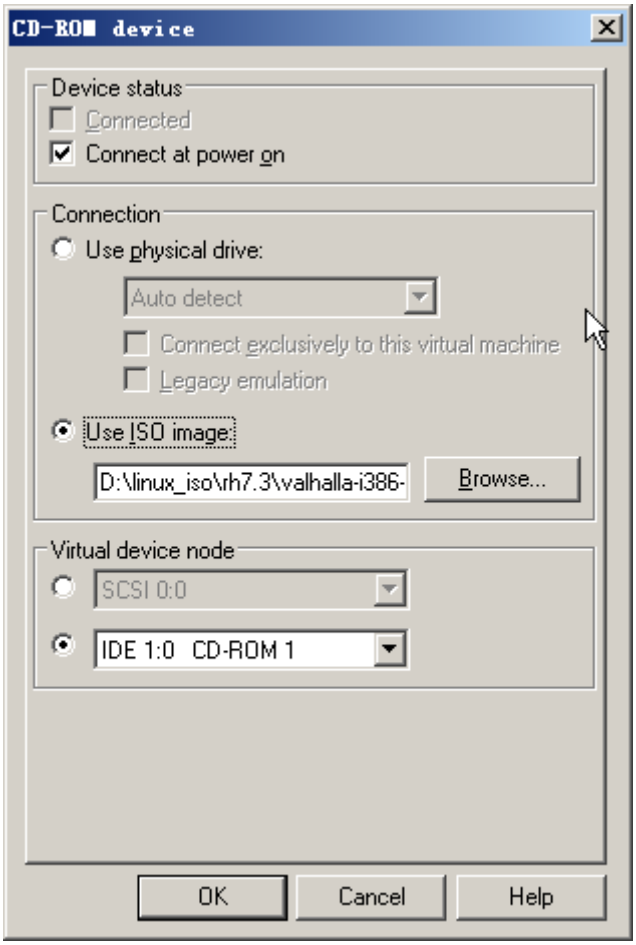

图 1-13 CD-ROM 对话框

如果你有光驱，也有 linux 的安装光盘就可以选择“Use physical drive”；如果有 linux 的 ISO 映象，就需要选“Use ISO image”，并且指定好映像的位置。安装光盘设置好后我们就可以启动虚拟机器进行安装了。

机器正常从光驱启动就会出现如图 1-14 的画面：

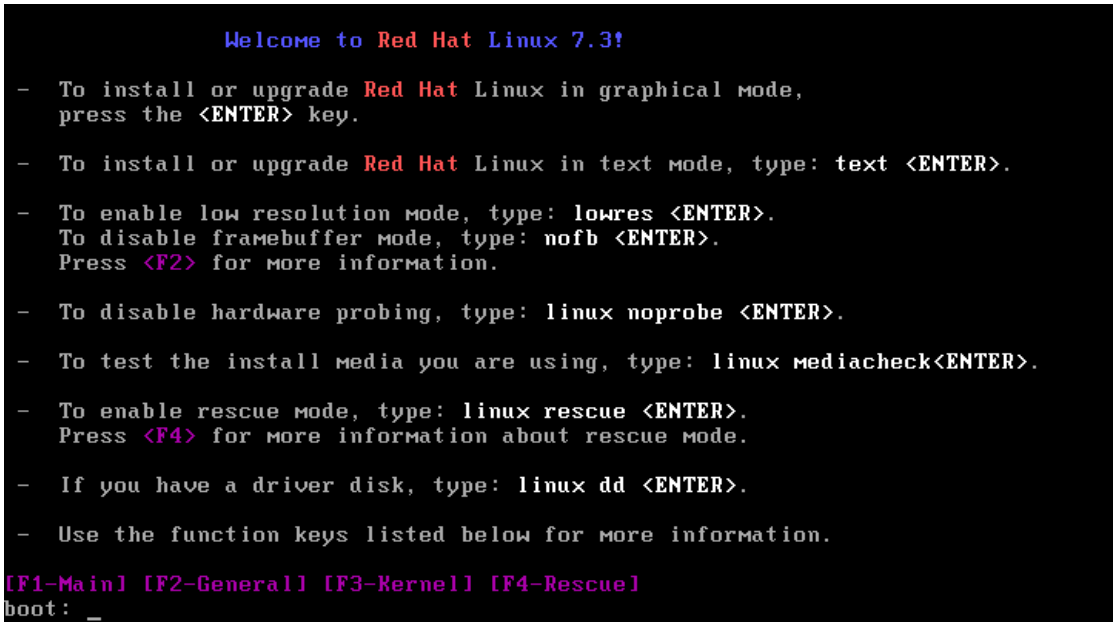

图 1-14 安装 Linux 启动画面

只有鼠标点击 vmware 的窗口之后，所有的鼠标和键盘输入才能定向到虚拟机

我们是新安装 linux 并且使用图形界面安装过程，所以这时我们只要敲“Enter”就可以开始安装过程了。

Ctrl+Alt 是 vmware 的热键，如果想将鼠标从 vmware 的环境回到 windows 下，就可以使用这个热键；如果要从全屏方式回到窗口方式，也需要点击这个热键

安装过程其实是很简单的大多情况下只要点击“next”按钮就可以了，下面仅就比较关键的步骤进行一下描述：

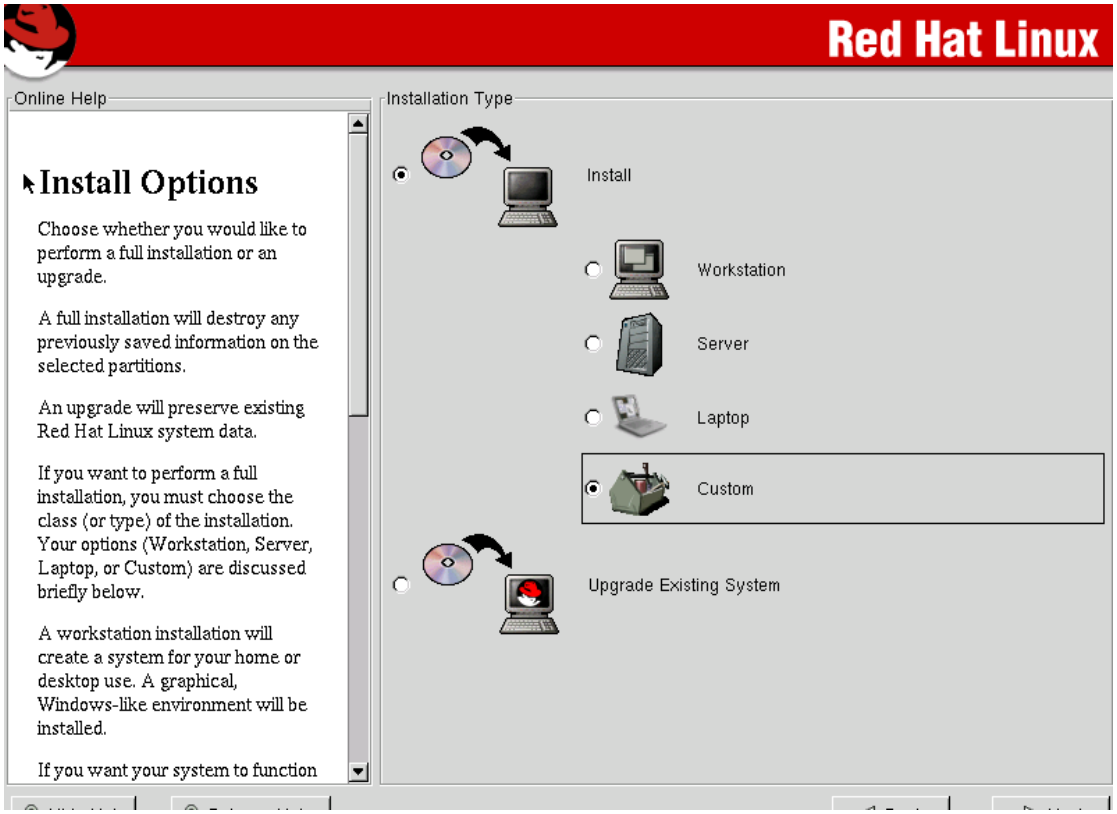

图 1-15　定制安装 Linux

在上图 1-15 所使得步骤中，最好是选择定制安装，这样我们可以对安装过程进行控制，安装我们所需要的东西。

在接下来的步骤中我们将会遇到如下的画面，它让我们选择怎样的分区方式，我们只要按照如图所示选择系统自动分区就行了，系统自动分区的结果还是比较合理的。（如图 1-16）

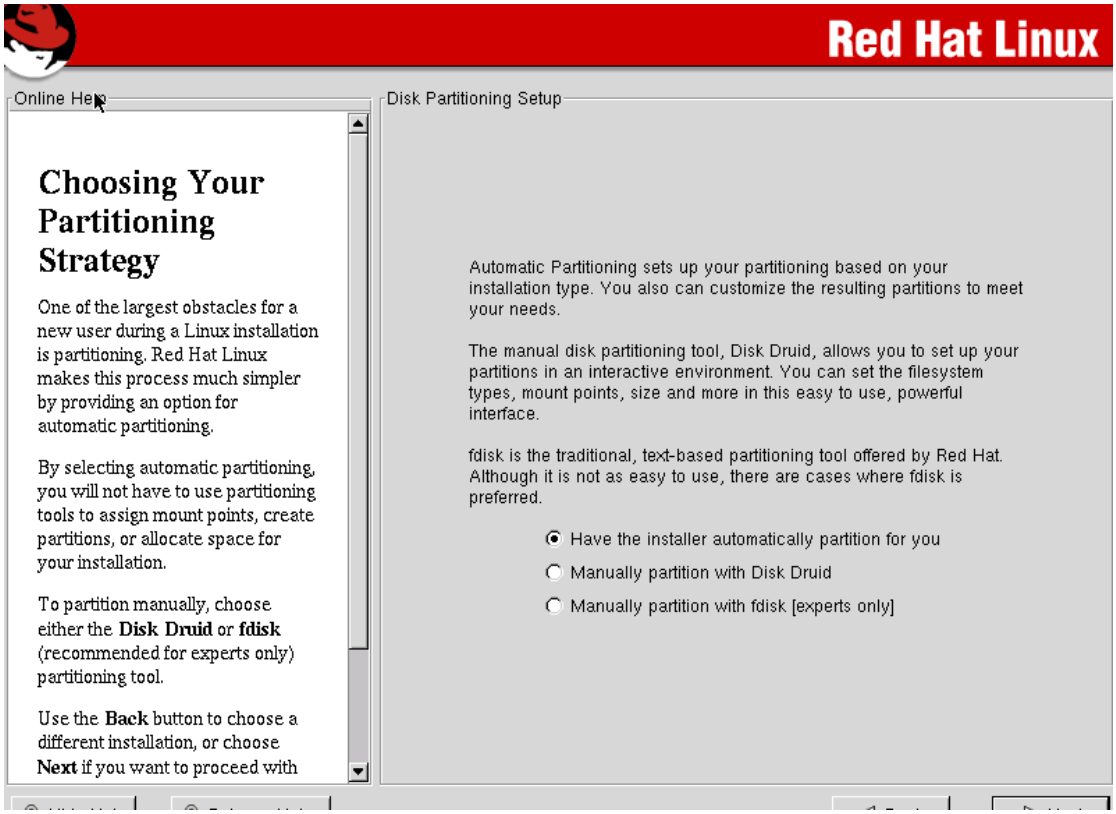

图 1-16 选择分区方式

接下来的若干步只要点击“next”就行了，其中有的步骤询问是否要删除磁盘上的数据这是只要回答“Yes”就行了，我们是新装操作系统，磁盘上没有什么有用的数据，也不用担心本机的数据会丢失，VMWare 操作的只是它的磁盘文件的数据，不会对本机造成任何伤害。

当安装程序如下图所示要求我们回答 root 密码时我们就要按照自己的情况回答这个密码，但是一定要记住，因为这个密码是超级用户的密码，对于系统特别重要。（如图 1-17）

接下来的几步仍然只需要“next”直到包的选择步骤：（如图 1-18）

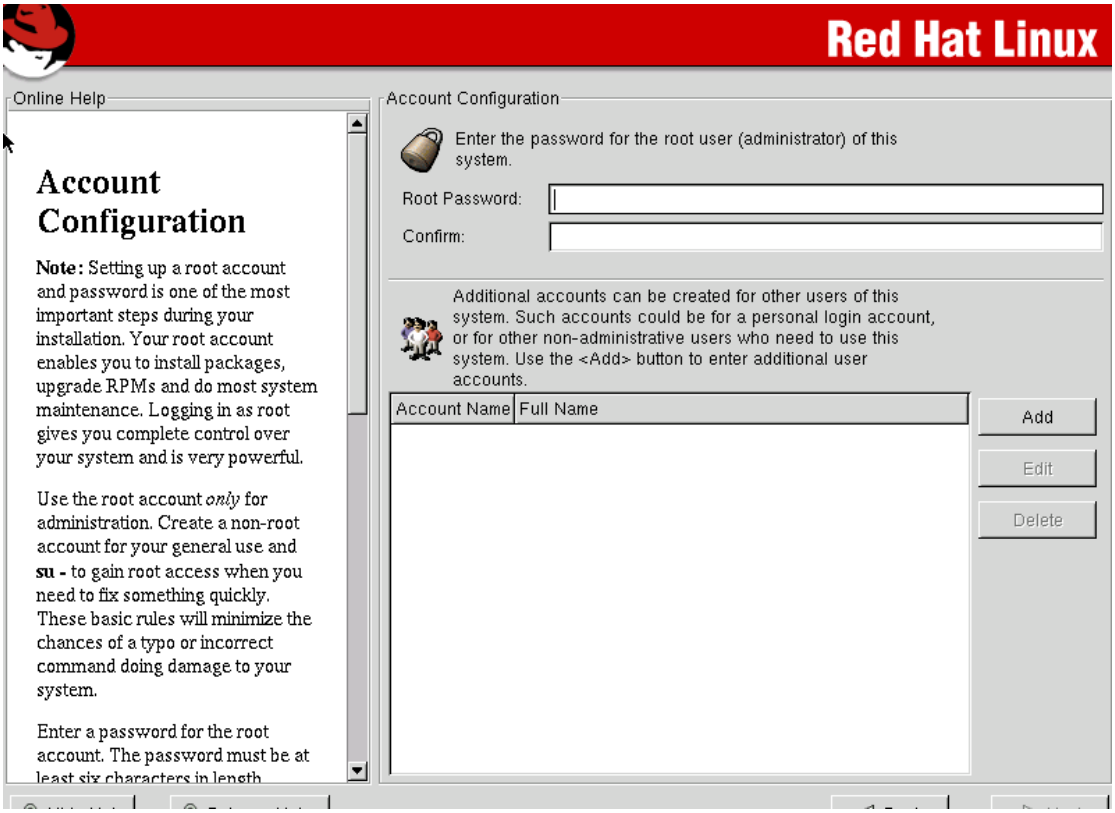

图 1-17 设置超级用户密码

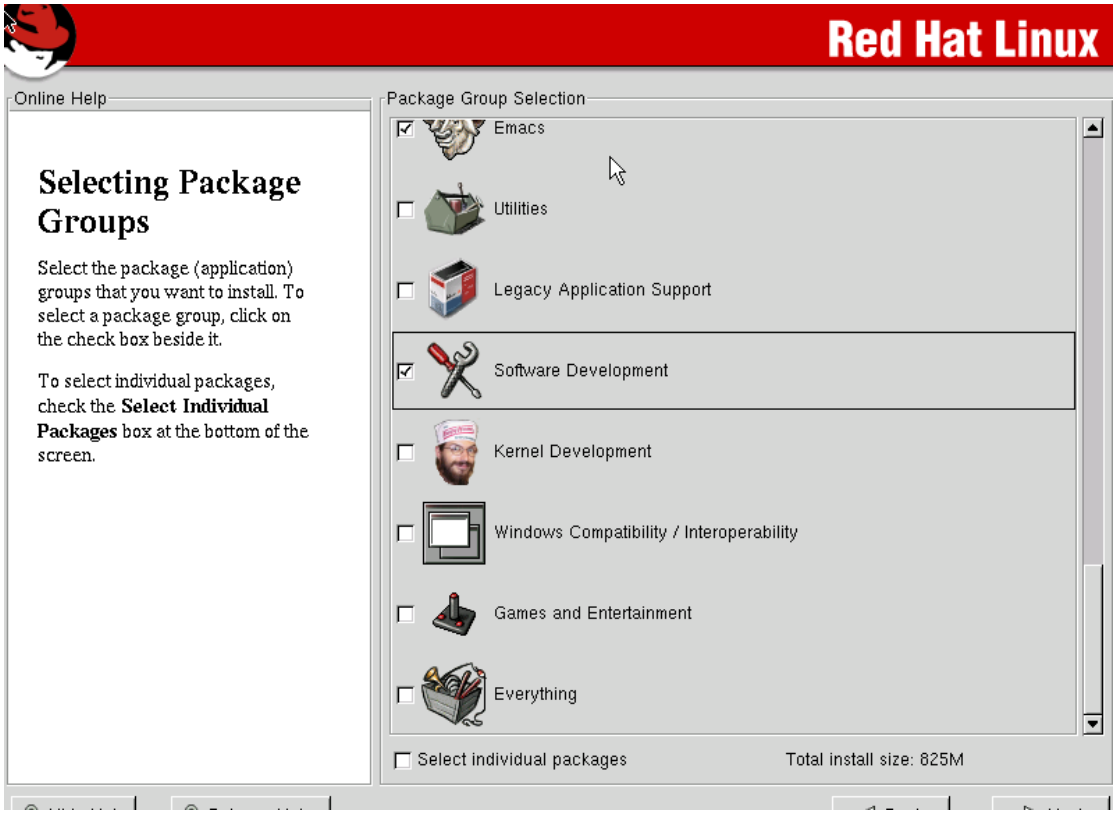

图 1-18 软件包的选择

我们只需要按照自己的需要选择自己所需要的包就可以了，包选择的多了最后的虚拟机磁盘的使用量就自然会大一些。

我们必须选择 software development 包，因为它会为我们安装编译调试工具，这是我们编译 nachos 系统所需要的

接下来的步骤就基本不用回答什么问题了只要一路“next”下去就行了。安装后虚拟机顺利重启，经过一个不太长的启动过程我们就可以看到登录界面，那么我们的安装过程就算是成功完成了。

虚拟机重启的时候不要忘记点击 F2 进入 BIOS 设置画面，将启动顺序变为从硬盘启动，这样我们启动的速度能加快一点

1.3.2.4 虚拟机和本机之间的数据传递

- 设 windows 主机的 ip 地址为 x.y.z.w，在 windows 下共享一个目录 nachos，设共享名为 nachos，建立用户 zk 具有完全访问权限，密码为：abcd。
- Linux 下通过 samba 进行访问。

mkdir /prog

smbmount //192.168.0.200/nachos /prog –o username=zk

输入密码：abcd

/prog 变为工作目录。

Linux 下的命令和文件命是区分大小写的，在 windows 命令提示符下是不区分大小写的

1.3.2.5 Si35Setup.rar 的安装

虽然在 linux 我们可以高效地完成所有开发所需的任务，但是对于刚刚开始接触 unix/linux 的同学，一下子用熟那么多的工具还是有些困难的。正是基于这一点我们的教学网站上也提供了在 windows 下用来阅读代码的工具 Source Insight，借助于它我们可以加快代码阅读速度。

这个软件的安装是个很简单的过程只要运行 setup 一路 next 下去就可以了。

代码阅读时要先建工程，点击 project－new project，然后按照要求回答源代码的位置，工程就会顺利的建成。代码阅读时如果需要一些功能比如想要查找某个符号的定义只要在选定的符号上点右键就会弹出菜单，选择相应的命令就可以了。

1.3.3 Nachos 的功能模块组成结构

编译后的 nachos 在我们的 linux 操作系统中，只相当于一个普通的程序，运行起来就是一个普通的进程。另一方面它又是一个不普通的程序，他是完全按照操作系统思想而开发出的一个操作系统内核。它的结构就是一个完整的操作系统结构。下图标出了 nachos 操作系统的结构。

图 1-20　nachos 系统结构

1.3.4 Nachos 的编译运行开发环境

nachos 虽说是一个操作系统但是他在 linux 下运行，也就相当与 linux 的一个普通进程，所以 nachos 的编译、调试、运行和一般的 linux 程序的调试连接运行是没有什么两样的。下面我们就从编辑、编译、编辑、代码阅读等方面对 linux 下的程序开发环境做一下阐述：

- 程序编辑

  用 Source Insight 编辑。

- 程序编译

  在 linux 下的 nachos 代码是借助 make 工具来实现编译连接的。make 工具是一个工程管理工具，make 是根据 Makefile 的指示来完成所有工作的。Nachos 的每个零部件的代码中都包含一个 Makefie 文件，所以我们在修改 nachos 代码后直接使用 make 命令就可以编译连接我们的程序了，而不再需要其他额外的工作。

- 程序调试

  在 linux 下调试我们一般使用 gdb 作为调试工具。

  gdb 使用过程大致如下：

  启动 gdb: >gdb　　gdb 启动后就进入了 gdb 的

  使用 file 命令加载要调试的可执行程序：>file nachos

  断点设置：　>break main　　在 main 函数上设置断点

  程序运行一步：>next　　next 命令不跟到函数里面去

  程序跟进：　>step　　step 命令可以跟跟到函数里面去

  显示变量的值：　>print temp　temp 是变量命，这个命令将显示 temp 的值

  显示源代码：　>list　　这个命令可以列出所调试程序的源代码

  gdb 是一个很强大的调试工具，这里我们只列出了我们平时用的几个简单的命令，更复杂的使用可以参照 gdb 的手册或是联机文档。

第二章 Nachos 平台技术实现说明

2.1 Nachos 的机器模拟机制概述

2.1.1 Sysdep 模块实现机制分析

本模块主要在文件 sysdep.cc 和 sysdep.h 中实现。

Nachos 的运行环境可以是多种操作系统，由于每种操作系统所提供的系统调用或函数调用在形式和内容上可能有细微的差别。sysdep 模块的作用是屏蔽掉这些差别。

下面就主要的函数功能进行一下说明：

- PoolFile 函数

  语法： bool PoolFile (int fd)

  参数： fd：文件描述符，也可以是一个套接字 (socket)

  功能：测试一个打开文件 fd 是否有内容可以读，如果有则返回 TRUE，否则返回 FALSE。当 Nachos 系统处于 IDLE 状态时，测试过程有一个延时，也就是在一定时间范围内如果有内容可读的话，同样返回 TRUE。

- OpenForWrite 函数

  语法： int OpenForWrite (char *name)

  参数： name：文件名

  功能：为写操作打开一个文件。如果该文件不存在，产生该文件；如果该文件已经存在，则将该文件原有的内容删除。

- OpenForReadWrite 函数

  语法： int OpenForReadWrite (char *name, bool crashOnError)

  参数： name:：文件名；crashOnError: crash 标志

  功能： 为读写操作打开一个文件。当 crashOnError 标志设置而文件不能读写打开时，系统出错退出。

- Read 函数

  语法： void Read (int fd, char *buffer, int nBytes)

  参数： fd：打开文件描述符；
  buffer：读取内容的缓冲区；
  nBytes：需要读取的字节数

  功能： 从一个打开文件fd 中读取nBytes 的内容到buffer 缓冲区。如果读取失败，系统退出。

  注意：这和系统调用read 不完全一样。read 系统调用返回的是实际读出的字节数，而 Read 函数则必须读出nBytes，否则系统将退出。如果需要使用同read 系统调用相对应的函数，请用ReadPartial。

- ReadPartial 函数

  语法： int ReadPartial (int fd, char *buffer, int nBytes)

  参数： fd：打开文件描述符；
  buffer：读取内容的缓冲区；
  nBytes: 需要读取的最大字节数

  功能： 从一个打开文件fd 中读取nBytes 的内容到buffer 缓冲区。

- WriteFile 函数

  语法： void WriteFile (int fd, char *buffer, int nBytes)

参数： fd：打开文件描述符；
buffer：需要写的内容所在的缓冲区；
nBytes: 需要写的内容最大字节数
功能： 将buffer 缓冲区中的内容写nBytes 到一个打开文件fd 中。
注意：这和系统调用write 不完全一样。write 系统调用返回的是实际写入的字节数，而WriteFile 函数则必须写入nBytes，否则系统将退出。

- Lseek 函数

语法： void Lseek (int fd, int offset, int whence)
参数： fd：文件描述符；
offset：偏移量；
whence: 指针移动的起始点
功能： 移动一个打开文件的读写指针，含义同lseek 系统调用；出错则退出系统。

- Tell 函数

语法： int Tell (int fd)
参数： fd：文件描述符
功能： 指出当前读写指针位置

- Close 函数

语法： void Close (int fd)
参数： fd：文件描述符
功能： 关闭当前打开文件fd，如果出错则退出系统。

- Unlink 函数

语法： bool Unlink (char *name)
参数： name：文件名
功能： 删除文件。

- OpenSocket 函数

语法： int OpenSocket ()
参数： 无
功能： 申请一个socket。

- CloseSocket 函数

语法： void CloseSocket (int sockID)
参数： sockID：socket 标识
功能： 释放一个socket。

- AssignNameToSocket 函数

语法： void AssignNameToSocket(char *socketName, int sockID)
参数： socketName：socket文件名；sockID：socket标识
功能： 将一个文件名和一个socket 标识联系起来，于是将一个SOCKET 文件同一个Nachos进程连接起来，使宿主机上该Nachos 进程成为一个网络节点。

- DeAssignNameToSocket 函数

语法： void DeAssignNameToSocket(char *socketName)
参数： socketName:：socket 文件名
功能： 将一个文件名删除，实际上是和相应的socket 标识脱离关系。

- PoolSocket 函数

语法： bool PoolSocket (int sockID)
参数： socketID：socket 标识

功能： 查询一个socket 是否有内容可以读取。

- ReadFromSocket 函数

语法： void ReadFromSocket (int sockID, char *buffer, int packetSize)

参数： socketID：socket 标识；

buffer：读取内容的暂存空间；

packetSize：读取数据包的大小

功能： 从一个socket 标识中读取packetSize 大小的数据包，放在buffer 缓冲中。

- SendToSocket 函数

语法： void SendToSocket (int sockID, char *buffer, int packetSize, char *toName)

参数： socketID：socket 标识；buffer：发送内容的暂存空间；

packetSize：发送数据包的大小；

toName：要接收数据包的Nachos 虚拟机模拟网络文件的文件名

功能： 向socket 标识中发送packetSize 大小的数据包。

- CallOnUserAbort 函数

语法： void CallOnUserAbort (VoidNoArgFunctionPtr func)

参数： func：函数指针

功能： 设定一个函数，在用户强制退出系统时调用。

- Delay 函数

语法： void Delay (int seconds)

参数： seconds：需要延迟的秒数

功能： 系统延迟一定的时间。

- Abort 函数

语法： void Abort ()

参数： 无

功能： 退出系统 (非正常退出)。

- Exit 函数

语法： void Exit (int exitCode)

参数： exitCod：向系统的返回值

功能： 退出系统。

- RandomInit 函数

语法： void RandomInit (unsigned seed)

参数： seed：随机数产生魔数

功能： 初始化随机数发生器。

- Random 函数

语法： int RandomInit ()

参数： 无

功能： 产生一个随机整数。

- AllocBoundedArray 函数

语法： char * AllocBoundedArray (int size)

参数： size：需要申请的空间大小

功能： 申请一个受保护的存储空间。

- DeallocBoundedArray 函数

语法： void DeallocBoundedArray (char *ptr, int size)

参数： ptr：要释放空间的指针；size：申请的空间大小

功能： 将受保护的存储空间释放。

2.1.2 中断处理模块实现机制分析

本模块主要在文件 interrupt.cc 和 interrupt.h 中实现。

中断模块的主要作用是模拟底层的中断机制。可以通过该模拟机制来启动和禁止中断(SetLevel)；该中断机制模拟了 Nachos 系统需要处理的所有的中断，包括时钟中断、磁盘中断、终端读/终端写中断以及网络接收/网络发送中断。

中断的发生总是有一定的时间。比如当向硬盘发出读请求，硬盘处理请求完毕后会发生中断；在请求和处理完毕之间需要经过一定的时间。所以在该模块中模拟了时钟的前进。为了实现简单和便于统计各种活动所占用的时间起见，Nachos 规定系统时间在以下三种情况下前进：

- 执行用户态指令

  执行用户态指令，时钟前进是显而易见的。我们认为，Nachos 执行每条指令所需时间是固定的，为一个时钟单位(Tick)。

- 重新打开中断

  一般系统态在进行中断处理程序时，需要关中断。但是中断处理程序本身也需要消耗时间，而在关闭中断到重新打开中断之间无法非常准确地计算时间，所以当中断重新打开的时候，加上一个中断处理所需时间的平均值。

- 就绪队列中没有进程

  当系统中没有就绪进程时，系统处于 Idle 状态。这种状态可能是系统中所有的进程都在等待各自的某种操作完成。也就是说，系统将在未来某个时间发生中断，到中断发生的时候中断处理程序将进行中断处理。在系统模拟中，有一个中断等待队列，专门存放将来发生的中断。在这种情况下，可以将系统时间直接跳到中断等待队列第一项所对应的时间，以免不必要的等待。

当前面两种情况需要时钟前进时，调用 OneTick 方法。OneTick 方法将系统态和用户态的时间分开进行处理，这是因为用户态的时间计算是根据用户指令为单位的；而在系统态，没有办法进行指令的计算，所以将系统态的一次中断调用或其它需要进行时间计算的单位设置为一个固定值，假设为一条用户指令执行时间的 10 倍。

虽然 Nachos 模拟了中断的发生，但是毕竟不能与实际硬件一样，中断发生的时机可以是任意的。比如当系统中没有就绪进程时，时钟直接跳到未处理中断队列的第一项的时间。这实际情况下，系统处于 Idel 状态到中断等待队列第一项发生时间之间，完全有可能有其它中断发生。由于中断发生的时机不是完全随机的，所以在 Nachos 系统中运行的程序，不正确的同步程序也可能正常运行，我们在此需要密切注意。

Nachos 线程运行有三种状态：

- Idle 状态

  系统 CPU 处于空闲状态，没有就绪线程可以运行。如果中断等待队列中有需要处理的除了时钟中断以外的中断，说明系统还没有结束，将时钟调整到发生中断的时间，进行

中断处理；否则认为系统结束所有的工作，退出。

- 系统态

  Nachos 执行系统程序。Nachos 虽然模拟了虚拟机的内存，但是 Nachos 系统程序本身的运行不是在该模拟内存中，而是利用宿主机的存储资源。这是 Nachos 操作系统同真正操作系统的重要区别。

- 用户态

  系统执行用户程序。当执行用户程序时，每条指令占用空间是 Nachos 的模拟内存。

Nachos 需要处理的中断种类主要有：

TimerInt：时钟中断
DiskInt：磁盘（读/写）中断
ConsoleWriteInt：终端写中断
ConsoleReadInt：终端读终端
NetworkSentInt：网络发送中断
NetworkRecvInt：网络接收中断

中断等待队列是 Nachos 虚拟机最重要的数据结构之一，它记录了当前虚拟机可以预测的将在未来发生的所有中断。当系统进行了某种操作可能引起未来发生的中断时，如磁盘的写入、向网络写入数据等都会将中断插入到中断等待队列中；对于一些定期需要发生的中断，如时钟中断、终端读取中断等，系统会在中断处理后将下一次要发生的中断插入到中断等待队列中。中断的插入过程是一个优先队列的插入过程，其优先级是中断发生的时间，也就是说，先发生的中断将优先得到处理。

当时钟前进或者系统处于 Idle 状态时，Nachos 会判断中断等待队列中是否有要发生的中断，如果有中断需要发生，则将该中断从中断等待队列中删除，调用相应的中断处理程序进行处理。

中断处理程序是在某种特定的中断发生时被调用，中断处理程序的作用包括可以在现有的模拟硬件的基础上建立更高层次的抽象。比如现有的模拟网络是有丢失帧的不安全网络，在中断处理程序中可以加入请求重发机制来实现一个安全网络。

在该部分模块中比较重要的两个类是PendingInterrupt类和Interrupt类，具体实现可以参考interrupt.cc和interrupt.h文件。

2.1.3 时钟中断模块实现机制分析

本模块主要在文件 timer.cc 和 timer.h 中实现。

该模块的作用是模拟时钟中断。Nacho 虚拟机可以如同实际的硬件一样，每隔一定的时间会发生一次时钟中断。这是一个可选项，目前 Nachos 还没有充分发挥时钟中断的作用，只有在 Nachos 指定线程随机切换时启动时钟中断，在每次的时钟中断处理的最后，加入了线程的切换。实际上，时钟中断在线程管理中的作用远不止这些，时钟中断还可以用作：

- 线程管理中的时间片轮转法的时钟控制，不一定每次时钟中断都会引起线程的切换，而是由该线程是否的时间片是否已经用完来决定；
- 分时系统线程优先级的计算；
- 线程进入睡眠状态时的时间计算，可以通过时钟中断机制来实现 sleep 系统调用，在时

钟中断处理程序中，每隔一定的时间对定时睡眠线程的时间进行一次评估，判断是否需要唤醒它们。

Nachos 利用其模拟的中断机制来模拟时钟中断。时钟中断间隔由 TimerTicks 宏决定（100 倍 Tick 的时间）。在系统模拟时有一个缺陷，如果系统就绪进程不止一个的话，每次时钟中断都一定会发生进程的切换（可参考 system.cc 文件中 TimerInterruptHandler 函数）。所以运行 Nachos 时，如果以同样的方式提交进程，系统的结果将是一样的。这不符合操作系统的运行不确定性的特性。所以在模拟时钟中断的时候，加入了一个随机因子，如果该因子设置的话，时钟中断发生的时机将在一定范围内是随机的。这样有些用户程序在同步方面的错误就比较容易发现。但是这样的时钟中断和真正操作系统中的时钟中断将有不同的含义。不能像真正的操作系统那样通过时钟中断来计算时间等等。是否需要随机时钟中断可以通过设置选项(-rs)来实现。

2.1.4 终端设备模块实现机制分析

本模块主要在文件 console.cc 和 console.h 中实现。

该模块的作用是模拟实现终端的输入和输出。包括两个部分，即键盘的输入和显示输出。终端输入输出的模拟是异步的，也就是说当发出终端的输入输出请求后系统即返回，需要等待中断发生后才是真正完成了整个过程。

Nachos 的终端模拟借助了Console 类中的两个成员函数readFile 和writeFile。这两个文件分别模拟键盘输入和屏幕显示。当readFile 为NULL 时，Nachos 以标准输入作为终端输入；当writeFile 为NULL 时，Nachos 以标准输出作为终端输出。

两个成员函数ConsoleReadPoll 以及ConsoleWriteDone 的作用同Timer 模拟中的TimerHandler成员函数。Nachos 需要以一定的时间间隔检查终端是否有字符供读取；当然在这些中断处理程序中，还包括了一些统计的工作。

系统的终端操作有严格的工作顺序，对读终端来说：

CheckCharAvail -> GetChar -> CheckCharAvail -> GetChar ->...

系统通过定期的读终端中断来判断终端是否有内容供读取，如果有则读出；如果没有，下一次读终端中断继续判断。读出的内容将一直保留到GetChar 将其读走。

对写终端来说：

PutChar -> WriteDone -> PutChar -> WriteDone -> ...

系统发出一个写终端命令PutChar，模拟系统将直接向终端输出文件写入要写的内容，但是对Nachos 来说，整个写的过程并没有结束，只有当写终端中断来到后整个写过程才算结束。

2.1.5 磁盘设备模块实现机制分析

本模块主要在文件文件 disk.cc 和 disk.h 中实现。

磁盘设备模拟了一个物理磁盘。Nachos 用宿主机中的一个文件来模拟一个单面物理磁盘，该磁盘由道组成，每个道由扇区组成，而每个扇区的大小是固定的。和实际的物理磁盘一样，Nachos 以扇区为物理读取/写入的最小单位，每个扇区有唯一的扇区地址，具体的计算方法是：

track * SectorsPerTrack + offset

该物理磁盘是一个异步的物理磁盘，同终端设备和网络设备一样，当系统发出读磁盘的

请求，立即返回，只有具体的磁盘终端到来的时候，整个过程才算结束。

Nachos 对物理磁盘的模拟和对网络、终端等的模拟非常类似，所采用的手段也很类似。这里就不详细叙述，需要说明的有以下几点：

1. 和其它的模拟不同的是，每次磁盘请求到磁盘中断发生之间的时间间隔是不一样的，这取决于两次磁盘访问磁道和扇区的距离。
2. 为了和实际情况更加接近，Nachos 的物理磁盘设置有 trackbuffer 高速缓冲区。其中存放的是最后一次访问的磁道中的所有内容。这样如果相邻的两次磁盘读访问是同一个磁道，可以直接从 trackbuffer 中读取扇区的内容，而不必要进行真正的磁盘读访问，当然写磁盘则多了向 trackbuffer 写入的步骤。
3. 磁盘模拟文件开头四个字节的值为 MagicNumber，其作用是为了不让 Nachos 磁盘模拟文件同其它文件混淆。

对于操作系统的实现来说，这部分内容并不是很重要。但是当我们认识到对于磁盘的访问会影响到系统的效率时，可能会重新设计文件系统，让系统不要在磁头的移动中无谓地浪费时间。比如在有些文件系统的设计中，采用了多 inode 区。目的是使一个文件的 inode 同文件内容所在的磁盘扇区比较接近，从而减少磁头移动的时间。

2.1.6 系统运行情况统计

在本节的最后部分，我们要说明的是对 Nachos 运行情况进行统计的类 Statistics。这并不属于机器模拟的一部分，但是为了帮助读者了解自己设计的操作系统的各种运行情况。Statistics 类中包含的各种统计项是非常有价值的。

Statistics 类中主要包括：Nachos 运行的时间；Nachos 在 Idle 态的时间；Nachos 在系统态运行的时间；Nachos 发出的读磁盘请求次数；Nachos 发出的写磁盘请求次数；Nachos 读取的终端字符数；Nachos 输出的字符数；等等。

2.2 Nachos 中的进程/线程管理

2.2.1 相关知识点回顾

2.2.1.1 进程

- 进程定义及特征
- 进程状态及转换
- 进程的组成：程序段、数据段、堆栈、进程控制块
- 进程管理：创建、调度、阻塞、撤销、挂起
- 进程的同步与互斥

2.2.1.2 线程

- 线程概念
- 线程与进程的联系

2.2.2 功能概述

Nachos提供了一个基本的线程管理系统和一个同步互斥机制信号量的实现，定义了锁和条件变量等另两种同步互斥机制的函数接口。利用Nachos，可以一条指令一条指令地跟踪线程切换的过程，深入了解在线程切换的过程，也可以在它上面编写并发性程序，并测试这些

程序。

这个简化的线程管理与实际的进程管理的不同在于：

- 不存在系统中所有线程的列表

  在一般的操作系统中，进程的数目总是有限的，但是Nachos 中的线程数目可以是无限的（当然，用户进程的数目应该也是有限的。当虚拟机内存以及虚拟内存都耗尽时，就不能产生新的用户线程）。这是因为，线程的控制结构和系统线程的运行是占用宿主机的。能够开多少线程完全由宿主机条件限制，理论上是无限的。

- 线程的调度比较简单

  在启动了时钟中断的情况下，当时钟中断到来时，如果就绪线程队列中有就绪线程，就必须进行线程切换；当没有启动时钟中断的情况下，Nachos 使用非抢占式调度。

- 没有实现父子线程的关系

  可以说，所有的Nachos 线程都是Nachos 的一个子线程。但是Nachos 线程之间的父子关系没有实现。这样产生的混乱体现在线程的空间释放上，一个线程空间的释放是由下一个被切换的线程也即兄弟线程进行的，而这两个线程可以是没有任何关系的。这样的情况对以后进一步进行系统扩充是不利的。

2.2.3 具体模块实现介绍

2.2.3.1工具模块（文件list.cc list.h utility.cc utility.h）

功能：定义了一个链表结构及其操作

2.2.3.2线程启动和调度模块（文件switch.s switch.h）

2.2.3.2.1功能概述

在Nachos 中，线程是最小的调度单位，在同一时间内，可以有几个线程处于就绪状态。Nachos 的线程切换借助于宿主机的正文切换，由于这部分内容与机器密切相关，而且直接同宿主机的寄存器进行交道，所以这部分是用汇编来实现的。由于Nachos 可以运行在多种机器上，不同机器的寄存器数目和作用不一定相同，所以在switch.s 中针对不同的机器进行了不同的处理。如果需要将Nachos 移植到其它机器上，就需要修改这部分的内容。

2.2.3.2.2核心函数介绍

- ThreadRoot函数

  语法： ThreadRoot (int InitialPC, int InitialArg, int WhenDonePC, int StartupPC)

  参数： InitialPC 指明新生成线程的入口函数地址

  InitialArg 是该入口函数的参数

  StartupPC 是在运行该线程是需要作的一些初始化工作，比如开中断

  WhenDonePC 是当该线程运行结束时需要作的一些后续工作

  功能： Nachos 中，除了main 线程外，所有其它线程都是从ThreadRoot入口运行的。在Nachos 的源代码中，没有任何一个函数和方法显式地调用ThreadRoot 函数，ThreadRoot 函数只有在线程切换时才被调用到。由ThreadRoot 入口可以转而运行线程所需要运行的函数，从而达到生成线程的目的。

- SWITCH函数

  语法： void SWITCH (Thread *t1, Thread *t2)

  参数： t1 是原运行线程指针

  t2 是需要切换到的线程指针

  功能： 完成线程切换

2.2.3.3线程定义模块（文件thread.cc thread.h）

2.2.3.3.1功能概述

定义了相当于线程控制块的Thread 类，与PCB (ProcessControl Block) 有相似之处。

Thread 线程控制类较PCB 为简单的多，它没有线程标识 (pid)、实际用户标识 (uid)等和线程操作不是非常有联系的部分，也没有将PCB 分成proc 结构和user 结构。这是因为一个Nachos 线程是在宿主机上运行的。无论是系统线程和用户进程，Thread 线程控制类的实例都生成在宿主机而不是生成在虚拟机上。所以不存在实际操作系统中proc 结构常驻内存，而user 结构可以存放在盘交换区上的情况，将原有的两个结构合并是Nachos 作的一种简化。Nachos 对线程的另一个简化是每个线程栈段的大小是固定的，为4096-5 个字 (word)，而且是不能动态扩展的。所以Nachos 中的系统线程中不能使用很大的栈空间，比如：

void foo () { int buff[10000]; ...}

可能会不能正常执行，如果需要使用很大空间，可以在Nachos 的运行堆中申请：

void foo () { int *buf = new int[10000]; ...}

如果系统线程需要使用的栈空间大于规定栈空间的大小，可以修改StackSize 宏定义。

2.2.3.3.2核心函数介绍

- Fork 方法

语法： void Fork (VoidFunctionPtr func, int arg)

参数： func：新线程运行的函数；arg：func 函数的参数

功能： 线程初始化之后将线程设置成可运行的。

- StackAllocate 方法

语法： void StackAllocate (VoidFunctionPtr func, int arg)

参数： func：新线程运行的函数；arg：func 函数的参数

功能： 为一个新线程申请栈空间，并设置好准备运行线程的条件。

- Yield 方法

语法： void Yield ()

参数： 无

功能： 当前运行强制切换到另一个就绪线程运行

- Sleep 方法

语法： void Sleep ()

参数： 无

功能： 线程由于某种原因进入阻塞状态等待一个事件的发生（信号量的V 操作、开锁或者条件变量的设置）。当这些条件得到满足，该线程又可以恢复就绪状态。

2.2.3.4线程调度算法模块（文件scheduler.cc scheduler.h）

2.2.3.4.1功能概述

该模块的作用是进行线程的调度。在Nachos 系统中，有一个线程就绪队列，其中是所有就绪线程。调度算法非常简单，就是取出第一个放在处理机运行即可。由于Nachos 中线程没有优先级，所以线程就绪队列是没有优先级的。

2.2.3.4.2核心函数介绍

- Run方法

语法： void Run (Thread *nextThread)

参数： nextThread：需要切换运行的线程

功能： 当前运行强制切换到nextThread 就绪线程运行

2.2.3.5 Nachos主控模块（文件main.cc system.cc system.h）

2.2.3.5.1功能概述

该模块是整个Nachos 系统的入口，它分析了Nachos 的命令行参数，根据不同的选项进行不同功能的初始化设置。选项的设置如下所示：

一般选项：

-d: 显示特定的调试信息
-rs: 使得线程可以随机切换
-z: 打印版权信息
和用户进程有关的选项：
-s: 使用户进程进入单步调试模式
-x: 执行一个用户程序
-c: 测试终端输入输出
和文件系统有关的选项：
-f: 格式化模拟磁盘
-cp: 将一个文件从宿主机拷贝到Nachos 模拟磁盘上
-p: 将Nachos 磁盘上的文件显示出来
-r: 将一个文件从Nachos 模拟磁盘上删除
-l: 列出Nachos 模拟磁盘上的文件
-D: 打印出Nachos 文件系统的内容
-t: 测试Nachos 文件系统的效率
和网络有关的选项：
-n: 设置网络的可靠度（在0-1 之间的一个小数）
-m: 设置自己的HostID
-o: 执行网络测试程序
2.2.3.5.2核心函数介绍

- main

```
int main (int argc, char **argv)
{
对命令行参数进行处理，并且初始化相应的功能
currentThread -> Finish ();
return (0); // 此行执行不到。
}
```

在main 函数的最后，是currentThread->Finish()语句。为什么不直接退出呢？这是因为Nachos是在宿主机上运行的一个普通的进程，当main函数退出时，整个占用的空间要释放，进程也相应的结束。但是实际上在Nachos 中, main函数的结束并不能代表系统的结束，因为可能还有其它的就绪线程。所以在这里我们只是将main 函数作为Nachos 中一个特殊线程进行处理，该线程结束只是作为一个线程的结束，系统并不会退出。

2.3 Nachos 中的文件系统管理

2.3.1 相关知识点回顾

2.3.1.1 文件

- 文件结构：逻辑结构、物理结构
- 文件访问方式：顺序访问、随机访问
- 文件类型、文件属性
- 文件操作

2.3.1.2 目录

- 目录结构：单级目录、两级目录、多级 目录

- 路径名
- 目录项
- 工作目录

2.3.2 功能概述

Nachos 是在其模拟磁盘上实现的文件系统。它包括一般文件系统的所有的特性，可以：

- 按照用户的要求创建文件和删除文件
- 按照用户要求对文件进行读写操作
- 对存放文件的存储空间进行管理，为各个文件自动分配必要的物理存储空间，并为文件的逻辑结构以及它在存储空间中的物理位置建立映照关系。
- 用户只需要通过文件名就可以对文件进行存放，文件的物理组织对用户是透明的。

相比实际的操作系统，存在的不足：

- 现有的文件系统没有实现互斥访问，所以每次只允许一个线程访问文件系统。
- 只有一级目录，也就是只有根目录，所有的文件都在根目录下。而且根目录中可以存放的文件数是有限的。
- 文件索引结构采用的都是直接索引，所以Nachos 的最大文件长度不能大于4K
- 必须在文件生成时创建索引表。所以Nachos 在创建一个文件时，必须给出文件的大小；而且当文件生成后，就不能改变文件的大小。
- 目前该文件系统没有Cache 机制
- 健壮性不够强。当正在使用文件系统时，如果突然系统中断，文件系统中的内容可能不保证正确。

2.3.3 具体模块实现介绍

2.3.3.1同步磁盘（文件synchdisk.cc synchdisk.h）

2.3.3.1.1 功能概述

和其它设备一样，Nachos 模拟的磁盘是异步设备。当发出访问磁盘的请求后立刻返回，当从磁盘读出或写入数据结束后，发出磁盘中断，说明一次磁盘访问真正结束。

Nachos 是一个多线程的系统，如果多个线程同时对磁盘进行访问，会引起系统的混乱。所以必须作出这样的限制：

- 同时只能有一个线程访问磁盘
- 当发出磁盘访问请求后，必须等待访问的真正结束。

这两个限制就是实现同步磁盘的目的。此模块定义了SynchDisk类，模拟同步磁盘的实现。

2.3.3.2位图模块（文件bitmap.cc bitmap.h）

2.3.3.2.1功能概述

在Nachos 的文件系统中，是通过位图来管理空闲块的。Nachos 的物理磁盘是以扇区为访问单位的，将扇区从0 开始编号。所谓位图管理，就是将这些编号填入一张表，表中为0的地方说明该扇区没有被占用，而非0 位置说明该扇区已被占用。此模块定义了BitMap 类来模拟位图管理。

2.3.3.3文件系统模块（文件filesys.cc filesys.h）

2.3.3.3.1功能概述

此模块模拟了创建文件、删除文件、打开文件操作。在Nachos 中，实现了两套文件系统，它们对外接口是完全一样的：一套称作为FILESYS_STUB，它是建立在UNIX 文件系统之上的，而不使用Nachos 的模拟磁盘，它主要用于先实现了用户程序和虚拟内存，然后再着手增强文件系统的功能；另一套是Nachos 的文件系统，它是实现在Nachos 的虚拟磁盘上的。当整个系统完成之后，只能使用第二套文件系统的实现。

2.3.3.3.2核心函数介绍

- Create方法

语法： bool Create (char *name, int initialSize)
参数： name：需要创建的文件名
initialSize: 需要创建的文件的初始大小
功能： 在当前的文件系统中创建一个固定大小的文件

- Open方法

语法： OpenFile * Open (char *name)
参数： name：需要打开的文件名
功能： 在当前的文件系统中打开一个已有的文件

- Remove方法

语法： bool Remove (char *name)
参数： name：需要删除的文件名
功能： 在当前的文件系统中删除一个已有的文件

2.3.3.4文件头模块（文件filehdr.cc filehdr.h）

2.3.3.4.1功能概述

文件头实际上就是UNIX 文件系统中所说的inode结构，它给出一个文件除了文件名之外的所有属性，包括文件长度、地址索引表等等（文件名属性在目录中给出）。所谓索引表，就是文件的逻辑地址和实际的物理地址的对应关系。通过文件头可以获取文件的所有信息。Nachos 的文件头可以存放在磁盘上，也可以存放在宿主机内存中，在磁盘上存放时一个文件头占用一个独立的扇区。此模块定义了文件头类，具体实现请查看源代码。

2.3.3.5打开文件结构（文件openfile.cc openfile.h）

2.3.3.5.1功能概述

该模块定义了一个打开文件控制结构。当用户打开了一个文件时，系统即为其产生一个打开文件控制结构，以后用户对该文件的访问都可以通过该结构。打开文件控制结构中的对文件操作的方法同UNIX 操作系统中的系统调用。

2.3.3.5.2核心函数介绍

针对FileSystem 结构中的两套实现，这里给出的函数皆属于建立立在Nachos 上的一套实现。

- Read

语法：int Read(char *into, int numBytes)
参数： into：读出内容存放的缓冲
numBytes：需要读出的字节数
功能： 从文件中读出numByte 到into 缓冲

- Write

语法： int Write(char *from, int numBytes)
参数： from：存放需写入内容的缓冲
numBytes：需写入的字节数
功能： 将from 缓冲中写入numBytes 个字节到文件中

- Seek

语法： void Seek(int position)
参数： position：要到达的文件位置
功能： 从文件头开始，移动文件位置指针至position处

- ReadAt方法

语法： int ReadAt (char *into, int numBytes, int position)
参数： into：读出内容存放的缓冲
numBytes：需要读出的字节数
position：需读出内容的开始位置
功能： 将从position 开始的numBytes 读入into 缓冲

- WriteAt方法

语法： int WriteAt (char *from, int numBytes, int position)
参数： from：存放需写入内容的缓冲
numBytes：需写入的字节数
position：需写入内容的开始位置
功能： 将from缓冲中的numberBytes 字节从position 开始的位置写入文件

注：实际上，对文件的一次写操作应该是原子操作，否则会出现两个线程交叉写的状况。比如A线程和B 线程都需要对扇区a 和b 进行修改。工作过程如下：A（写a）-> 线程切换 ->B（写a） ->B（写b）-> 线程切换 ->A（写b）这样扇区b 中的内容是A 线程写的，而扇区a 的内容是B 线程写的。这样，数据的一致性不能得到保证。但是目前Nachos 没有对此进行处理。

2.3.3.6目录模块（文件directory.cc directory.h）

2.3.3.6.1功能概述

每个目录都对应一个目录文件，目录文件由目录项组成，目录项使得字符形式的文件名与实际文件的文件头相对应，这样用户就能方便地通过文件名来访问文件。此模块模拟了单级目录相关操作，具体实现请参考源代码。

2.4 Nachos 中的存储系统管理

2.4.1 相关知识点回顾

2.4.1.1 连续分配存储管理方式

- 分配方式：单一连续分配、固定分区、动态分区
- 分区分配数据结构、分区分配算法

2.4.1.2 分页存储管理方式

- 页表、地址变换
- 两级页表、多级页表
- 反置页表、快表

2.4.1.3 分段存储管理方式

- 段表、地址变换

2.4.1.4 段页式存储管理方式

2.4.1.5 虚拟存储器

- 虚拟存储器定义及特征
- 缺页中断
- 页面分配算法、页面置换算法

2.4.2 功能概述

Nachos 目前实现的存储系统，一次只能执行一个用户程序，且可执行程序的大小必须

小于物理内存，否则无法正常执行。 要达到对现代操作系统中存储系统的支持多道程序、程序可用空间不受限制的目标还是有差距的，有待改进。

2.4.3 具体模块实现介绍

2.4.3.1 相关文件介绍

machine.h： 包含了内存的相关定义
machine.cc： 包含了对内存和页表的初始化
translate.h： 定义了页表结构
translate.cc： 包含了地址转换的实现以及读、写内存的操作
address.cc： 包含了用户程序页表的操作

2.4.3.2 函数介绍

- ReadMem函数

语法：bool ReadMem(int addr, int size, int* value)
参数：addr：用户程序逻辑地址；
size：需要读出的字节数；
value：读出的内容暂存地
功能： 从用户逻辑地址读出size 个字节，转换成相应的类型，存放在value 所指向的空间

- WriteMem函数

语法： bool WriteMem(int addr, int size, int value)
参数： addr：用户程序逻辑地址；
size：需要写入的字节数；
value：需要写入的内容
功能： 将value 表示的数值根据size 大小转换成相应的机器类型存放在add 用户逻辑地址

- Translate函数

语法： Exception Translate (int virtAddr, int *physAddr, int size, bool writing)
参数： virtAddr：用户程序的逻辑地址
physAddr：转换后的实际地址
size：数据类型的大小
writing：读/写内存标志
功能： 将用户的逻辑地址转换成实际的物理地址，同时需要检查对齐。

2.5 Nachos 中的网络系统管理

2.5.1 相关知识点回顾

- 网络操作系统
- ISO 七层协议
- Socket
- 网络端口号
- 邮局协议

2.5.2 现有功能分析

Nachos 网络系统管理模块模拟了和一个 Nachos 虚拟机相连接的物理网络，该模拟网络有如下特点：

- 数据报发送是有序的；
- 数据报的发送可能丢失，不是可靠的；但是数据报一旦到达目标节点，其内容一定是正确的，所以并不需要校验；
- 网络之间的连接是全互连的。

每个 Nachos 虚拟机有唯一的一个网络地址，该地址可以在 Nachos 启动时在命令行中给出。Nachos 实现时用一个 SOCKET 文件模拟了和自己相连接的网络部分。该文件名的格式是 SOCKET_网络地址。

一个网络数据包由数据包头和数据部分两部分组成。头部的内容是数据包发送者和接收者的地址信息和数据报文内容的长度。每个数据包的长度是固定的，为 64 个字节。

Nachos 在基本物理网络的基础上建立了一个邮局协议。在每个 Nachos 模拟机上，有一个由邮箱组成的邮局，它负责监控与其相联的网络是否有可以接收的邮件。如果有，就接收下来分发给特定的邮箱；邮局对象同时还负责将本机的邮件发送给其它 Nachos 模拟机。由于 Nachos 是一个多线程的系统，每个线程都可以独立地利用网络，这些线程在使用网络的时候如何不互相干扰呢？这里实际上借用了网络端口号的概念，每个线程需要进行网络通讯，就必须和一个网络端口建立联系，操作系统发现有网络数据包到达时，才知道发送给哪个线程。这里的网络端口就是邮箱。

2.5.3 具体模块实现介绍

2.5.3.1 物理网络的模拟

Nachos 的物理网络主要由 machine 目录下的 network.cc 和 network.h 文件中的 PacketHeader 类和 Network 类进行模拟。和机器模拟部分终端模拟实现相类似，两个内部函数 NetworkReadPoll 和 NetworkSendDone 作为网络读写的中断处理函数，这两个内部函数通过 CheckPktAvail 和 SendDone 两个内部方法调用真正的中断处理函数。这样 Nachos 可以以一定的时间间隔检查是否存在发给自己的数据包。当然在这些网络中断处理程序中，还包括了一些统计的工作。

系统的网络读写操作同样有严格的工作顺序，对网络读来说：

CheckPktAvail -> Receive -> CheckPktAvail -> Receive -> CheckPktAvail -> Receive ->...

系统通过定期的网络读中断来判断是否有发给自己的数据包，如果有则读出；如果没有，下一次读网络中断继续判断。读出的内容将一直保留到 Receive 将其读走。

对写网络来说：

Send -> SendDone -> Send -> SendDone -> Send -> SendDone -> Send -> SendDone -> ...

系统发出一个向网络发送数据包的指令 Send，模拟系统将直接向和目标机相连接的网络模拟文件发送数据包，但是对 Nachos 来说，整个写的过程并没有结束，只有当写网络中断来到后整个写过程才算结束。

其中核心函数 Send 的主要功能就是向一个目标地址发送一个数据包。它的函数定义是：void Send (PacketHeader hdr, char * data)。其中参数 hdr 表示发送数据包头，data 表示发送数

据包数据内容。

在该模拟网络中，有可能发生这样的现象，就是发送方调用Send 函数发送一个数据报，由于发送方此时只有等待到网络发送中断到达才可能确定此次发送成功。但是在调用Send 函数到网络发送中断到达之前，与接收节点相连的网络Socket 文件中已经有内容，接收节点完全有可能探测到数据报内容的存在并且将内容读走。所以目前的机制存在网络数据保送和接收同步问题的隐患。

需要说明为什么用socket 机制来模拟网络，而不选用普通的文件？我们知道，通过socket和文件连接，这样的文件拥有这样的性质：已经读走的内容不可再现。于是对于这样的文件只需要不断地从文件头部读取内容即可。与此相类似，还可以通过使用有名管道文件达到同样的效果。

2.5.3.2 邮局协议的模拟

在 network 目录下的 post.cc 和 post.h 文件中，Nachos 实现了在基本物理网络基础上的模拟邮局协议，而 nettest.cc 文件对该协议的运作做了一个简单测试。

前文提到一个网络数据报由两个部分组成：数据报头和数据部分。报头部分包括数据报的收发地址，以及数据报的长度。实现 PostOffice 协议时，网络数据报又有两个部分组成：邮件的头部和邮件的内容。而具体的邮件头部结构和邮件结构在类 MailHeader 和类 Mail 中实现。

邮箱中存放的实际上是一个邮件的链表，邮局负责将接收的邮件放入邮箱，系统上特定的线程到邮局中特定的邮箱去取。其中要注意到邮箱的操作是互斥的，而邮箱的结构主要在类 Mail 中实现。

网络是一个异步设备，向网络发出请求后不需要等待其结束有可以返回，当网络处理的中断到来时，整个处理结束。在邮局对象生成方法中，生成了一个 Demon 线程，专门同步监测传给自己的邮件，分析邮件头部并将邮件放入特定的邮箱中。邮局的结构主要在类 PostOffice 中实现。其中比较核心的两个函数功能说明如下：

- PostalDelivery 函数

语法： void PostalDelivery ()

参数： 无

功能： 监测网络上是否有邮件到来，并将到来的邮件分发给各个邮箱。这就是邮局生成的demon 线程执行的程序。

- Send 方法

语法： void Send (PacketHeader pktHdr, MailHeader mailHdr, char *data)

参数： 无

功能： 将邮件发出去，接收邮件的地址由pktHdr 中的接收地址和mailHdr 中的接收邮箱决定。

第三章 Nachos 平台上机实践项目设置

3.1 实践项目 1：Nachos 系统调用

3.1.1 上机实践具体要求

【背景描述】

操作系统的重要的功能之一就是在用户和计算机硬件之间提供界面，向用户屏蔽底层的硬件细节，为用户提供交互的接口。界面大致可分为两类，一类是字符终端界面，比如 Linux、Unix 等操作系统的 Shell，Microsoft Windows 的 Dos 环境等；另一类是图形界面，比如 Linux、Unix 的 XWindows 和 Microsoft Windows 的桌面管理系统等。这些界面往往通过系统调用使用操作系统提供的服务。Nachos 平台既没有提供字符终端界面，也没有提供图形界面，Nacos 对运行于其上的程序以“Nachos -x Program”的方式运行，与实际的操作系统使用体验有较大的差距。

【实践要求】

本实践项目希望通过修改 Nachos 系统平台的底层源代码来实现以下目标：

1. 为 Nachos 增加终端输入输出系统调用；
2. 在终端输入输出系统调用的基础上，为 Nachos 实现一个 Shell（字符终端）界面，通过该界面，用户可以实现类似于 Linux Shell 的大部分功能；
3. Shell 界面上以“--”作为 Shell 命令输入提示符，输入命令后以“ENTER”作为输入结束键。

【提交形式】

针对以上实践要求，实践小组必须提交符合以下规范的实践成果。

1. 项目实践文档

   一份 Word 或 PDF 格式的文档，在文档中说明以下内容：

   - 实践小组的人员组成、各自的分工；
   - 对实践要求的满足程度，本项目有 2 项要求，请注明共满足了几项。
   - 对实践过程的详细说明，针对已经满足的实践要求，采用了何种算法或思想，对 Nachos 平台的哪些代码进行了什么样的修改。

2. 修改后的 Nachos 平台源码备份

   本文档描述的实践环境均基于 VMWare 虚拟机，在第一章中已经描述了如何将 Nachos 源码包备份为可在 Windows 环境下传递的磁盘文件。请将修改后的 Nachos 源码打包备份提交。

3. 内部评测过程描述文档

   在完成实践后，需要进行适当的内部评测来验证实践的效果。请以文档的形式说明设计了什么样的评测过程，在评测过程中发现了哪些问题。

3.1.2 实践的过程和步骤

本项目在实践的过程中需要注意以下要点：

1. 研究 code/shell 下的代码：

   Nachos 所给的源代码的 code/shell 目录下的源代码实现了一个 shell，该 shell 利用 C/C++对 Linux Shell 命令的调用支持(execl (SHELL, SHELL, "-c", exec_name, NULL))实现了一个 Shell 的 Wrapper，使得一个函数(nachos_syscall.c 中的 Exec(char* exec_name))可以实现所有的 Linux Shell 命令。在 Nachos 内核中，对具体的 Shell 命令实现可参考其实现方法，不要求对每个 Shell 命令都去具体的实现。

2. 研究 Nachos 的系统调用机制：

   研究 Nachos 的系统调用机制，在 Nachos 内核中实现关键系统调用，以支持 test/shell.c 通过交叉编译所生成程序的运行。通过分析 test/shell.c 可知，这些关键系统调用包括：Write，Read，Exec，Join。假设 test/shell.c 通过交叉编译所生成的程序为 shell.noff，如果实现正确，则运行“nachos –x shell.noff”时会在 Nachos 平台上为用户提供提供一个满足要求的 Shell。

3.1.3 实践结果验证方式

针对本项目实践，采用的验证方式如下：

1. 阅读项目文档，考察实践过程中所采用的思想和方法

   项目实践小组必须提交项目文档，项目文档的详细程度直接决定了对实现过程的评价。实践指导教师将首先考察项目文档中描述的实现过程，特别是对所采用算法以及对 Nachos 修改过程的描述。如果描述过程中存在严重的、理论上不可正确运行的错误，那么将不会进一步检验。

2. 源代码浏览

   在理解了实践小组设计思想的基础上，实践指导教师将阅读实践小组提交的 Nachos 源代码，检验源代码的修改状况是否与文档相符。

3. Nachos 编译运行以及验证

   在通过以上两步检验后，实践指导教师将编译运行实践小组提交的 Nachos 源码，并通过运行实现编写好的用户程序来验证 Nachos 的真实运行过程是否满足要求。

3.2 实践项目 2：Nachos 的线程管理模块升级

3.2.1 上机实践具体要求

【背景描述】

在 Nachos 平台中，线程是作业调度的基本单位，除了线程和进程的概念差别之外，可直接运用操作系统教学课程中关于进程管理的知识。Nachos 平台所使用的是非抢占式调度，线程一旦占用 CPU，就会一直运行到结束或者被阻塞（等待 I/O 事件）；Nachos 平台中的线程数据结构定义非常简单，并无用户 ID、线程 ID 等数据成员，也就是说，无法基于线程的

ID 来实现通信、同步互斥等机制。Nachos 平台中并无全局性的线程管理机制，并未限制线程的数目，也无法了解有多少线程存在。

【实践要求】

本实习项目希望通过修改 Nachos 系统平台的底层源代码来实现以下目标：

1. 扩充 Nachos 中线程的管理模式，目前 Nachos 并不限制线程的个数，而是依靠宿主机（我们的实验环境中宿主机就是指使用 VMWare 创建的 Linux 虚拟机，预设内存为 256M）的内存来保存线程的信息。请在 Nachos 中增加对线程数量的限制，使得 Nachos 中最多能够同时存在 128 个用户线程。
2. 修改扩充 Nachos 中线程调度的机制，将其改变为遵循“优先级调度”的抢占式调度。在时钟中断发生时，检查处于“就绪状态”的等待线程中是否存在优先级高于目前占用 CPU 线程的线程，如果存在这样的线程就使其抢占 CPU，如果不存在则不发生线程切换。
3. （可选）为 Nachos 的线程管理模块增加通过“公共信箱”进行通信的机制，由 Nachos 系统核心维护一个预定大小的 Mailbox（4K），规定每一封信的大小为 128 字节，实现两个用户进程之间通过这个“公共信箱”进行通信。

【提交形式】

针对以上实践要求，实践小组必须提交符合以下规范的实践成果。

1. 项目实践文档

   一份 Word 或 PDF 格式的文档，在文档中说明以下内容：

   - 实践小组的人员组成、各自的分工；
   - 对实践要求的满足程度，本项目有四项要求，请注明共满足了几项。
   - 对实践过程的详细说明，针对已经满足的实践要求，采用了何种算法或思想，对 Nachos 平台的哪些代码进行了什么样的修改。

2. 修改后的 Nachos 平台源码备份

   本文档描述的实践环境均基于 VMWare 虚拟机，在第一章中已经描述了如何将 Nachos 源码包备份为可在 Windows 环境下传递的磁盘文件。请将修改后的 Nachos 源码打包备份提交。

3. 内部评测过程描述文档

   在完成实践后，需要进行适当的内部评测来验证实践的效果。请以文档的形式说明设计了什么样的评测过程，在评测过程中发现了哪些问题。

3.2.2 实践的过程和步骤

本项目在实践的过程中需要注意以下要点：

1. 数据结构的修改和维护：

   线程核心管理机制的升级首先依赖于对线程数据结构的修改。例如实现“时间片轮

转”的线程调度机制，必须首先在线程数据结构中增加“已使用时间片计数”这样一个变量。同时在“线程创建”、“时钟中断”、“线程切换”等代码中，增加对这个数据成员的维护性代码。

2. 线程管理机制所依赖的细节技术处理：

   线程管理机制的运行过程非常复杂，实践过程中应把握好关键的处理步骤：

   - 时钟中断处理：在 Nachos 中，时钟中断处理包含了引起线程调度的代码，修改线程调度机制必须调整这部分代码。
   - 线程上下文切换：在发生线程切换时，必须妥善保存线程的上下文。由于修改了线程的底层数据结构，因此上下文切换的代码也必须修改。
   - 线程调度：这是本实践项目的核心代码，应认真阅读后修改。

3.2.3 实践结果验证方式

针对本项目实践，采用的验证方式如下：

1. 阅读项目文档，考察实践过程中所采用的思想和方法

   项目实践小组必须提交项目文档，项目文档的详细程度直接决定了对实现过程的评价。实践指导教师将首先考察项目文档中描述的实现过程，特别是对所采用算法以及对 Nachos 修改过程的描述。如果描述过程中存在严重的、理论上不可正确运行的错误，那么将不会进一步检验。

2. 源代码浏览

   在理解了实践小组设计思想的基础上，实践指导教师将阅读实践小组提交的 Nachos 源代码，检验源代码的修改状况是否与文档相符。

3. Nachos 编译运行以及验证

   在通过以上两步检验后，实践指导教师将编译运行实践小组提交的 Nachos 源码，并通过运行实现编写好的用户程序来验证 Nachos 的真实运行过程是否满足要求。

3.3 实践项目 3：Nachos 的文件管理模块升级

3.3.1 上机实践具体要求

【背景描述】

文件系统是负责管理和存取文件信息的子系统，是操作系统中与用户关系最为密切的部分。目前Nachos的文件系统中实现的是单级目录，目录中的文件个数有限。文件的索引结构采用了直接索引，使得文件的最大长度不能大于4K。文件大小在创建时指定，并不是根据文件内容动态分配，而且文件大小不可变。描述文件的信息有限，并无文件类型、文件创建时间之类的信息。没有实现同步操作，多个进程同时访问磁盘位图、目录文件等临界资源时就会出现错误。

【实践要求】

本实习项目希望通过修改 Nachos 系统平台的底层源代码来实现以下目标：

1. 增加对 Nachos 现有文件系统的多线程访问机制，目前的 Nachos 系统同时只允许一个线程访问文件系统，请实现多个线程可同时访问文件系统的机制。

2. 扩充 Nachos 的目录体系，使其能够支持多级目录（最多 4 级）。每个目录下的文件个数没有限制。
3. 扩充 Nachos 下的文件管理功能，也就是编写几个完整的系统调用函数，它们可以实现文件的更名、删除、拷贝。
4. （可选），更改 Nachos 中文件空间分配机制，使其能够支持最大 2MB 的文件，目前 Nachos 只能支持最大不超过 4KB 的文件。

【提交形式】

针对以上实践要求，实践小组必须提交符合以下规范的实践成果。

1. 项目实践文档

   一份 Word 或 PDF 格式的文档，在文档中说明以下内容：

   - 实践小组的人员组成、各自的分工；
   - 对实践要求的满足程度，本项目有五项要求，请注明共满足了几项；
   - 对实践过程的详细说明，针对已经满足的实践要求，采用了何种算法或思想，对 Nachos 平台的哪些代码进行了什么样的修改。

2. 修改后的 Nachos 平台源码备份

   本文档描述的实践环境均基于 VMWare 虚拟机，在第一章中已经描述了如何将 Nachos 源码包备份为可在 Windows 环境下传递的磁盘文件。请将修改后的 Nachos 源码打包备份提交。

3. 内部评测过程描述文档

   在完成实践后，需要进行适当的内部评测来验证实践的效果。请以文档的形式说明设计了什么样的评测过程，在评测过程中发现了哪些问题。

3.3.2 实践的过程和步骤

本项目在实践的过程中需要注意以下要点：

1. 数据结构的修改和维护：

   文件管理的升级基于对原有 Nachos 数据结构的修改。对于增加文件的描述信息需对文件头结构进行简单修改。多级目录中可创建目录也可创建文件，应根据实际的文件类型初始化文件头信息。

2. 实现多级目录应注意的地方：

   - 理解目录文件的含义，每个目录对应一个文件，通过此文件可了解其子目录及父目录的信息，正是通过目录文件记录了目录结构。
   - Nachos 的目录模块中由于是单级目录且子目录项个数有限，因此目录文件大小是预先定义的，但是要达到实践要求，目录文件的大小应是根据内容确定的，且能改变。
   - 实现多级目录后，添加、删除目录项要根据具体的路径，因此对树的遍历要有

深刻的理解。

3. 为了实现文件长度无限，要采取混合索引分配方式，必须对此概念有所了解。

3.3.3 实践结果验证说明

针对本项目实践，采用的验证方式如下：

1. 阅读项目文档，考察实践过程中所采用的思想和方法

   项目实践小组必须提交项目文档，项目文档的详细程度直接决定了对实现过程的评价。实践指导教师将首先考察项目文档中描述的实现过程，特别是对所采用算法以及对 Nachos 修改过程的描述。如果描述过程中存在严重的、理论上不可正确运行的错误，那么将不会进一步检验。

2. 源代码浏览

   在理解了实践小组设计思想的基础上，实践指导教师将阅读实践小组提交的 Nachos 源代码，检验源代码的修改状况是否与文档相符。

3. Nachos 编译运行以及验证

   在通过以上两步检验后，实践指导教师将编译运行实践小组提交的 Nachos 源码，并通过运行实现编写好的用户程序来验证 Nachos 的真实运行过程是否满足要求。

3.4 实践项目 4：Nachos 的内存管理模块升级

3.4.1 上机实践具体要求

【背景描述】

目前Nachos实现的内存管理模块中，没有实现真正的虚拟内存，内存分配基于的单个用户程序，系统能够运行的用户程序大小也是有限制的，必须小于模拟的物理内存空间大小，否则出错。现代操作系统对内存管理的要求是支持多道程序，并且程序可用空间应是无限即需实现虚拟内存，还要有存储保护机制。

【实践要求】

本实习项目希望通过修改 Nachos 系统平台的底层源代码来实现以下目标：

1. 在 Nachos 中修改目前的内存分配方式，使得多个线程可以同时存在于内存之中，这些线程可以按照“优先级”的方式进行调度。
2. 编写一个虚拟的“分页式”存储管理机制，也就是说不需要实现真正的分页管理，只需要建立并维护一个内存页表，页面大小为 4K，当生成新的用户线程时，可以通过检索页表来为用户线程分配可用的页面号。
3. （可选）在实现内存页表管理的基础上，进一步实现“缺页中断”，这也是虚拟的，也就是说编写一个测试函数，这个测试函数可以作为监测线程运行，它会提示发生了缺页中断，然后检查内存页表并根据适当的缺页中断处理策略选择一个可以被替换的内存页面，只需要记录被替换的页面号即可。

【提交形式】

针对以上实践要求，实践小组必须提交符合以下规范的实践成果。

1. 项目实践文档

   一份 Word 或 PDF 格式的文档，在文档中说明以下内容：

   - 实践小组的人员组成、各自的分工；
   - 对实践要求的满足程度，本项目有四项要求，请注明共满足了几项；
   - 对实践过程的详细说明，针对已经满足的实践要求，采用了何种算法或思想，对 Nachos 平台的哪些代码进行了什么样的修改。

2. 修改后的 Nachos 平台源码备份

   本文档描述的实践环境均基于 VMWare 虚拟机，在第一章中已经描述了如何将 Nachos 源码包备份为可在 Windows 环境下传递的磁盘文件。请将修改后的 Nachos 源码打包备份提交。

3. 内部评测过程描述文档

   在完成实践后，需要进行适当的内部评测来验证实践的效果。请以文档的形式说明设计了什么样的评测过程，在评测过程中发现了哪些问题。

3.4.2 实践的过程和步骤

本项目在实践的过程中需要注意以下要点：

1. 关于多道程序的实现：

   Nachos 中的内存分配基于单道程序，要实现多道程序，必须了解内存的使用情况及如何分配内存，因此就要在原有程序中增加描述内存使用的数据结构，对原有的内存分配算法进行修改。

2. 关于虚拟内存的实现：

   - 当前 Nachos 中并不支持虚拟内存，要实现虚拟内存需对页表结构进行修改，添加缺页中断的处理以及实现置换算法。
   - 实现虚拟内存之后，对于较大程序，初次分配时只是将可保证其正常执行的部分程序调入内存，分析各种分配策略，选取合适的加以实现。
   - 分析各种置换算法，选取合适的加以实现，避免出现抖动现象。

3.4.3 实践结果验证说明

针对本项目实践，采用的验证方式如下：

1. 阅读项目文档，考察实践过程中所采用的思想和方法

   项目实践小组必须提交项目文档，项目文档的详细程度直接决定了对实现过程的评价。实践指导教师将首先考察项目文档中描述的实现过程，特别是对所采用算法以及对 Nachos 修改过程的描述。如果描述过程中存在严重的、理论上不可正确运行的错误，那么将不会进一步检验。

2. 源代码浏览

   在理解了实践小组设计思想的基础上，实践指导教师将阅读实践小组提交的 Nachos

源代码，检验源代码的修改状况是否与文档相符。

3. Nachos 编译运行以及验证

在通过以上两步检验后，实践指导教师将编译运行实践小组提交的 Nachos 源码，并通过运行实现编写好的用户程序来验证 Nachos 的真实运行过程是否满足要求。

第四章 附录

4.1 Unix 常用命令介绍

5.1.1 目录及文件操作命令

1) more

功能：显示文件内容

格式：more [选项] 文件 ...

选项：-c 显示文件之前先清屏

-n 行数 指定每屏显示的行数

+ 行号 从指定行号开始显示

2) rm

功能： 删除一个目录中的一个或多个文件或目录，它也可以将某个目录及其下的所有文件及子目录均删除。对于链接文件，只是断开了链接，原文件保持不变。

格式： rm [选项] 文件…

选项： - f 忽略不存在的文件，从不给出提示。

- r 指示 rm 将参数中列出的全部目录和子目录均递归地删除。

- i 进行交互式删除。

3) mv

功能： 为文件或目录改名或将文件由一个目录移入另一个目录中

格式： mv [选项] 源文件或目录 目标文件或目录

选项： - i 交互方式操作。目标文件或目录存在时询问用户是否继续执行操作。

-f 禁止交互操作。

4) cp

功能： 文件或目录的拷贝

格式： －r 源文件(source) 目的文件(target)

5) mkdir

功能： 创建一个目录

格式： mkdir [选项] dir-name

选项: -m 对新建目录设置存取权限。也可以用 chmod 命令设置。

-p 可以是一个路径名称。此时若路径中的某些目录尚不存在， 加上此选项后， 系统将自动建立好那些尚不存在的目录，即一次可建立多个目录。

6) rmdir

功能： 删除空目录

格式： rmdir [选项] dir-name

选项： 递归删除目录 dirname，当子目录删除后其父目录为空时，也一同被删除。

7) cd

功能： 改变工作目录

格式： cd [directory]

8) pwd

功能： 显示出当前工作目录的绝对路径。

格式: pwd

9) ls

功能： 其功能为列出目录的内容

格式: ls [选项] [目录或是文件]

选项： - a 显示指定目录下所有子目录与文件，包括隐藏文件。

- A 显示指定目录下所有子目录与文件，包括隐藏文件。但无“.”和 “..”。

- l 以长格式来显示文件的详细信息。每行列出的信息依次是： 文件类型与权限 链接数 文件属主 文件属组 文件大小 建立或最近修改的时间 名字

10) tar

功能： 可以为文件和目录创建档案

格式： tar [主选项+辅选项] 文件或者目录

主选项：c 创建新的档案文件

t 列出档案文件的内容

x 从档案文件中释放文件

辅助选项：

f 使用档案文件或设备，这个选项通常是必选的。

v 详细报告 tar 处理的文件信息。如无此选项，tar 不报告文件信息

z 用 gzip 来压缩/解压缩文件，加上该选项后可以将档案文件进行压缩

但还原时也一定要使用该选项进行解压缩。

5.1.2 设备管理命令

1) fdisk

命令 1：fdisk –l 功能：列出系统所有硬盘的信息

命令 2：fdisk -l 硬盘 功能：列出相应硬盘信息

命令 3：fdisk 硬盘 功能：可对相应硬盘添加、删除分区以及改变分区属性

2) mkfs

功能：格式化磁盘

格式：mkfs [-V] [-t fstype] [fs-options] filesys [blocks]

选项：-t fstype 指定建立某种档案系统

-c 在建立前检查是否有坏轨

示例：mkfs –t vfat /dev/hdb1

3) mount

功能：挂载某一设备成为某个目录名称

格式：mount [选项] 设备 目录

选项：-a 将 /etc/fstab 中定义的所有档案系统挂上。

示例：mount /dev/hdb1 /mnt/dev

4) umount

功能：将已安装的文件系统卸下

格式：umount 设备

示例：umount /dev/hdb1

5.1.3 系统及用户管理命令

1) shutdown –h now

功能：关闭系统

2) reboot

功能：重启系统

3) passwd

功能： 修改密码

格式： passwd 用户名

5.1.4 其他命令

1) man

功能：查看指令的使用方法

格式：man 指令名称

2) vi 文本编辑器

语法：vi 文件名，可对指定的文件进行编辑

可在 vi 中使用的命令：

Esc：按<Esc>键将返回命令模式，允许输入新命令。

r：替换一个字符。

R：无限制地在一行中替换。

i：插入模式。

dd：删除一行。

x：删除一个字符。

wq：写文件并退出 vi。

q：退出 vi，不存文件。

4.2 Nachos 的系统调用介绍

1) void Halt()

功能：停机

2) void Exit(int status)

功能：用户程序执行完毕

3) SpaceId Exec(char *name)

功能：执行 Nachos 的可执行文件，返回地址空间标识

4) int Join(SpaceId id)

功能：用户程序“id”执行完毕时返回，返回退出时的状态

5) void Create(char *name)

功能：创建文件，文件名在 name 中

6) OpenFileId Open(char *name)

功能：打开文件

7) void Write(char *buffer, int size, OpenFileId id)

功能：从 buffer 中向打开文件中写入 size 个字节

8) int Read(char *buffer, int size, OpenFileId id)

功能：将打开文件中的 size 个字节读入 buffer，返回实际读出的字节数

9) void Close(OpenFileId id)

功能：关闭文件

10) void Fork(void (*func)())

功能：在相同的地址空间中创建用于运行 func 的线程，并置其为当前线程

11) void Yield()

功能：放弃 cpu 给另一可执行线程，无论是否与其在同一地址空间