附录 A 操作系统实验环境

A.1 实验目的

1. 了解操作系统实验的意义
2. 掌握操作系统实验环境搭建
3. 了解实验用到的开发工具

在本章中，我们介绍了一般的 linux 环境下，操作系统开发的环境搭建的过程，这个过程对不同的操作系统开发，有一定的普适性。但目前整个实验环境在虚拟机上已经提供，如果无意在本地搭建环境且认为自己足够了解实验意义和相关工具的话，可以跳过这个章节。

A.2 了解操作系统实验

关于操作系统的教程可以说是数不胜数，但是对于一个从来没有写过或者参与过操作系统开发的人来说，这些书读起来总觉得有隔阂，没有一个感性的认识。这其中的根本原因，在于初学者一开始就面对一个完整的操作系统，或者是面对前人积累了几十年的一系列理论成果 (比如经典的页面替换算法)。而这些书的作者无论多擅长写作，或者写的代码多么优秀，要一个初学者理清其中的头绪都是相当困难的。

操作系统教程中往往注重一些理论知识的讲解，对具体实现的细节描述不够。初学者要是真的想自己动手写一个操作系统，会发现理论书籍一下子变得毫无用武之地。初学者甚至连如何将一个已经写好的操作系统原型加载到内存中都要花费很长时间，更不要说自己写出一个比较完善的操作系统。因此，要对操作系统有一个全面的理解，不仅要精读操作系统理论书籍，更要亲自动手编写代码，只有理论联合实际才能够完全掌握操作系统的精髓所在。

很多国际顶尖的高校为了操作系统的教学，编写了专门的操作系统。JOS 是由麻省理工学院教学专用的操作系统，麻省理工学院的操作系统课程之前一直用的是 JOS 操

作系统，该系统也是在很多学生的努力下完善起来的。这个操作系统实验是 MIT 公开课中的一个课程，搭建了一个基础的操作系统框架，让学生一步一步地实现操作系统中的内存管理、中断和异常处理、进程的创建和调度、SMP 支持、文件系统等功能，对于理解操作系统的实现非常有帮助。做完这个实验之后，可以对操作系统的实现有一个整体的、又不失细节的理解。这个操作系统还有一个特点就是它是一个微内核的系统，如果想要对比宏内核（比如 Linux）和微内核，也是一个很不错的选择。Harvard 大学的 David A.Holland 等也设计了 OS161 操作系统用于实现操作系统实验教学。因此，我们可以站在巨人的肩膀上，参考他们的设计思路、方法和源代码，尝试完成一个可以在 mips 上运行的小型操作系统。“麻雀虽小，五脏俱全”，在完成这个基于 mips 架构的小操作系统过程中，我们可以更好的理解操作系统启动、物理内存管理、虚拟内存管理、进程管理、中断处理、系统调用、文件系统、Shell 等主要操作系统内核功能，每个实验包含的内核代码量（C、汇编、注释）在 1000 行左右，充分体现了“小而全”的指导思想。

这个基于 mips 的小操作系统是运行在 mips 体系结构上的，而我们平常使用的都是基于 x86 体系结构的计算机，所以为了调试和开发的方便，我们采用 mips 硬件模拟器 GXEMUL。实验的开发环境是 GNU 的 gcc、gas 等工具。整个实验的运行和开发环境都在 Linux 中。

那么，我们如何来一步步地实现这个基于 mips 的小操作系统呢？根据一个操作系统和设计实现过程，可以有如下的实验步骤。

1. 启动操作系统的 bootloader，用于了解操作系统启动前的状态和准备工作，了解运行操作系统的硬件支持，操作系统如何加载到内存中，理解中断等。

2. 物理内存和虚拟内存管理子系统，理解分段和分页，了解操作系统如何管理物理内存和虚拟内存。

3. 进程管理和中断管理，了解进程的创建、执行、切换和结束，了解中断的完整过程。

4. 系统调用，了解系统调用的实现过程。

5. 文件系统，了解文件系统的具体实现，与进程管理等的关系。

6. Shell，了解 Shell 的实现过程。

其中每个开发步骤都是建立在上一个步骤之上，一步一步最终形成一个比较完善的小操作系统，如图1所示。

A.3 操作系统实验工具

A.3.1 交叉编译器

首先，整个实验是建立在 MIPS 上的，通过之前的计算机组成等课程的学习我们知道，不同类型的 CPU 有不同的 ISA。我们本地常使用的是 Intel 的 X86 指令集，或 AMD64(EMT64) 等指令集。而我们的小操作系统的目标机是 MIPS 指令集的。但我们需要使用交叉编译器来完成编译过程。我们的交叉编译器运行于 x86 平台上，但编译产

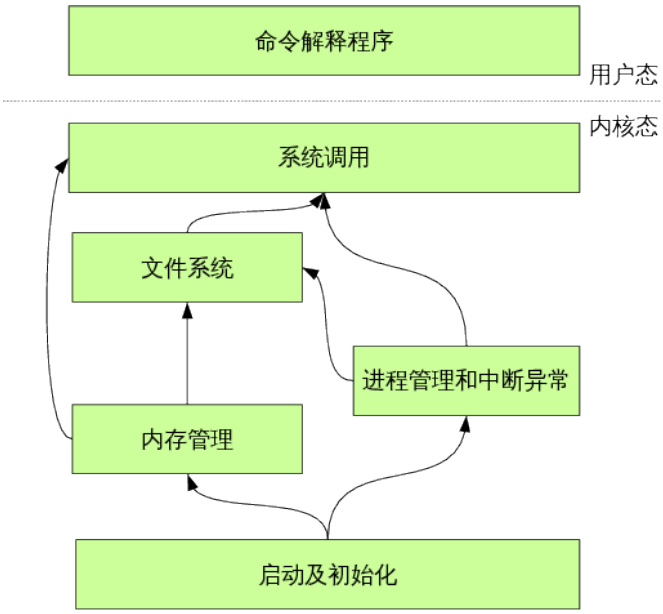

图 A.1: 六个实验间的关系

生的二进制文件却是在 mips 平台上运行的。如图?? 所示，编译器所运行的平台与其编译出来的程序的平台不同，因此叫做交叉编译器。

> **Note A.3.1** 严格意义上来说，所谓平台包含了两个概念：体系结构 (Architecture)、操作系统 (Operating System)。一个体系结构上，可以运行多种不同的操作系统。而同一个操作系统，也可以在不同体系结构上运行。举例来说，我们常说的 x86 Linux 平台实际上是 Intel x86 体系结构和 Linux for x86 操作系统的统称；而 x86 WinNT 平台实际上是 Intel x86 体系结构和 Windows NT for x86 操作系统的简称。

交叉编译器通常用于解决目标平台因性能不足等原因难以直接在其上开发的问题。举个例子，假如我们需要为一个 500MHz 的 ARM 开发程序，因为其性能、工具、环境等原因，我们很难直接在这个 ARM 的板子上进行开发。因此，我们可以选择在一个性能强劲、工具齐全的 x86 PC 机上完成程序，再通过交叉编译器编译为 ARM 指令集的程序。通过这样的方式，我们就能够轻松地为 ARM 开发程序了。

A.3.2 Linux 系统

Unix 是一个经典的操作系统。目前操作系统中的很多设计思想以及算法均是源自 Unix 的。但 Unix 在商业化后，我们很难以相对廉价的方式直接接触 Unix。目前相对较为完善的自由的 Unix 衍生版仅有 FreeBSD 等寥寥几个，其上的软件生态环境等等十分有限。不过，幸运的是，Linux 为我们提供了一个相对良好的接触 Unix 思想的机会。Linux 是一个自由的类 Unix(Unix-like) 系统，兼容 POSIX 标准，为我们的实验提供了一个良好的环境。近些年 Linux 发展迅猛，其上的软件环境等也十分丰富，这一点相较于 BSD 家族来说要好很多。我们实验中采用了 GNU 的工具，这一套工具主要是为 Unix 类系统编写的。同时，我们的小操作系统中有很多 Unix 风格的设计，因此，采用 Linux 平台做实验更有利于大家理解实验中的很多内容。实验为大家提供的虚拟机上的

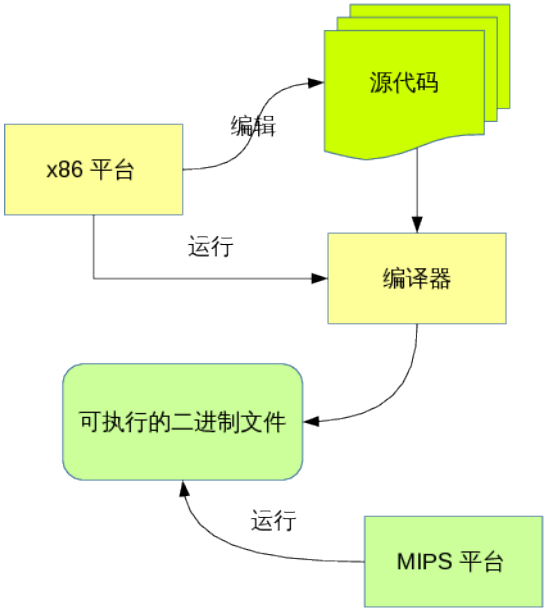

图 A.2: 交叉编译

环境是一个无图形界面的 Ubuntu 12.04，大家通过 ssh 远程连接到虚拟机上去做实验。通过命令行界面来进行全部的操作。很多同学以前没有太接触过这种操作方式，这一点需要大家慢慢熟悉。

> **Note A.3.2** 事实上，Linux 仅仅是一个操作系统内核。一般一个完整的基于 Linux 的操作系统还包含 GNU 的各类工具软件，图形环境，应用软件等等其他一系列的软件。但习惯上，将以 Linux 为内核的操作系统简称为 Linux。由于大家长期以来习惯在 Linux 内核上使用大量的 GNU 软件，所以，Richard Stallman 认为将这类操作系统称为“GNU/Linux”更为恰当。

A.4 实验环境配置

A.4.1 Linux 操作系统

由于所有的实验都是在 Linux 下完成的，所以需要 Linux 操作系统。我们选用设计上更为用户友好的 Ubuntu 系统。有两种可供参考的安装方式，一是通过虚拟机来安装 Linux 系统，二是直接安装 Linux 系统。这里考虑到大部分同学对于 Linux 不同熟悉，我们介绍虚拟机安装的方法。

首先，我们需要下载操作系统的镜像。为了下载速度考虑，我们选择从中国科学与技术大学的镜像站http://mirrors.ustc.edu.cn/上下载。点击该页面右侧的**获取安装镜像**，出现如图A.3所示的界面，版本选择与实验环境一致的 12.04。

接下来，我们下载虚拟机软件。虚拟机相当于一台虚拟的机器，虚拟机上的操作就好像是在操作另一台机器，对本机不会造成影响，是相对安全的一种体验 Linux 的方案。这里我们选择采用 VirtualBox 虚拟机。可以去官网https://www.virtualbox.org/wiki/Downloads下载，也可以去百度上自行寻找国内的较快的下载地址。当然，你也可以选

图 A.3: 获取安装镜像

择其他的虚拟机软件，如 VMware（性能更佳，但是是专有软件，你懂得），具体操作相仿。但 Virtualbox 是自由软件，所以出于版权方面的考虑，这里以 VirtualBox 为例。

安装 VirtualBox（相信安装这种小事一定难不住你）后，打开它，看到如图A.4所示的界面。

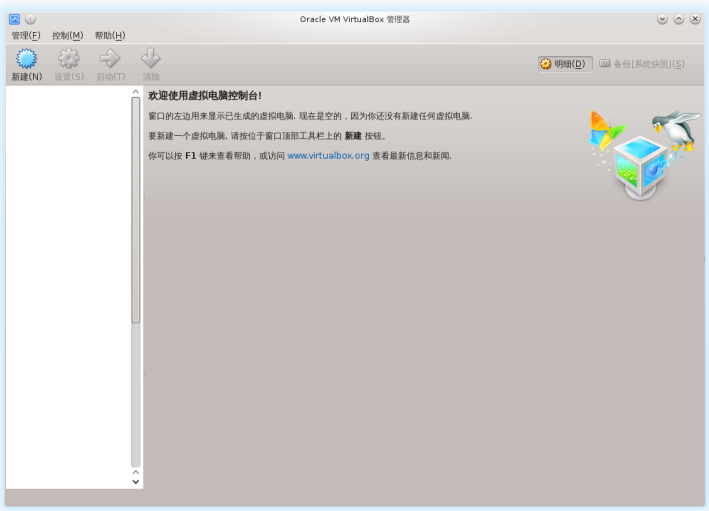

图 A.4: VirtualBox 虚拟机界面

点击**新建**，出现如图A.5所示的对话框。类型选择 **Linux**，版本选择 **Ubuntu(64bit)**（这里如果你下载的时候选择的是 i386 版本则选择 **Ubuntu(32bit)**）。点击**下一步**，内存选择 **1024MB**。下一步，选择**现在创建虚拟硬盘**，后面的设置如无特殊需求不必改动，一直下一步即可。直到选择磁盘大小处，建议选择 **20GB**，同时选定一个至少有 20GB 空间的的位置放置磁盘镜像文件。

图 A.5: 新建虚拟机

之后选中我们刚才建立的虚拟机，点**设置**，**存储**，点选**光盘加号形状图标**，**添加虚拟光驱**，如图A.6然后**选择磁盘**，选中之前下载好的 Ubuntu 的 ISO 文件。

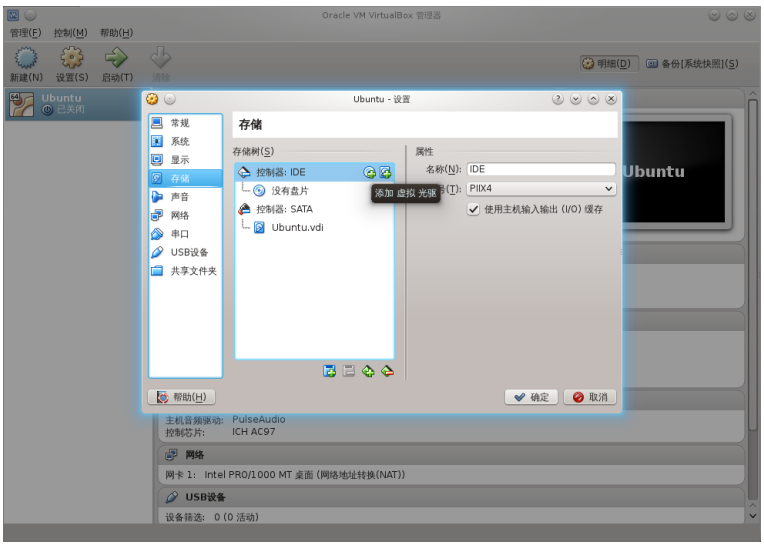

图 A.6: 选择启动镜像

选中虚拟机，点击**启动**，弹出虚拟机画面，启动后进入 Ubuntu 的安装界面（见图A.7)。左侧可以选择语言，这里我们选择 **English**，并点击 **Install Ubuntu**。(当然，按理说选择中文也是没问题的，但这里为了避免任何不必要的麻烦，我们选择了英文)。

之后由于我们没有改镜像源等设置，为了安装速度考虑，不要选择 **Download updates while installing**。以免 Ubuntu 从缓慢的官方服务器下载更新。**Install this third-party software** 选项与我们实验无关，你可以自行选择是否勾选。下一步选择 **Erase disk and install Ubuntu**。由于我们是在虚拟机上安装，且使用了新建的虚拟磁盘，所以自然可以大胆地让 Ubuntu 清空整个磁盘并安装。下一步一般不需要做改动，

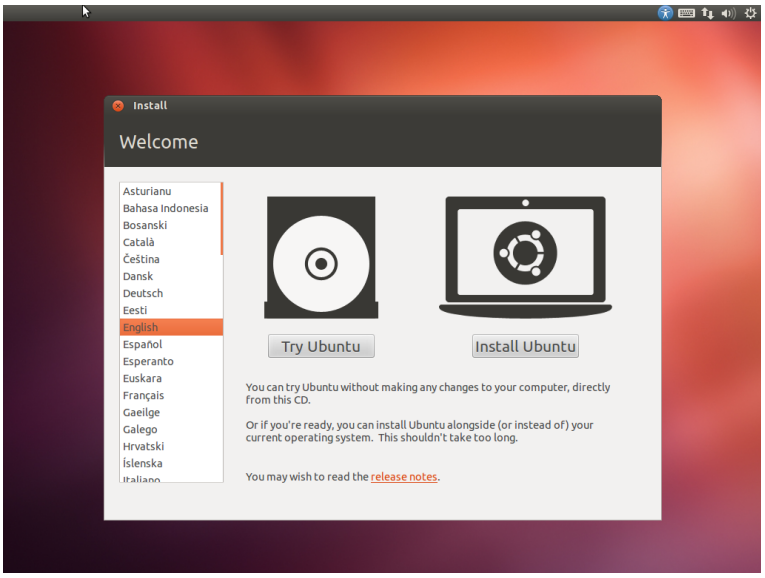

图 A.7: 选择启动镜像

点击 **Install Now** 就好。之后输入密码，等待安装结束就好。

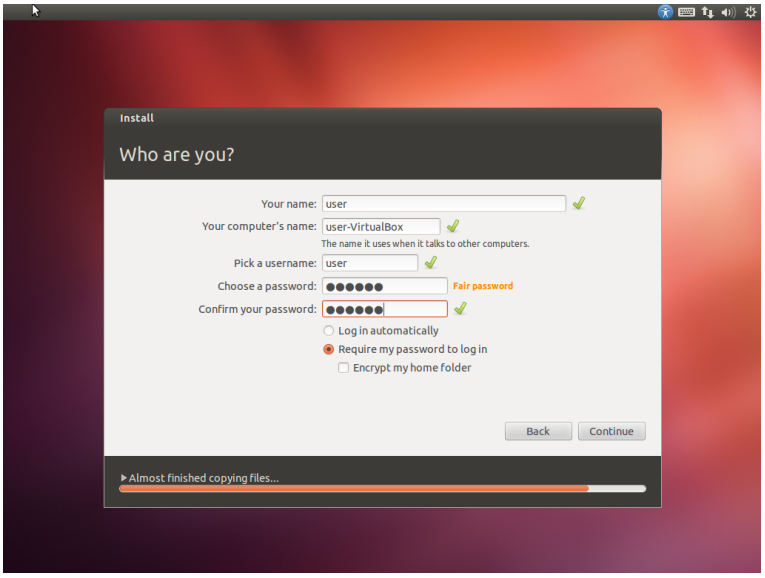

图 A.8: Ubuntu 安装中新建用户的界面

安装结束后点击 **Restart** 重启。重启时注意，安装镜像关闭时一般会提示你取出光盘并按回车键。光盘镜像正常情况下会被自动弹出。一般来说，你需要做的仅仅是按下回车键并等待重启。这里注意，有时候由于种种原因它不会输出这句提示，遇到这种情况就需要发动你的第六感了:) 感觉系统不再动了以后敲一个回车就好。

重启完成后登入 Ubuntu 界面，接下来我们安装增强功能。点击虚拟机的**设备**，然后选择**安装增强功能**。之后会 Ubuntu 会选择弹出对话框询问是否自动运行，如图A.9所示。点击 **Run** 后会弹出对话框要求输入密码，输入密码后会启动一个命令行，开始执行安装脚本，看到 **Press Return to close this window** 后按回车键结束安装。手动重启虚拟机，重启后虚拟机的分辨率会变成相对正常的状态，说明增强功能已经成功安

装。

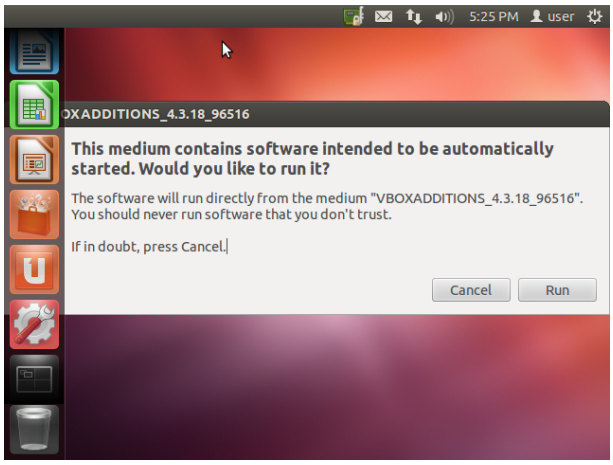

图 A.9: Ubuntu 安装中新建用户的界面

至此，一个 Ubuntu 环境就搭建完成了。你可以在上面学习并熟悉 Linux 的相关指令，并编写我们的实验代码。

Exercise A.1 安装一个 Linux 环境，并尝试学会使用 ls、cat、cd、cp、mv 这五条指令 ■

A.4.2 安装交叉编译器

交叉编译器的下载可以访问http://ftp.sunet.se/pub/Linux/distributions/eldk/4.1/mips-linux-x86/iso/ 到其中下载 **mips-2007-01-21.iso** 文件。这里下载网速较慢，建议大家下载完一份后相互拷贝，一般课程中心也会提供相应的文件。

在 Ubuntu 下，按住 Ctrl+Alt+t 快捷键，可以打开一个终端，我们在终端中

```
# 建立一个用于挂载 ISO 镜像的目录
sudo mkdir /mnt/mipsiso
# 挂载 iso 文件
sudo mount -o loop mips-2007-01-21.iso /mnt/mipsiso
# 切换到/mnt/mipsiso 文件夹中
cd /mnt/mipsiso
# 安装 32 位运行库（仅在 64 位系统上需要执行此步骤）
sudo apt-get install ia32-libs
# 运行安装脚本
sudo ./install -d /opt/eldk
```

执行完上面的指令后，检查/opt/eldk/usr/bin 下是否有以 mips_4KC 开头的一系列工具。打开命令行，尝试运行其中的 gcc，如果看到如下输出，则说明一切 OK～

```
# 注意，此时需要位于/opt/eldk/usr/bin 目录下
$ ./mips_4KC-gcc
Reading specs from /opt/eldk/usr/bin/../lib/gcc/mips-linux/4.0.0/specs
Target: mips-linux
Configured with: /opt/eldk/build/mips-2007-01-21/work/usr/src/denx/
BUILD/crosstool-0.35/build/gcc-4.0.0-glibc-2.3.5-eldk/mips-linux/
gcc-4.0.0/configure --target=mips-linux --host=i686-host_pc-linux-gnu
```

```
--prefix=/var/tmp/eldk.PyrfxY/usr/crosstool/gcc-4.0.0-glibc-2.3.5-eldk
/mips-linux --with-headers=/var/tmp/eldk.PyrfxY/usr/crosstool/gcc-4.0.
0-glibc-2.3.5-eldk/mips-linux/mips-linux/include --with-local-prefix=
/var/tmp/eldk.PyrfxY/usr/crosstool/gcc-4.0.0-glibc-2.3.5-eldk/
mips-linux/mips-linux --disable-nls --enable-threads=posix
--enable-symvers=gnu --enable-languages=c,c++ --enable-shared
--enable-c99 --enable-long-long --enable-__cxa_atexit
Thread model: posix
gcc version 4.0.0 (DENX ELDK 4.1 4.0.0)
```

A.4.3 安装仿真器

由于实验的操作系统内核是运行在 mips 体系结构上的，而我们平常使用的是基于 x86 体系结构的 PC，所以需要使用仿真器来仿真一个 mips 体系结构的环境，来让我们的操作系统内核运行。在这个实验中我们使用的是 gxemul 仿真器。

我们需要从http://gxemul.sourceforge.net/src/下载 gxemul-0.4.6.tar.gz。这里需要注意，一定要下载 0.4 系列的 gxemul，而不是最新的 0.6 系列，以免发生和实验提供的文件发生不兼容的现象。调出终端，切换到 gxemul-0.4.6.tar.gz 所在目录，执行以下指令

```
# 解压 gxemul-0.4.6.tar.gz
tar -zxvf gxemul-0.4.6.tar.gz
# 进入 gxemul 文件夹
cd gxemul-0.4.6
# 配置并编译安装
./configure
make
sudo make install
sudo cp gxemul /usr/local/bin
```

在命令行中执行 gxemul 指令，看到如下输出则说明 gxemul 安装正确。

```
$ gxemul
GXemul 0.4.6    Copyright (C) 2003-2007  Anders Gavare
Read the source code and/or documentation for other Copyright messages.

usage: gxemul [machine, other, and general options] [file [...]]
   or  gxemul [general options] @configfile
   or  gxemul [userland, other, and general options] file [args ...]

Run  gxemul -h  for help on command line options.

No filename given. Aborting.
```

Exercise A.2 安装交叉编译器及 gxemul，并确认其正常运行。 ■

A.5 实验思考

Thinking A.1 思考以下几条指令有何作用？

- ls -l
- mv test1.c test2.c
- cp test1.c test2.c
- cd ..

■

我们的整个实验环境全部是运行于 Linux 基础上的，且所提供的虚拟机是没有安装图形界面的，需要通过 ssh 远程连接访问。实验的所有操作全部在命令行下完成。因此，在开始实验前，你需要掌握最基本的一些命令，以保证你可以**在命令行下生存**

Thinking A.2 思考 grep 指令的用法，例如如何查找项目中所有的宏？如何查找指定的函数名？ ■

grep 指令可以在文件中**匹配特定的文本**。支持递归匹配（即查找当前目录及子目录下所有文件）、正则表达式等一系列有用的功能。当你需要在整个项目目录中查找某个函数名、变量名等等的时候，grep 将是你手头一个强有力的工具。这里给一个提示，grep 的-a、-i、-r 三个选项使用率较高，可以着重了解一下。

Thinking A.3 思考 gcc 的-Werror 和-Wall 两个参数在项目开发中的意义。 ■

编译器警告很多时候可以帮你发现一些代码上的错误。善用编译器警告可以大大加快你调试的进度。