《操作系统》

实 验 指 导 手 册

授课教师：

实验指导教师：

教学对象： 二年级本科生

开课时间： 春季学期

北京邮电大学软件学院

2018年3月

第一部分 上机实验的指导思想和要求

一、上机实验的目的

《操作系统》是计算机及相关专业的一门重要专业课，其讲授内容及研究对象即操作系统则为整个计算机系统的核心组成部分和关键所在。课程要求理解操作系统的基本概念、原理，掌握操作系统设计方法与实现技术，能够运用操作系统原理、方法与技术分析问题和解决问题。同时，掌握操作系统原理、熟悉操作系统的使用是各层次计算机软硬件开发人员必不可少的基本技能。

由于操作系统在整个计算机系统中的核心地位及其特性，学生对操作系统的认识往往只局限于简单的操作系统配置和使用，这与本课程的教学和考查目标有很大差距。为了让学生深刻理解操作系统的内部机制和系统结构，清楚操作系统的资源管理的主要过程，进而理解和掌握操作系统的设计方法和实现技术，就必须进行上级实验。具体的说，上机实验的目的包括：

（1）了解和分析操作系统的系统结构。通过分析操作系统，特别是Linux系统的内核源代码，了解操作系统的内部机制和体系结构，将教材中的抽象的算法、原理转换为具体的程序和代码。

（2）理解和掌握操作系统的基本概念、原理和算法。通过编写程序实现操作系统基本算法，深刻理解这些算法的前因后果，理解设计这些算法的目的，理解算法的运转过程。

（3）理解操作系统中资源管理的过程。通过编写程序及分析系统源代码，理解系统的运行过程，尤其是资源分配和管理过程。

（4）能够运用所学知识分析问题和解决问题。通过上机实验加深对操作系统的体系结构和运行过程的全面了解和理解，进而掌握运用操作系统的基本概念、理论和方法分析和解决具体问题的能力。

二、上机实验前的准备工作

在上机实验前应事先做好准备工作，以提高上机实验的效率，准备工作至少应包括：

（1）了解所用的实验环境；

（2）复习和掌握与本实验有关的教学内容；

（3）准备好上机所需的程序。由于计算机实验室给每个学生安排的时间是有限的，要珍惜时间，充分利用。应当在上机前按指定的题目编写好程序。手编程序应书写整齐，并经人工检查无误后才能上机，以提高上机效率。初学者切忌不编程序或抄别人

程序去上机，应从一开始就养成严谨的科学作风；

（4）对运行中可能出现的问题事先作出估计，对程序中自己有疑问的地方，应作出记号，以便在上机时给予注意；

（5）准备好调试和运行时所需的数据。

三、实验报告

实验后，应整理出实验报告，实验报告应包括以下内容：

（1）题目；

（2）程序清单（计算机打印出的程序清单）；

（3）运行结果（必须是上面程序清单所对应打印输出的结果）；

（4）对运行情况所作的分析以及本次调试程序所取得的经验。如果程序未能通过，应分析其原因。

第二部分 实验环境介绍

实验环境可根据实际情况选择Windows或者Linux环境，开发语言可根据学生习惯和能力选择C/C++或者Java 。

一、Windows实验环境

1．Windows 环境

2．C/C++环境

在Windows下，采用以Turbo C 2.0或Visual C++ 6.0为集成开发工具的C/C++实验环境。

3．Java环境

在Windows下，采用以JDK1.8和Eclipse为开发工具的Java 实验环境。

实验环境的搭建及配置详见附录A。

二、Linux实验环境

1．Linux环境

建议采用在VMware虚拟机下安装Ubuntu系统作为Linux实验环境。

安装及配置详见附录B。

2．C环境

在Linux下，采用以Vi +GCC+GDB为开发环境的C语言实验环境。其中Vi 作为编辑器，GCC作为编译器，GDB作为调试器。

3．Java 环境

在Linux下，采用以JDK1.8和Eclipse为开发工具的Java 实验环境。

实验环境的搭建及配置详见附录A。

第三部分 实验内容与安排

实验一 进程管理

1．实验目的

（1）理解进程的概念，明确进程和程序的区别。

（2）理解并发执行的实质。

（3）掌握进程的睡眠、同步、撤销等进程控制方法。

2．实验内容和步骤

2.1. 进程的创建。

① 编写一段源程序，使系统调用fork()创建两个子进程，当此程序运行时，在系统中 有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符：父进程显示字符“a”；子进程分别显示字符“b”和字符“c”。试观察记录屏幕上的显示结果，并分析原 因。

② 修改已编写的程序，将每个进程输出一个字符改为每个进程输出一句话，在观察程序执行时屏幕出现的现象，并分析原因。

2.2. 进程的控制 ① 用fork()创建一个进程，再调用exec()用新的程序替换该子进程的内容。 ② 利用wait()来控制进程执行顺序。

3．实验准备

3.1进程

Linux 中，进程既是一个独立拥有资源的基本单位，又是一个独立调度的基本单位。一个进程实体由若干个区（段）组成，包括程序区、数据区、栈区、共享存储区等。每个区又分为若干页，每个进程配置有唯一的进程控制块 PCB，用于控制和管理进程。

PCB 的数据结构如下：

1. 进程表项（Process Table Entry）

包括一些最常用的核心数据如：进程控制符 PID、用户标识符 UID、进程状态、事件描 述符、进程和 U 区在内存或外存的地址、软中断信号、计时域、进程的大小、偏置值 Nice、 指向就绪队列中下一个 PCB 的指针 P_Link、指向 U 区进程正文、数据及栈在内存区域的指针等。

2. U 区（U Area）

用于存放进程表项的一些扩充信息。每一个进程都会有一个私用的 U 区，其中包

含：进程表项指针、真正用户标识符 u-ruid(read user ID)、有效用户标识符 u-euid(effective user ID)、用户文件描述符表、计时器、内部 I/O 参数、限制字段、差错字段、返回值、信号处理数组。

3. 系统区表项 系统区表项用来存放各个段在物理存储器中的位置等信息。系统把一个进程的虚地址空 间划分为若干个连续的逻辑区，有正文区、数据区、栈区等。这些区是可被共享和保护的独 立实体，多个进程可共享一个区。为了对区进行管理，核心中设置一个系统区表，各表项中 记录了以下有关描述活动区的信息： 区的类型和大小、区的状态、区在物理存储器中的位置、引用计数、指向文件索引节点 的指针。

4. 进程区表

系统为每个进程配置了一张进程区表。其中，每一项记录一个区的起始虚地址及指向系统区表中对应的区表项。内核通过查找进程区表和系统区表，便可将区的逻辑地址变换为物理地址。

3.2所涉及的系统调用

1. fork() 创建一个新进程。

系统调用格式：

Pid=fork()

参数定义：

Int fork() fork()返回值意义如下：

0：在子进程中，pid 变量保存的 fork()返回值为 0，表示当前进程是子进程。

>0：在父进程中，pid 变量保存的 fork()返回值为子进程的 id 值（进程唯一标识符）。

-1：创建失败。

如果 fork()调用成功，它向父进程返回子进程的 pid，并向子进程返回 0，即 fork() 被调用了一次， 但返回了两次。 此时 OS 在内存中建立一个新进程， 所建的新进程是调用 fork() 父进程（parent process）的副本，称为子进程（child process） 。子进程继承了父进程的 许多特性，并具有父进程完全相同的用户级上下文，父进程与子进程并发执行。

内核为 fork()完成以下操作：

1）为新进程分配一进程表项和进程标识符

进入 fork()后，内核检查系统是否有足够的资源来建立一个新进程。若资源不足，则 fork()系统调用失败；否则，核心为新进程分配一进程表项和唯一的进程标识符。

2）检查同时运行的进程数目

超过预先规定的最大数目时，fork()系统调用失败。

3）拷贝进程表项中的数据

将父进程的当前目录和所有已打开的数据拷贝到子进程表项中，并置进程的状态为“创 建”状态。

4）子进程继承父进程的所有文件

对父进程当前目录和所有已打开的文件表项中的引用计数加 1.

5）为子进程创建进程上、下文

进程创建结束，设子进程状态为“内存中就绪”并返回子进程的标识符。

6）子进程执行

虽然父进程与子进程程序完全相同，但每个进程都有自己的程序计数器 PC，然后根据 Pid 变量保存的 fork()返回值的不同，执行了不同的分支语句。

2. exec()系列

系统调用 exec()系列，也可用于新程序的运行。fork()只是将父进程的用户级上下文拷贝到新进程中，而 exec()系列可以将一个可执行的二进制文件覆盖在新进程的用户级上 下文的存储空间上，以更改新进程的用户级上下文。exec()系列中的系统调用都完成相同的 功能，它们把一个新程序装入内存，来改变调用进程的执行代码，从而形成新进程。如果 exec()调用成功后，没有任何数据返回，这与 fork()不同。exec()系列调用在 LINUX 系统 库 unistd.h 中，共有 execl、execlp、execv、execvp 五个，其基本功能相同，只是以不同的方式来给出参数。

一种是直接给出参数的指针，如：

```
Int execl(path,arg0[,arg1,...argn],0);
Char *path,*argv[];
```

3. exec()和 fork()联合使用

系统调用 exec()和 fork()联合使用能为程序开发提供有力支持。 用 fork()建立子进程，然后再在子进程中使用 exec()，这样就实现了父进程与一个和它完全不同子进程的并发执行。

一般，wait、exec 联合使用的模型为：

```
Int status;
.....
If(fork()==0)
{ ......;
execl(...);
......;
```

}

Wait(&status);

4. wait()

等待子进程运行结束。如果子进程没有完成，父进程一直等待。wait()将调用进程挂起， 直至其子进程因暂停或终止而发来软中断信号为止。如果在 wait()前已有子进程暂停或终 止，则调用进程做适当处理后便返回。

系统调用格式：

Int wait(status)

Int *status;

其中，status 是用户空间的地址。它的低 8 位反应子进程状态，为 0 表示子进程正常结束，非 0 则表示出现了各种各样的问题；高 8 位则带回了 exit()的返回值。exit()返回值由系统给出。

核心对 wait()作以下处理：

① 首先查找调用进程是否有子进程，若无，则返回出错码；

② 若找到一处于“僵死状态”的子进程，则将子进程的执行时间加到父进程的执行时间上，并释放子进程的进程表项；

③ 若未找到处于“僵死状态”的子进程，则调用进程便在可被中断的优先级上睡眠，等待其子进程发来软中断信号时被唤醒。

5. exit()

终止进程的执行。

系统调用格式：

Void exit(status)

Int status;

其中，status 是返回给父进程的一个整数，以备查考。

为了及时回收进程所占用的资源并减少父进程的干预，LINUX/LINUX 利用 exit()来实现 进程的自我终止，通常父进程在创建子进程时，应在进程的末尾安排一条 exit()，使子进 程自我终止。exit(0)表示进程正常终止，exit(1)表示进程运行有错，异常终止。

如果调用进程在执行 exit()时，其父进程正在等待它的终止，则父进程可立即得到其返回的整数。核心须为 exit()完成以下操作：

1）关闭软中断

2）回收资源

3）写记账信息

4）置进程为“僵死状态”

3.3.程序示例

自己补充。

3.4实验结果

结果展示以及分析原因。

实验二 进程通讯

1．实验目的

（1）消息缓冲队列、共享存储区机制进行进程间的通信；

（2）理解通信机制。

2．实验内容

1.使用消息缓冲队列来实现 client 进程和 server 进程之间的通信

server 进程先建立一个关键字为 SVKEY（如 75）的消息队列，然后等待接收类型为 REQ （例如 1）的消息；在收到请求消息后，它便显示字符串“serving for client”和接收到 的 client 进程的进程标识数，表示正在为 client 进程服务；然后再向 client 进程发送应答消息，该消息的类型是 client 进程的进程标识数，而正文则是 server 进程自己的标识 ID。client 进程则向消息队列发送类型为 REQ 的消息（消息的正文为自己的进程标识 ID） 以取得 sever 进程的服务， 并等待 server 进程发来的应答；然后显示字符串“receive reply from”和接收到的 server 进程的标识 ID。

2. 使用共享存储区来实现两个进程之间的进程通信

进程 A 创建一个长度为 512 字节的共享内存，并显示写入该共享内存的数据；进程 B 将共享内存附加到自己的地址空间，并向共享内存中写入数据。

3.实验准备

3.1消息队列

消息队列是消息的链接表，包括 Posix 消息队列 systemV 消息队列。它克服了前两种通信方式中信息量有限的缺点，具有写权限的进程可以向消息队列中按照一定的规则添加新消息；对消息队列有读权限的进程则可以从消息队列中读取消息。

消息队列通过系统调用 msgget()、msgrcv()来实现，函数的功能和实现过程分别如下。 原型：int msgget（key_t key，int msgflg）

参数：

key：消息队列关联的键。

msgflg：消息队列的建立标志和存取权限。

返回值：成功执行时，返回消息队列标识值。失败返回-1，errno 被设为以下的某个值：

EACCES： 指定的消息队列已存在，但调用进程没有权限访问它，而且不拥有 CAP_IPC_OWNER 权能。

EEXIST：key 指定的消息队列已存在，而 msgflg 中同时指定 IPC_CREAT 和 IPC_EXCL 标志。

ENOENT：key 指定的消息队列不存在同时 msgflg 中不指定 IPC_CREAT 标志。

ENOMEM：需要建立消息队列，但内存不足。

ENOSPC：需要建立消息队列，但已达到系统的限制

原型：ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg)

参数：

msqid：消息队列的识别码。

msgp：指向消息缓冲区的指针，此位置用来暂时存储发送和接收的消息，是一个用户 可定义的通用结构。

msgsz：消息的大小。

msgtyp：从消息队列内读取的消息形态。如果值为零，则表示消息队列中的所有消息 都会被读取。

msgflg：用来指明核心程序在队列没有数据的情况下所应采取的行动。如果 msgflg 和常数 IPC_NOWAIT 合用，则在消息队列呈空时，不做等待马上返回-1，并设定错误码为 ENOMSG。当 msgflg 为 0 时， msgrcv()在队列呈满或呈空的情形时，采取阻塞等待的处理模式。

返回值：当成功执行时， msgrcv()返回拷贝到 mtext 数组的实际字节数。失败返回-1， errno 被设为以下的某个值：

E2BIG：消息文本长度大于 msgsz，并且 msgflg 中没有指定 MSG_NOERROR。

EACCES：调用进程没有读权能，同时没具有 CAP_IPC_OWNER 权能。

EAGAIN：消息队列为空，并且 msgflg 中没有指定 IPC_NOWAIT。

EFAULT：msgp 指向的空间不可访问。

EIDRM：当进程睡眠等待接收消息时，消息已被删除。

EINTR：当进程睡眠等待接收消息时，被信号中断。

EINVAL：参数无效。

ENOMSG：msgflg 中指定了 IPC_NOWAIT，同时所请求类型的消息不存在。

3.2共享内存

共享内存是操作系统常采用的进程间通信方式。它使得多个进程可以访问同一块内存空间，不同进程可以及时看到对方进程中对共享内存中数据的更新。这种通信方式需要依靠某种同步机制，如互斥锁和信号量等。

共享内存通过系统调用 shmget()、shmat()、shmdt()、shmctl()来实现，函数的功能和实现过程分别如下。

系统调用：shmget()

原型：int shmget(key_t key, int size, int shmflg)

参数：

Key:共享存储区的名字；

Size：共享存储区的大小（以字节计）；

Shmflag：用户设置的标志，如 IPC_CREAT，IPC_CREAT 表示若系统中尚无指名的共 享存储区，则由核心建立一个共享存储区；如系统中已有共享存储区，便忽略 IPC_CREAT。

返回值：

成功返回共享内存的标识符；不成功返回-1，errno 储存错误原因。

EINVAL 参数 size 小于 SHMMIN 或大于 SHMMAX。

EXIST 预建立 key 所致的共享内存，但已经存在。

EIDRM 参数 key 所致的共享内存已经删除。

ENOSPC 超过了系统允许建立的共享内存的最大值(SHMALL)。

ENOENT 参数 key 所指的共享内存不存在，参数 shmflg 也未设 IPC_CREAT 位。

EACCES 没有权限。

ENOMEM 核心内存不足。

系统调用：shmat()

原型：void* shmat(int shmid, const void *shmaddr, int shmflg);

参数：

Shmid：共享存储区的标识符；

Shmaddr：用户给定的，将共享存储区附接到进程的虚拟地址空间；

Shmflg：规定共享存储区的读、写权限，以及系统是否应对用户规定的地址做舍 入操作。其值为 SHM_RDONLY 时，表示只能读，其值为 0 时，表示可读、可写，其值为 SHM_RND （取整）时，表示操作系统在必要时舍去这个地址。

返回值：

该系统调用的返回值为共享存储区所附接到的进程虚地址。

系统调用：shmdt()

原型：int shmdt(void *shmaddr)

参数：

Shmaddr：要断开连接的虚地址，亦即以前由连接的系统调用 shmat（）所返回的虚地址。

返回值：

调用成功时，返回 0 值，调用不成功，返回-1 值。

系统调用：shmctl()

原型：int shmctl(int shmid, int cmd, struct shmid_ds *buf);

参数：

Shmid：由 shmget 所返回的标识符； Cmd：是操作命令，可以分多种类型：

（1）用于查询有关共享存储区的情况。如其长度、当前连接的进程数、共享 区的创建者标识符。

（2）用于设置或改变共享存储区的属性，如共享存储区的许可权、当前连接 的进程计数等。

（3）对共享存储区的加锁和解锁命令。

（4）删除共享存储区标识符等。

Buf：用户缓冲区地址。

返回值：

调用成功则返回 0，如果失败则返回-1。

3.3程序示例

1. 使其用消息缓冲队列来实现进程通信。

2. 使用共享存储区实现进程通信。

3.4实验结果

结果展示以及分析原因。

实验三 多线程实现生产者-消费者问题

一、预备知识

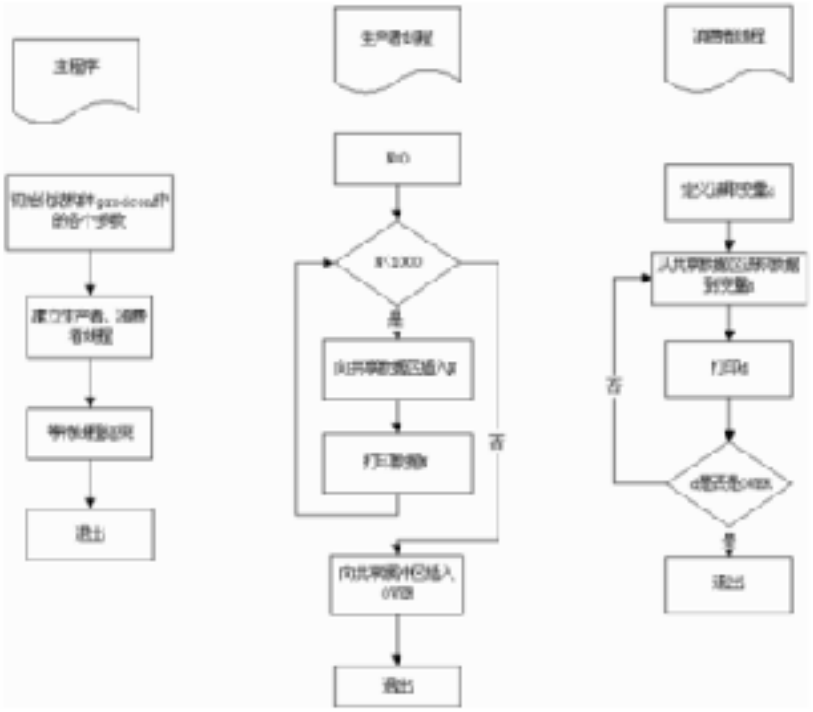
- 有 C 语言基础
- 掌握在 Linux 下常用编辑器的使用
- 掌握 Linux 下的程序编译过程
- 学习 pthread 库函数的使用
- 掌握共享锁和信号量的使用方法

二、实验设备及工具

利用PV操作解决生产者消费者问题

三、实验流程

生产者写入缓冲区和消费者从缓冲区读数的具体流程, 生产者首先要获得互斥锁, 并且判断写指针+1 后是否等于读指针，如果相等则进入等待状态，等候条件变量 notfull；如果不等则向缓冲区中写一个整数，并且设置条件变量为 notempty，最后 释放互斥锁。消费者线程与生产者线程类似。流程图如下：



生产者-消费者实验源代码结构流程

四、实验源代码:

自己补充。

五、线程 **API**说明：

在程序的代码中大量的使用了线程函数，如 pthread_cond_signal、pthread_mutex_init、pthread_mutex_lock 等。下面简单介绍，详细的说明请查阅资料。

- 线程创建函数：int pthread_create (pthread_t * thread_id, __const pthread_attr_t * __attr, void *(*__start_routine) (void *),void *__restrict __arg)
- 获得父进程 ID：pthread_t pthread_self (void)
- 测试两个线程号是否相同：int pthread_equal (pthread_t __thread1, pthread_t __thread2)
- 线程退出：void pthread_exit (void *__retval)
- 等待指定的线程结束：int pthread_join (pthread_t __th, void **__thread_return)
- 互斥量初始化：pthread_mutex_init (pthread_mutex_t *,__const pthread_mutexattr_t *)
- 销毁互斥量：int pthread_mutex_destroy (pthread_mutex_t *__mutex)
- 锁定互斥量（阻塞）：：int pthread_mutex_lock (pthread_mutex_t *__mutex)
- 解锁互斥量：int pthread_mutex_unlock (pthread_mutex_t *__mutex)
- 唤醒线程等待条件变量：int pthread_cond_signal (pthread_cond_t *__cond)
- 等待条件变量（阻塞）：：int pthread_cond_wait (pthread_cond_t *__restrict __cond, pthread_mutex_t *__restrict __mutex)

六、实验步骤

- 编辑源码，得到pthread.c
- 使用gcc编译（或者使用make工具）
- 运行

七、运行结果：

结果展示以及原因分析。

实验四 进程调度

1 实验目的

1. 理解 Linux 管理进程所用到的数据结构。

2. 理解 Linux 的进程调度算法的处理逻辑及其实现所用到的数据结构。

2 实验内容

1. 通过查阅参考书或者上网找资料，熟悉/usr/src/linux（注意：这里最后一级目录名可能是个含有具体内核版本号和“linux”字符串的名字）下各子目录的内容，即所含Linux源代码的情况。

2. 分析 Linux 进程调度有关函数的源代码，主要是 schedule()函数，并且要对它们引用的头文件等一并分析。

3. 实现 Linux 的进程调度算法及理解其实现所用的主要数据结构。

3 实验准备

3.1.Linux进程状态的描述

Linux将进程状态描述为如下五种：

TASK_RUNNING：可运行状态。处于该状态的进程可以被调度执行而成为当前进程。

TASK_INTERRUPTIBLE：可中断的睡眠状态。处于该状态的进程在所需资源有效时被唤醒，也可以通过信号或定时中断唤醒。

TASK_UNINTERRUPTIBLE：不可中断的睡眠状态。处于该状态的进程仅当所需资源有效时被唤醒。

TASK_ZOMBIE：僵死状态。表示进程结束且已释放资源，但其 task_struct 仍未释放。

TASK_STOPPED：暂停状态。处于该状态的进程通过其他进程的信号才能被唤醒。

3.2. 进程的虚拟地址空间

调度方式Linux中的每个进程都分配有一个相对独立的虚拟地址空间。该虚存空间分为两部分：用户空间包含了进程本身的代码和数据；内核空间包含了操作系统的代码和数据。Linux采用“有条件的可剥夺”调度方式。对于普通进程，当其时间片结束时，调度程序挑选出下一个处于TASK_RUNNING状态的进程作为当前进程（自愿调度）。对于实时进程，若其优先级足够高，则会从当前的运行进程中抢占CPU成为新的当前进程（强制调度）。发生强制调度时，若进程在用户空间中运行，就会直接被剥夺CPU；若进程在内核空间中运行，即使迫切需要其放弃CPU，也仍要等到从它系统空间返回的前夕才被剥夺CPU。

3.3. 调度策略

1）SCHED_OTHER

SCHED_OTHER 是面向普通进程的时间片轮转策略。采用该策略时，系统为处于TASK_RUNNING状态的每个进程分配一个时间片。当时间片用完时，进程调度程序再选择下一个优先级相对较高的进程，并授予CPU 使用权。

2）SCHED_FIFO

SCHED_FIFO 策略适用于对响应时间要求比较高，运行所需时间比较短的实时进程。采用该策略时，各实时进程按其进入可运行队列的顺序依次获得 CPU。除了因等待某个事件主动放弃CPU，或者出现优先级更高的进程而剥夺其CPU 之外，该进程将一直占用CPU运行。

3）SCHED_RR

SCHED_RR 策略适用于对响应时间要求比较高，运行所需时间比较长的实时进程。采用该策略时，各实时进程按时间片轮流使用 CPU。当一个运行进程的时间片用完后，进程调度程序停止其运行并将其置于可运行队列的末尾。

3.4. 相关函数

1）Schedule()函数

Schedule()函数首先对所有任务（进程）进行检测，唤醒任何一个得到信号的任务。具 体方法是针对任务数组中的每个任务，检查其报警定时值alarm。如果任务的alarm时间已经 过期（alarm<jiffies），则在它的信号位图中设置SIGALRM信号，然后清alarm值。jiffies是 系统从开机开始算起的滴答数（10ms/滴答），在sched..h 中定义。如果进程的信号位图中 除去被阻塞的信号外还有其他信号，并且任务处于可中断睡眠状态 （TASK_INTERRUPTIBLE），则置任务为就绪状态（TASK_RUNNING）。

2）sleep_on()函数

sleep_on()函数的主要功能是当一个进程（或任务）所请求的资源正忙或不在内存中时 暂时切换出去，放在等待队列中等待一段时间，当切换回来后再继续运行。放入等待队列的 方式是利用了函数中的tmp指针作为各个正在等待任务的联系。

3）wake_up()函数 唤醒操作函数wake_up()把正在等待可用资源的指定任务置为就绪状态。该函数是一个通用唤醒函数。在有些情况下，例如读取磁盘上的数据块，由于等待队列中的任何一个任务 都可能被先唤醒，因此还需要把被唤醒任务结构的指针置空。这样，在其后进入睡眠的进程 被唤醒而又重新执行sleep_on()时，就无需唤醒该进程了。

3.5程序示例

自己补充。

3.4实验结果

结果展示以及分析原因。

实验五 虚拟存储器管理

1 实验目的

1. 了解虚拟存储技术的特点；

2. 掌握请求页式管理的页面置换算法。

2 实验内容

1. 通过随机数产生一个指令序列，共 320 条指令。其地址按下述原则生成：

（1）50%的指令是顺序执行的；

（2）50%的指令是均匀分布在前地址部分；

（3）50%的指令是均匀分布在后地址部分；

具体的实施方法是：

A. 在[0，319]的指令地址之间随机选取一起点 M；

B. 顺序执行一条指令，即执行地址为 M+1 的指令；

C. 在前地址[0，M+1]中随机选取一条指令并执行，该指令的地址为 M’；

D. 顺序执行一条指令，其地址为 M’+1；

E. 在后地址[M’+2，319]中随机选取一条指令并执行；

F. 重复 A—E，直到执行 320 次指令。

2. 指令序列变换成页地址流

设：

（1）页面大小为 1K；

（2）用户内存容量为 4 页到 32 页；

（3）用户虚存容量为 32K。

在用户虚存中，按每 K 存放 10 条指令排列虚存地址，即 320 条指令在虚存中的存放方式为：

第 0 条—第 9 条指令为第 0 页（对应虚存地址为[0，9]）；

第 10 条—第 19 条指令为第 1 页（对应虚存地址为[10，19]）；

……

第 310 条—第 319 条指令为第 31 页（对应虚存地址为[310，319]）；

按以上方式，用户指令可组成 32 页。

3. 计算并输出下述各种算法在不同内存容量下的命中率。

A. 先进先出（FIFO）页面置换算法

B. 最近最久未使用（LRU）页面置换算法--最近最少使用算法

C. 最少使用（LFR）页面置换算法

D. 最佳（Optimal）页面置换算法

3. 实验准备

1. 先进先出（FIFO）页面置换算法

该算法总是淘汰最新进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰。该算法实现简单，只需把一个进程已调入内存的页面，按先后次序链接成一个队列，并设置一个指针，称为替换指针，使它总是指向最老的页面。

2. 最近最久未使用（LRU）页面置换算法

最近最久未使用（LRU）页面置换算法，是根据页面调入内存后的使用情况进行决策的。由于无法预测各页面将来的使用情况，只能利用“最近的过去”作为“最近的将来”的近似，因此，LRU 置换算法是选择最近最久未使用的页面予以淘汰。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间 t，当需淘汰一个页面时，选择现有页面中其 t 值最大的，即最近最久未使用的页面予以淘汰。

3. 最少使用（LFR）页面置换算法

在采用该算法时，应为在内存中的每个页面设置一个移位寄存器，用来记录页面被访问的频率。该置换算法选择在最近使其使用最少的页面作为淘汰页。

4. 最佳（Optimal）页面置换算法

该算法选择的被淘汰页面，将是以后永远不使用的，或许是在最长（未来）时间内不再被访问的页面。采用该算法，通常可保证获得最低的缺页率。但由于人们目前还无法预知一个进程在内存的若干个页面中，哪一个页面是未来最长时间内不再被访问的，因而该算法是无法实现的，但可以利用该算法去评价其他算法。

提示：A.命中率=1-页面失效次数/页地址流长度

B.本实验中，页地址流长度为 320，页面失效次数为每次访问相应指令时，该指令所对应的页不在内存的次数。

C.关于随机数产生方法，采用 VC 系统提供函数 RAND()和 RANDOMIZE()来产生

4. 程序示例

自己补充。

5. 实验结果

结果展示以及分析原因。

实验六 Linux 文件系统调用

1 实验目的

1.掌握 linux 提供的文件系统调用的使用方法;

2.熟悉文件系统的系统调用用户接口;

3.了解操作系统文件系统的工作原理和工作方式。

2 实验内容

编写一个文件工具 filetools，使其具有以下功能:

0.退出

1.创建新文件

2.写文件

3.读文件

4.修改文件权限

5.查看当前文件权限并退出

提示用户输入功能号，并根据用户输入的功能选择相应的功能

3 实验准备

用户在针对文件进行操作之前时一定要先打开它，这是由于系统需要根据用户提供的参数来查找文件的目录项，并由目录项找到文件的磁盘 i 结点，再将它调到内存 i 结点，才能建立用户进程与该文件之间的联系。

同样，在文件操作完毕后要关闭文件，以切断用户进程与文件的联系，释放相关资源。

Open 系统调用

int open(const char *path, int flags);

int open(const char *path, int flags,mode_t mode);

一般情况下使用两个参数的格式，只有当打开的文件不存在时才使用 3 个参数的格式。

参数:

Path 指向所要打开的文件的路径名指针。

Flag 标志参数，用来规定打开方式，必须包含以下 3 个之一:

O_RDONLY 只读方式

O_WRONLY 只写方式

O_RDWR

读写方式

利用按位逻辑或“|”对下列标志进行任意组合：

O_CREAT 如果文件不存在则创建该文件，若存在则忽略。

O_TRUNC 如果文件存在则将文件长度截为 0，属性和所有者不变。

C_EXECL 如果文件存在且 O_CREAT 被设置则强制 open 调用失败。

O_APPEND 每次写入时都从文件尾部开始。

Mode 是文件的访问权限，分为文件所有者、文件用户组和其他用户。

Close 系统调用

对于一个进程来说可以同时打开的文件是有限的，为了使文件标识符能够及时释放，系统必须提供关闭文件操作。

Int close(int fd)

Fd 为打开文件时系统返回的文件标识符

头文件：

#include <unistd.h>

系统执行该系统调用时，根据 fd 值在该进程的进程打开文件表中找到 fd 标识，根据指针找到系统打开文件表，再找到内存 i 结点表中相应的 i 结点，对其 i_count 进行减 1 操作，然后释放系统打开文件表中的表项和进程打开文件表的表项。

返回结果：调用成功返回 0

4 程序示例

自己补充。

5实验结果

结果展示以及分析原因。

附录A VMware虚拟机下安装Ubuntu系统

1安装VMware Workstation

（1）下载安装文件

在www.wmware.com下载VMware Workstation，安装过程很简单，默认配置一路下一步直至完成。安装过程需要填入注册码，在官网注册可得到30天免费试用的注册码。

（2）安装VMware

点击安装文件，全部默认选项，直至安装完成。

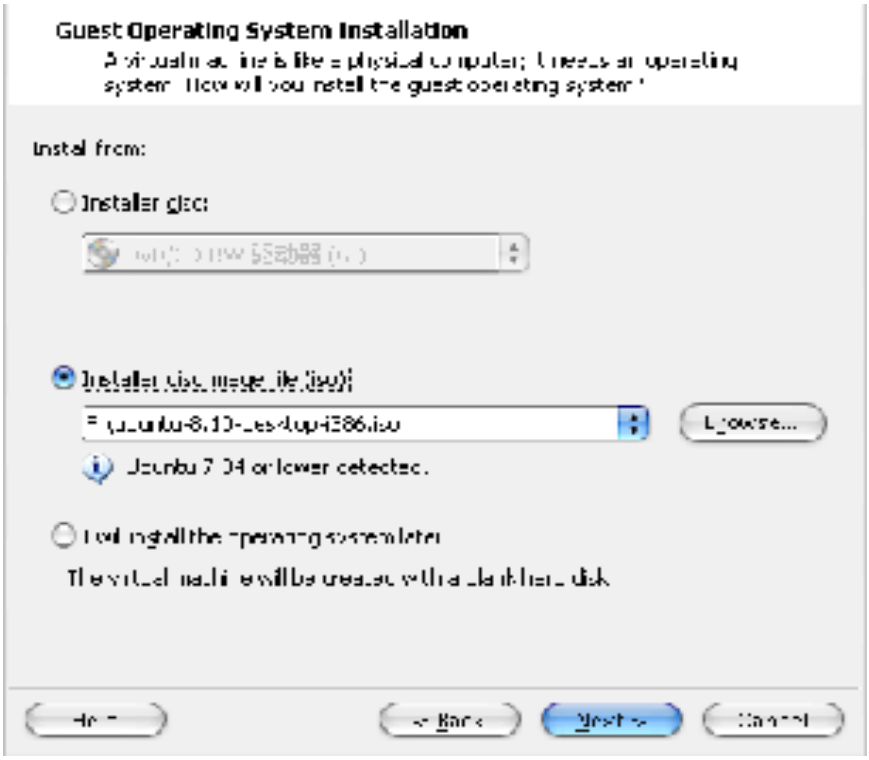
2 启动VMware，创建新的虚拟机

（1）双击图标启动VMware Workstation。

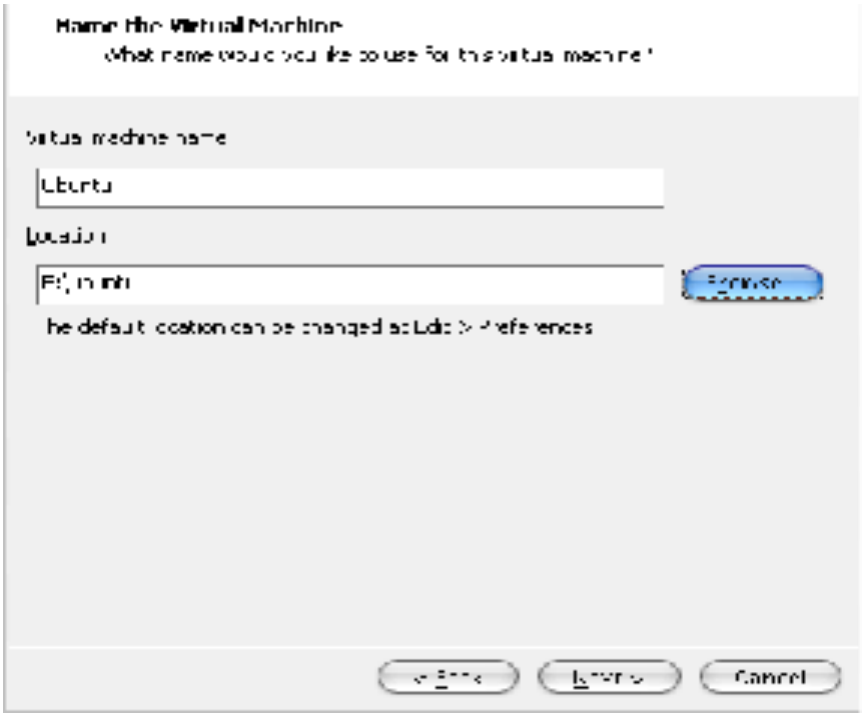
（2）在Home标签中选择New Virtual Machine新建虚拟机。

（3）在创建虚拟机向导中选择Typical。

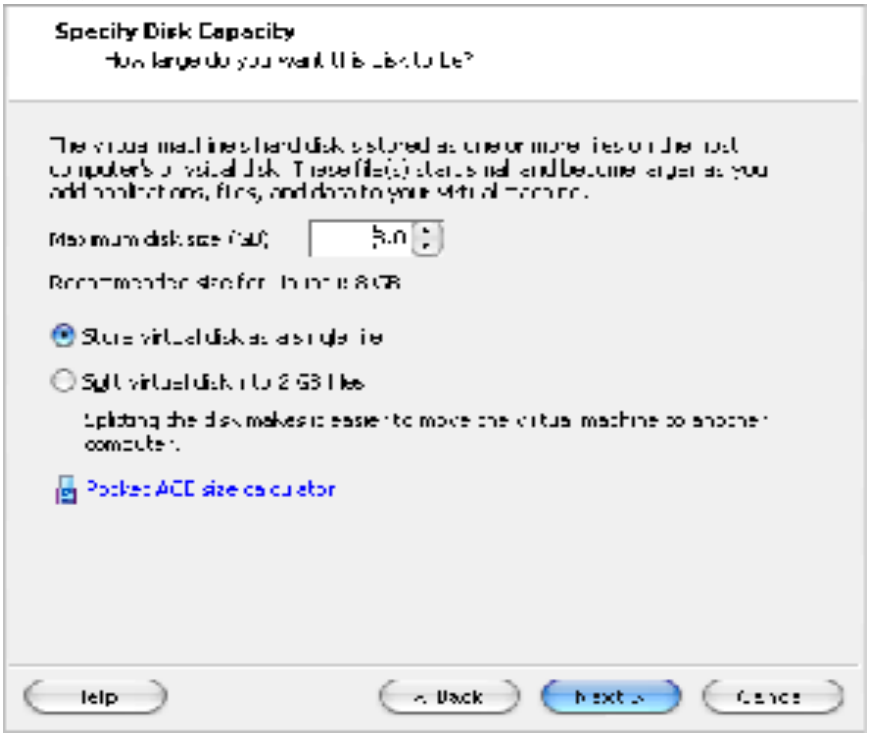
（4）选择从磁盘镜像安装，点击Browse选择.iso文件。



（5）填写虚拟机名字，和虚拟机文件的位置。

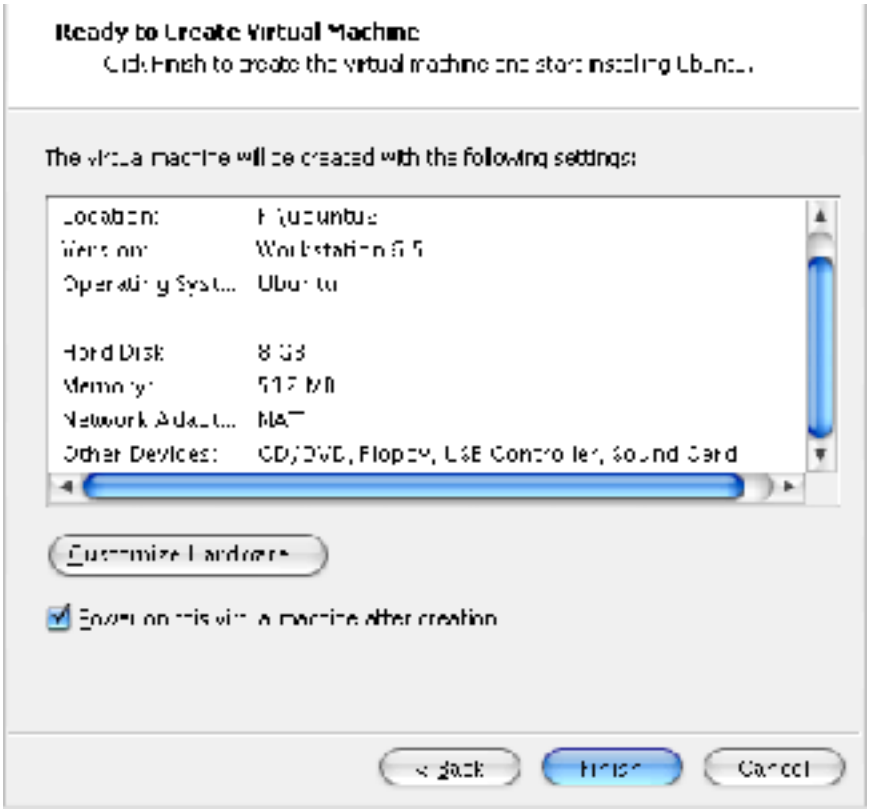


(6)虚拟机文件的大小，默认8G，应该够用了。



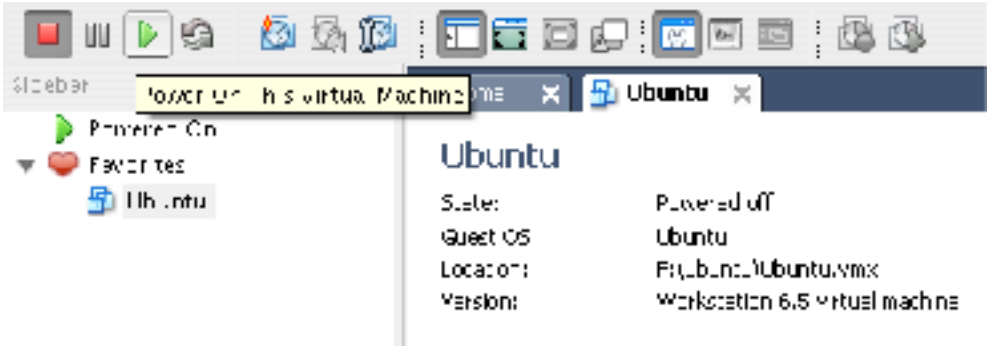
（7）完成虚拟机设置

检查创建信息，其中Network Adapter项默认选择的是NAT，表示使用网络地址翻译，这时只要宿主计算机已联网，则虚拟机就可以上网（虚拟机开启DHCP服务）。

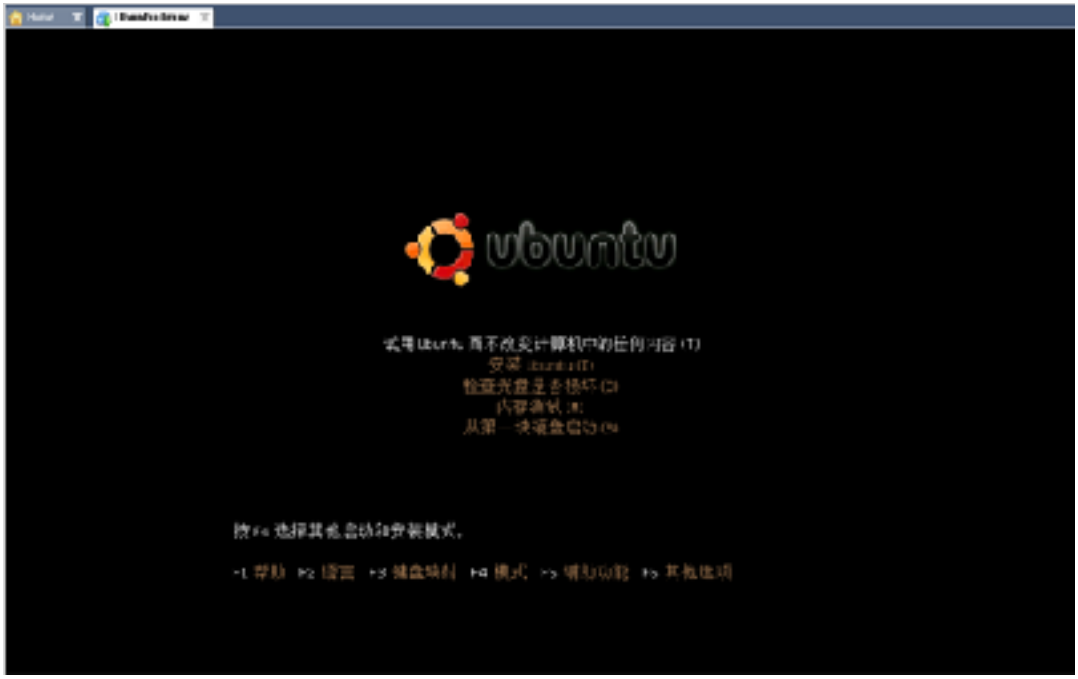


3 启动虚拟机安装ubuntu

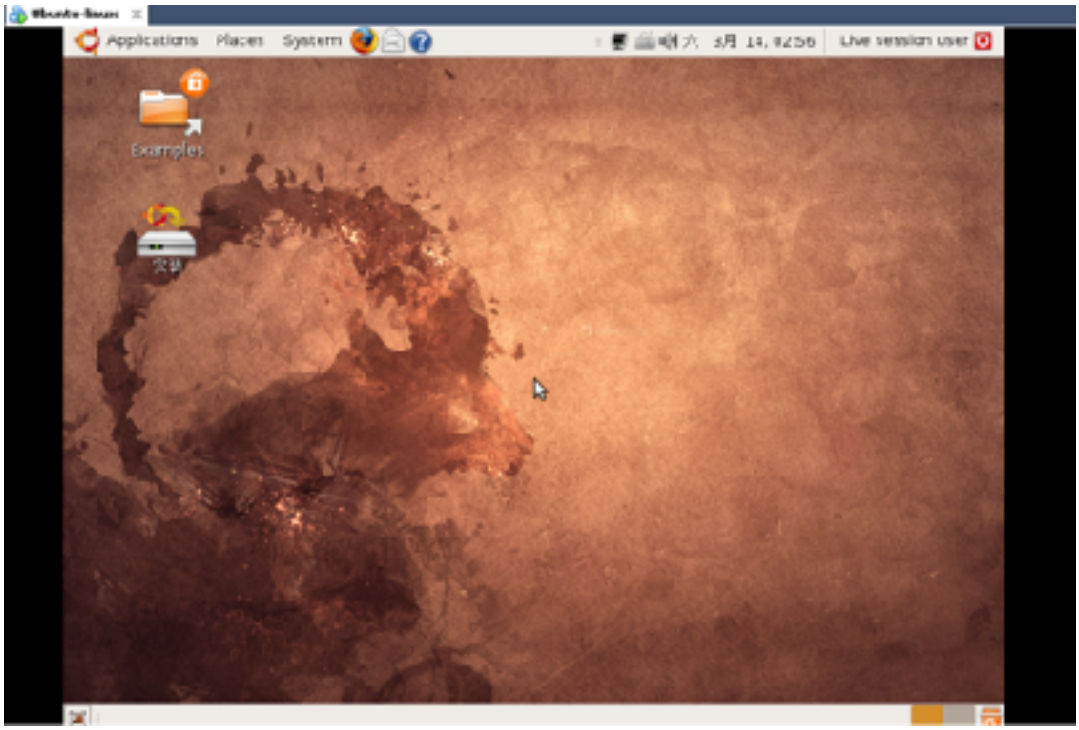
（1）点击绿色箭头启动虚拟机



启动虚拟机后就可以看到ubuntu启动界面，在虚拟机窗口鼠标左键点击，即可进入虚拟机界面进行操作，使用ctrl+alt组合键可释放鼠标返回原系统。

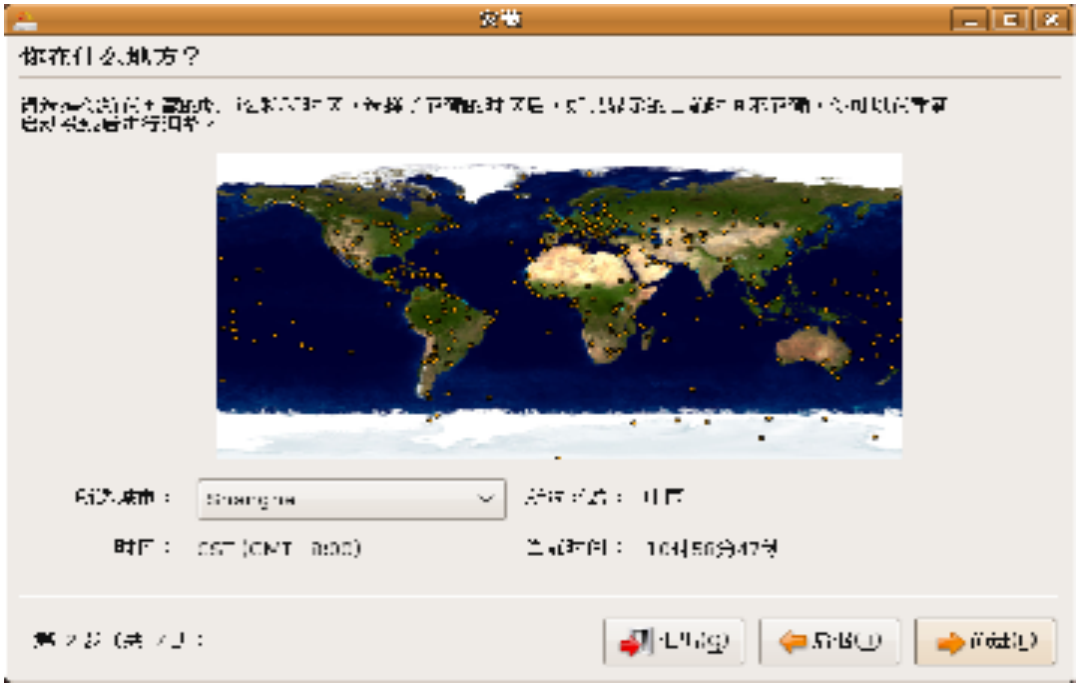


（1）在虚拟机中启动Ubuntu，点击安装图标，安装系统。

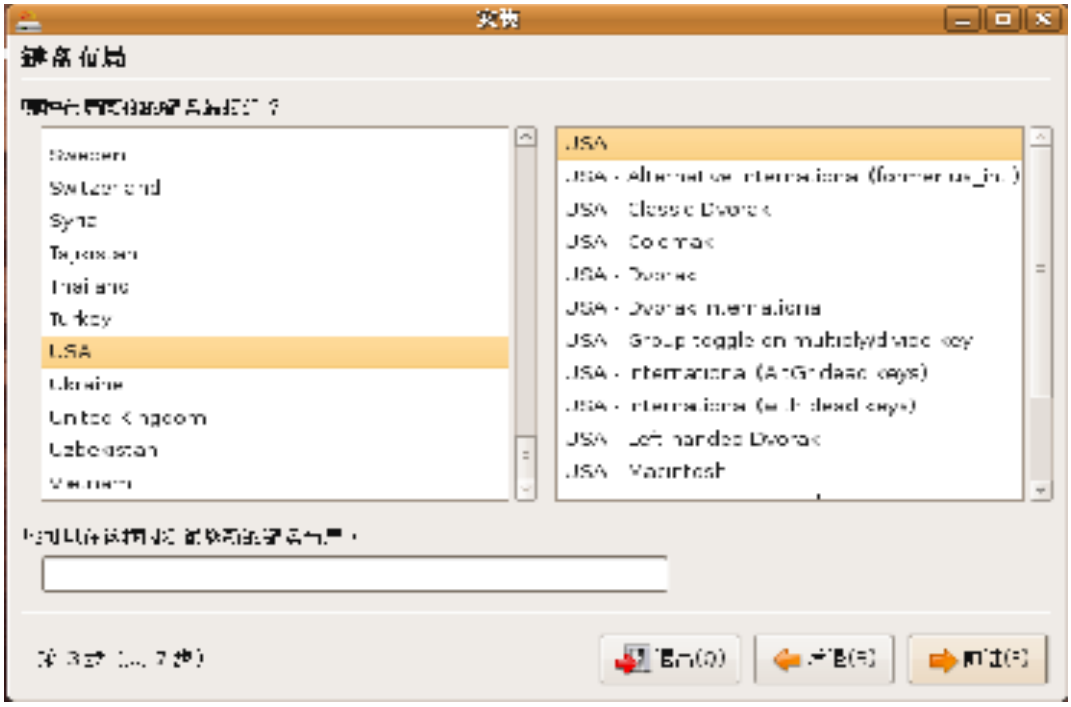


（2）选择语言。

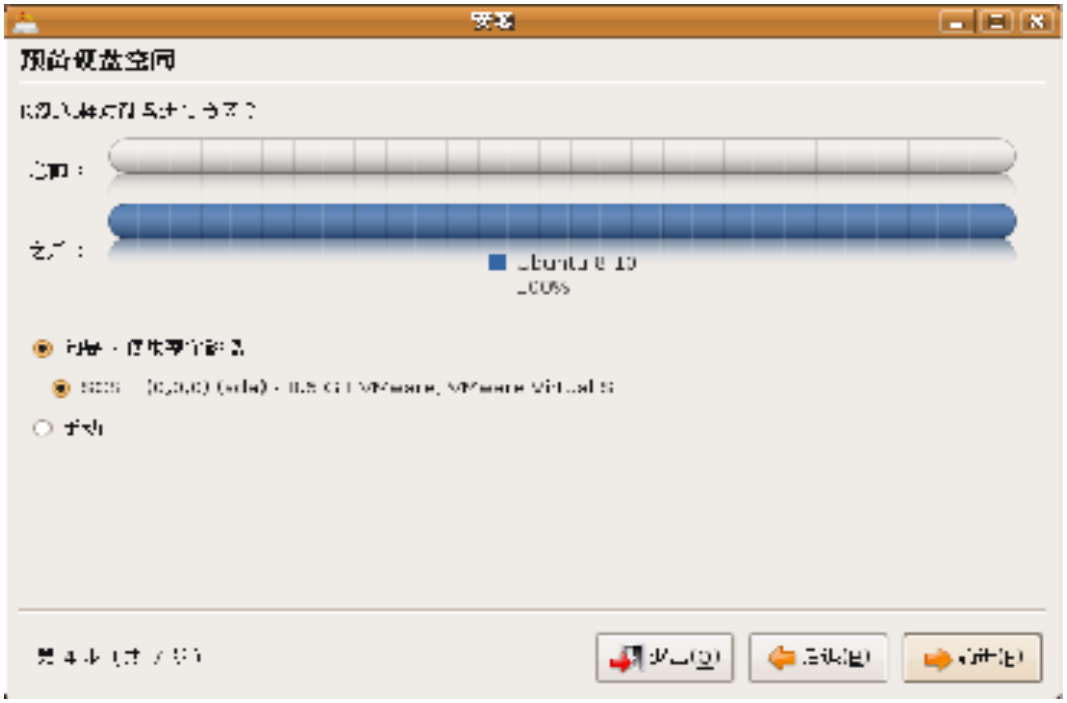
（3）选择时区。



（4）选择键盘布局。

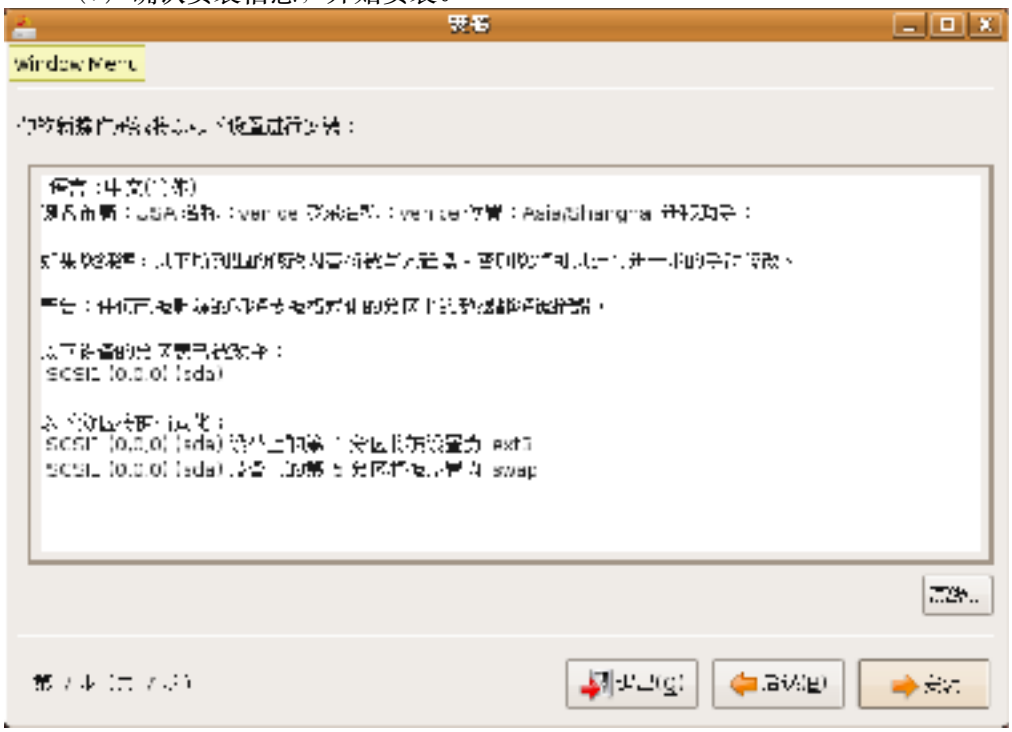


（5）分区，由于是用磁盘上的文件模拟了虚拟机的磁盘，所以直接选择向导- 使用整个磁盘，无需担心。



（6）填写登录名和密码。

（7）确认安装信息，开始安装。



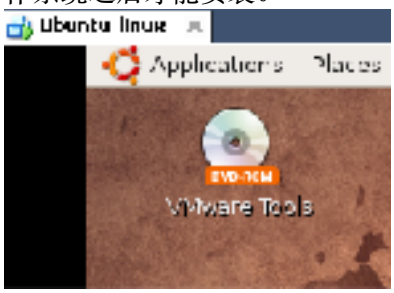
剩下的就是耐心等待，安装中可断开网络，安装程序默认将联网下载语言包，但使

用官方更新源速度较慢，建议安装好系统后再使用速度较快的源下载安装。

4 安装VMware Tools

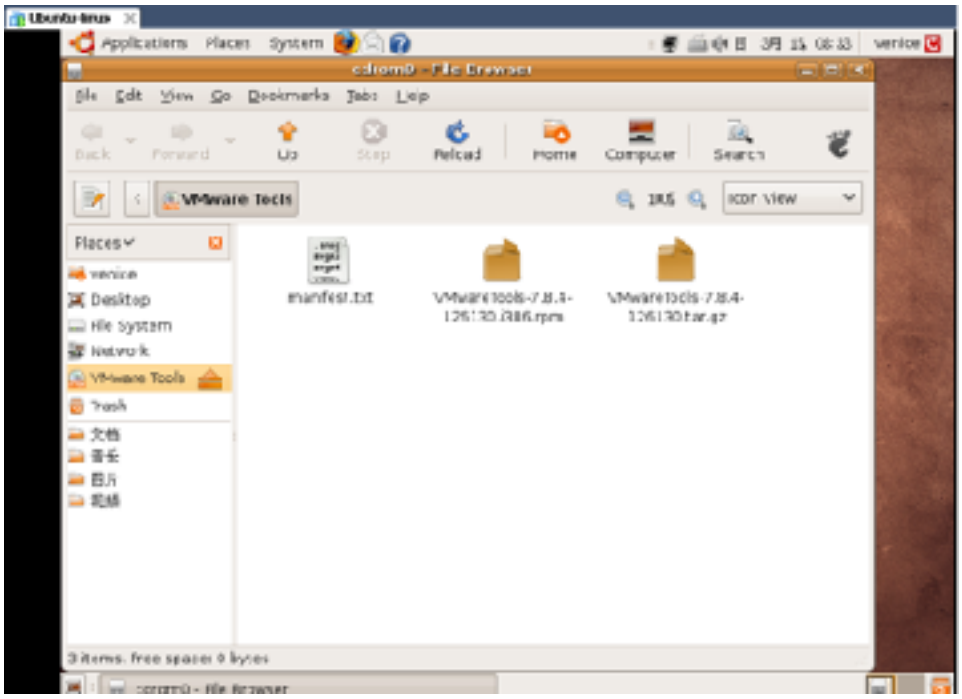
安装完成后再次启动虚拟机，将会在虚拟机窗口中看到Ubuntu的启动过程，完成后出现Ubuntu桌面。

桌面上出现了一个名为VMware Tools的光盘图标，并且被自动打开。这个工具将帮助我们完成虚拟机分辨率、文件共享、联网等配置，它必须在以及在虚拟机中安装了操作系统之后才能安装。



（1）在Ubuntu中安装VMware Tools。

双击图标，打开目录。



点击Ubuntu桌面左上角的Applications- 附件- 终端。

在终端界面中依次运行如下命令:

$ tar zxf /media/cdrom0/VMwareTools-7.8.4-126130.tar.gz（解压缩.tar.gz文件，文件路

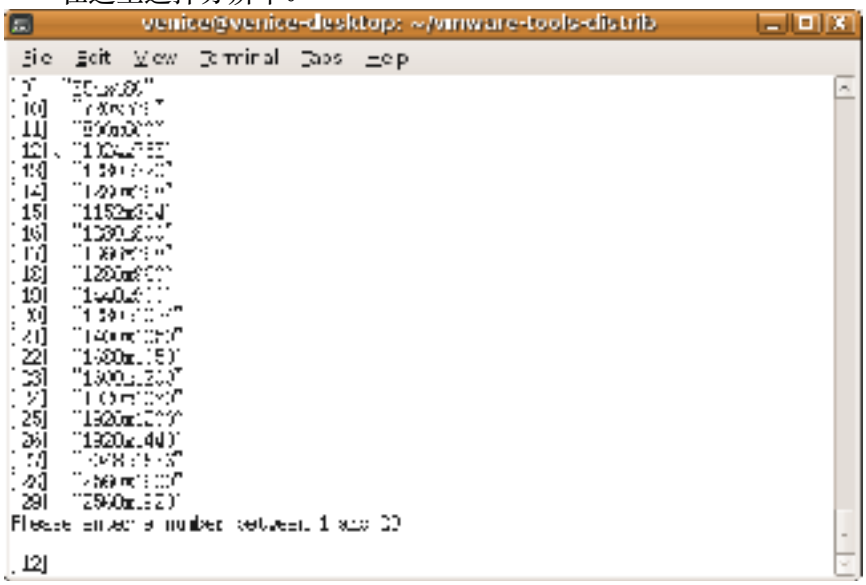
径和文件名可能不同在前面自动打开的目录中可以看到）

$ cd vmware-tools-distrib（切换到解压后的目录）

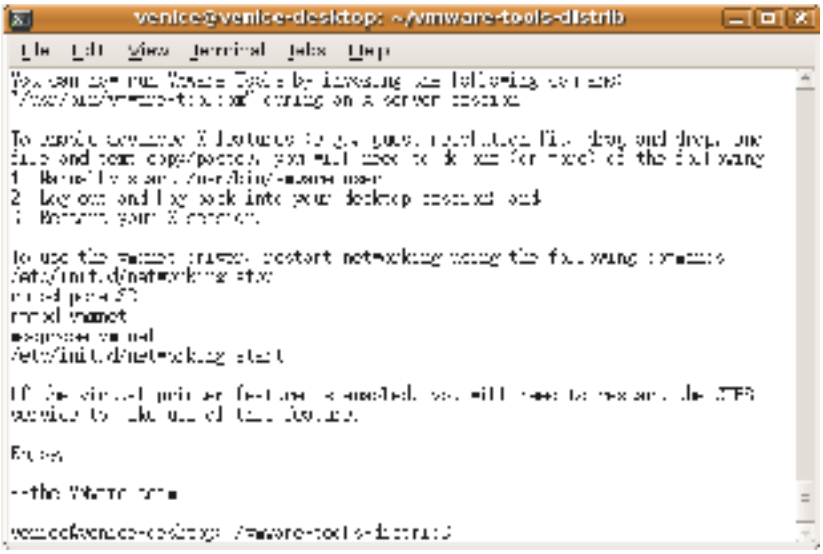
$ sudo ./vmware-install.pl（sudo表示要以根用户权限执行后面的安装命令，回车后提示输入用户密码）

经过一番确认回车后，直到最后出现“Enjoy——the VMware team”的字样后，VMwareTools的安装就完成了。

在这里选择分辨率。

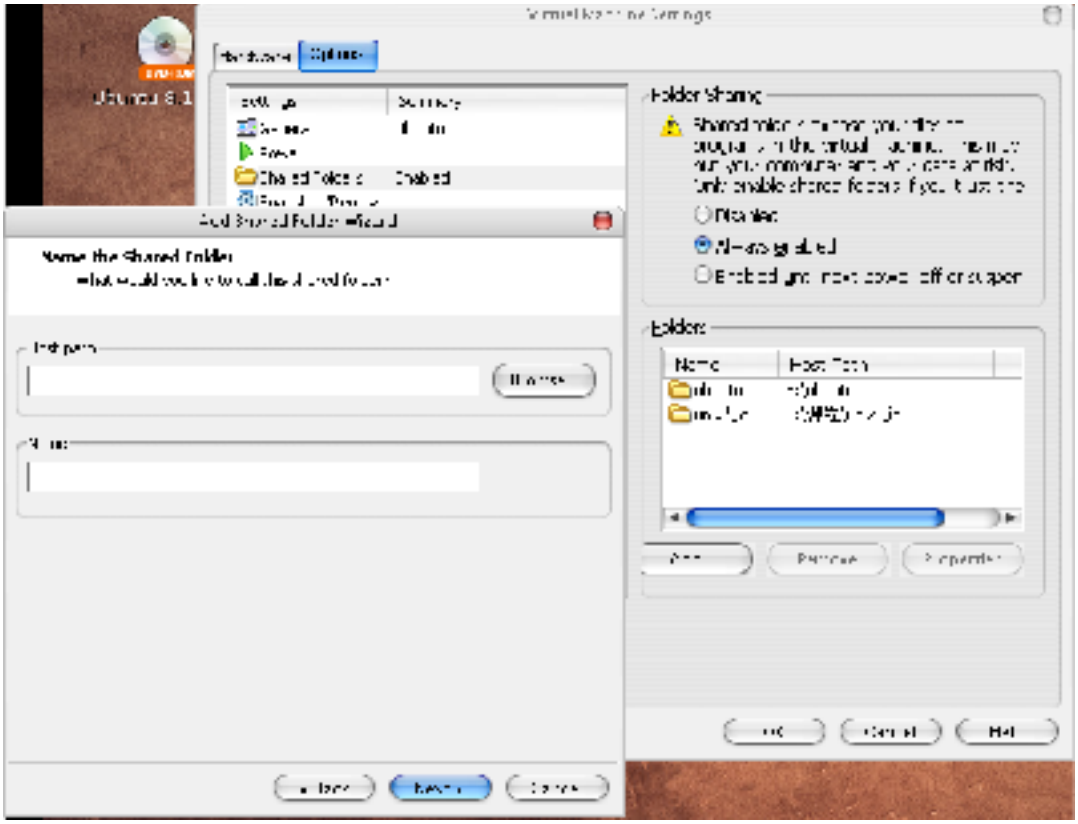


安装完成。

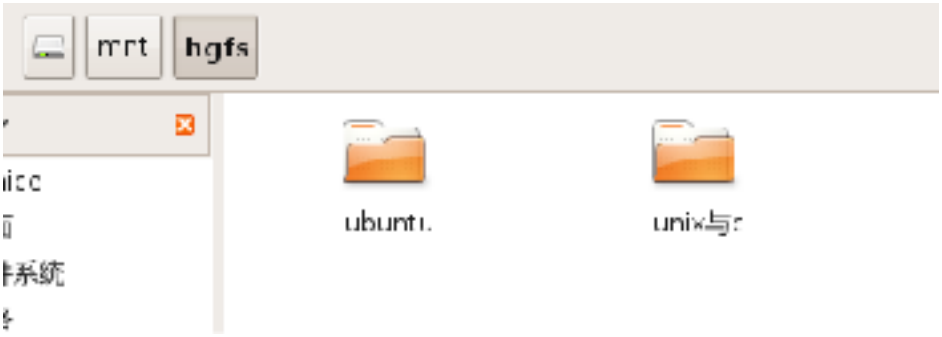


（2）共享文件的设置

在虚拟机VM菜单中选择settings，打开设置窗口的Options选项卡，选择Shared Forders。点击Add按钮后在添加共享目录向导的Host Path中选择你要共享的Windows中的目录。



完成设置后，在Ubuntu系统的/mnt下会出现hgfs目录，刚才设置的共享目录就在其中。



附录B Java开发环境安装与配置

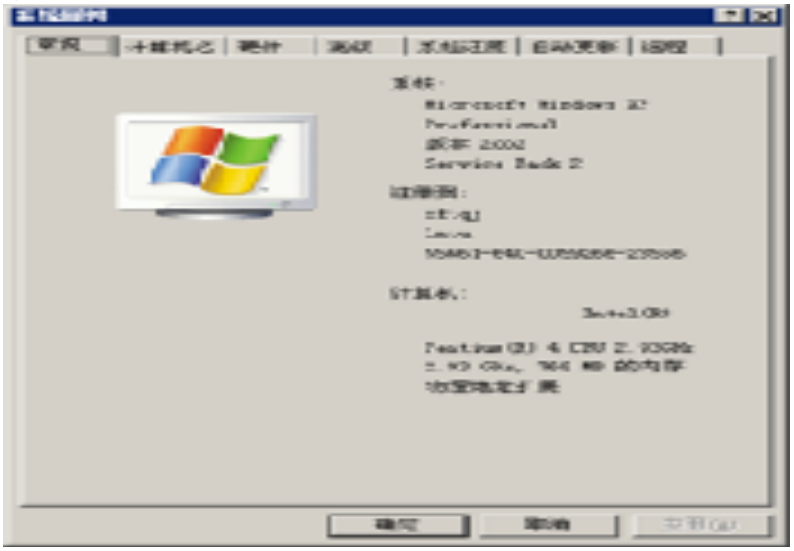
1 Windows下安装JDK

（1）下载JDK安装文件。

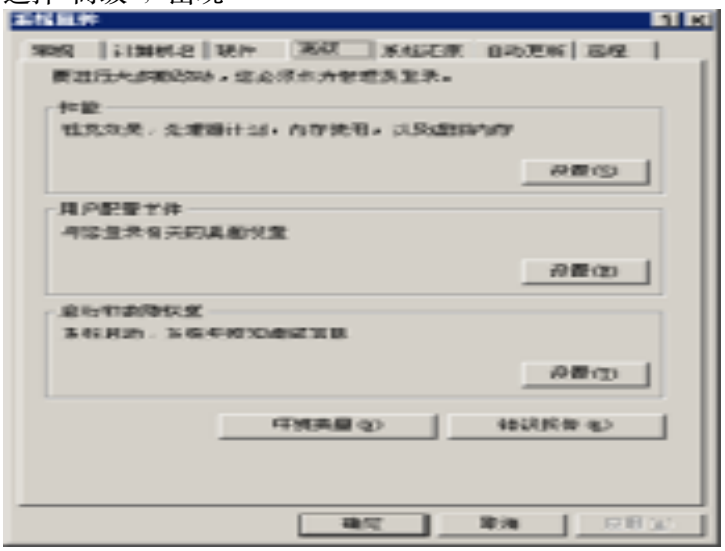
（2）点击安装文件，按照默认方式同意一切协议，一直“下一步”，在选择安装目录时，填写D:/Java作为安装目录，然后按照默认方式一直“下一步”。等待安装成功提示“完成”，点击完成，安装完成。整个过程大约5-10分。

（3）配置环境变量

桌面右击“我的电脑”，选择“属性”，出现如下对话框

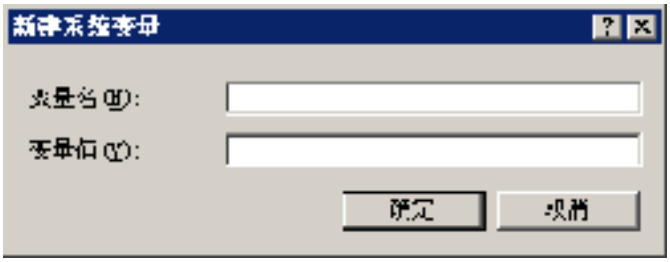


选择“高级”，出现



选择“环境变量”，出现

在“系统变量”窗口中选择“新建”，出现



在变量名中填写：JAVA_HOME；变量值：D:\Java\ jdk1.5.0_12（其中D:\Java是JDK的安装目录)，填写完成点击确定。

继续选择“新建”，在变量名中填写：CLASSPATH；变量值：

.;%JAVA_HOME%\bin; %JAVA_HOME%\lib; %JAVA_HOME%\lib\dt.jar; %JAVA_HOME%\lib\tools.jar(其中.表示当前目录，必须有)。完成后点击确定。

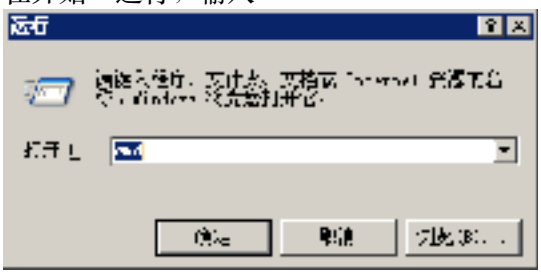
继续选择“新建”，在变量名中填写：Path；变量值：

.;%JAVA_HOME%\bin; %JAVA_HOME%;(其中“.”表示当前目录，必须有)。完成后点击确定。

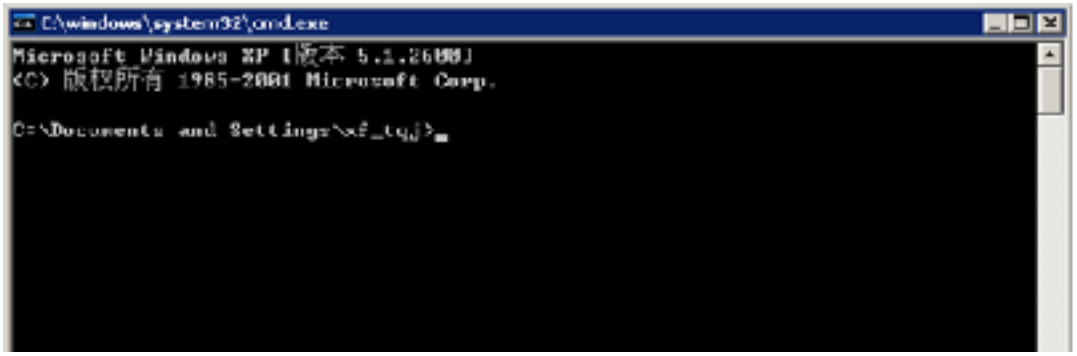
最后完成点击确定。

（4）测试

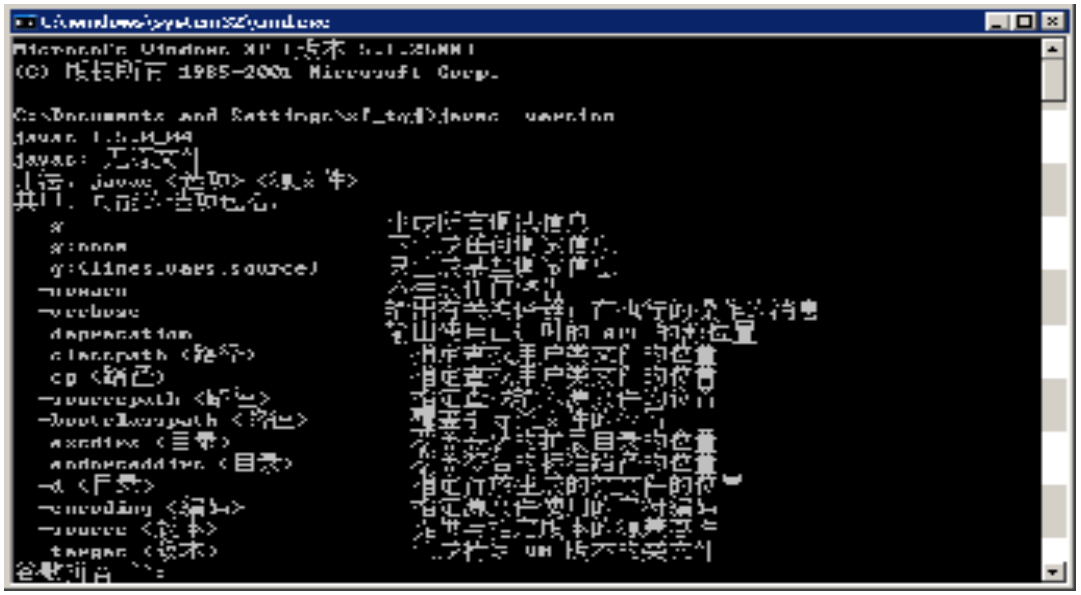
在开始- 运行，输入“cmd”



出现如下dos窗口



输入javac –version，出现java版本信息，例如



说明安装成功。

2 Linux下安装JDK

以Ubuntu系统为例，在命令行下输入：

$sudo apt-get install sun-java6-jre sun-java6-sdk

在联网情况下，系统将自动下载安装JDK。

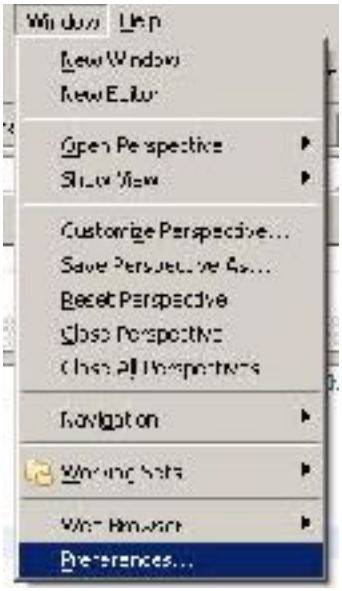
3 安装Eclipse

（1）下载与安装。

Eclipse是开源产品，在www.eclipse.org下载Windows或Linux平台相应的压缩格式的安装文件，解压后即可直接使用。

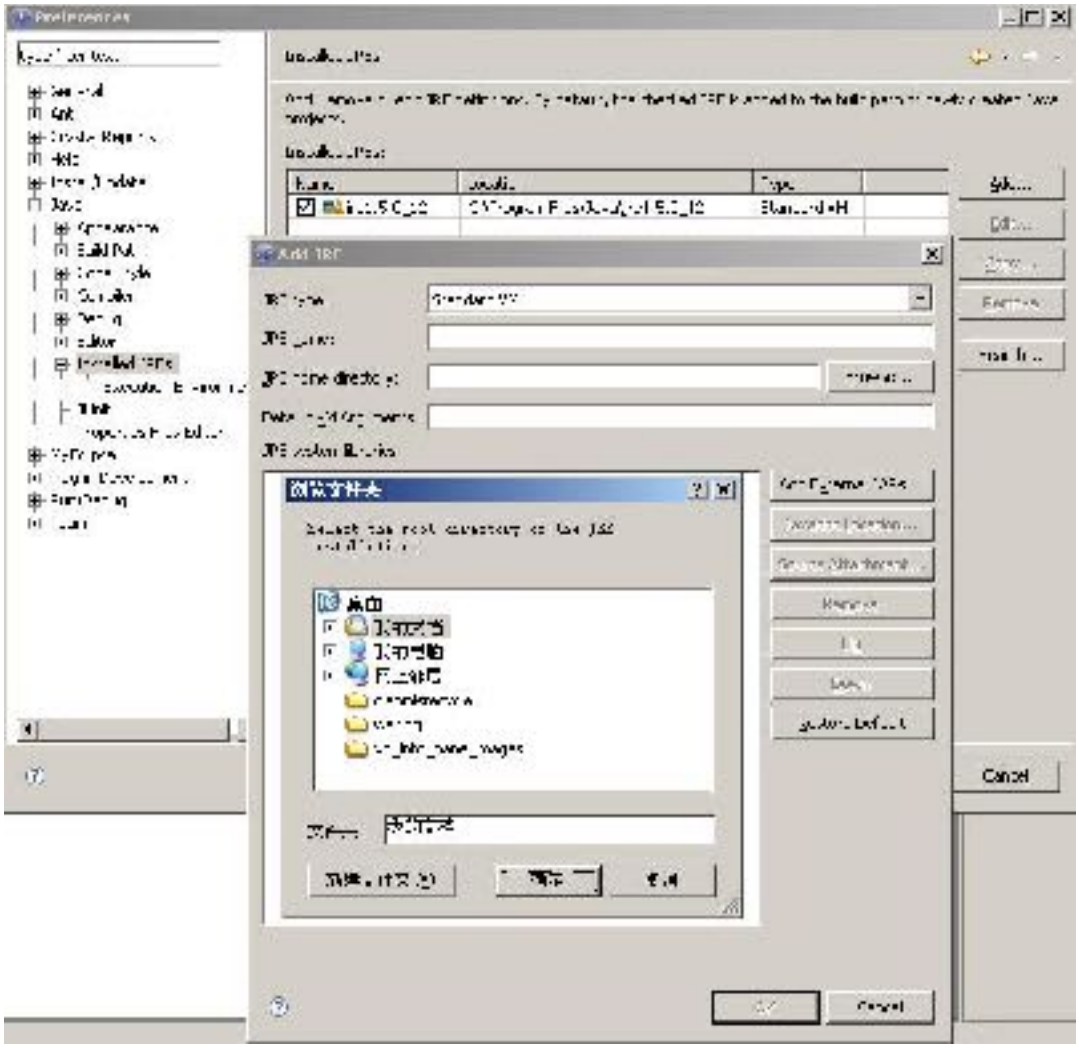
（2）在Eclipse中配置JRE

打开window- Perferences菜单



在Perferences对话框中，选择Java- Installed JREs，设置使用的JRE。

在右侧点击Add按钮，在Add JRE对话框中点击Browse按钮选择JRE所在目录，其余项目由Eclipse自动完成，点击OK，回到Perference对话框再次点击OK，完成配置。


Cancel