

实验一：初探Linux与环境配置

实验目的

- 学习如何在虚拟机中使用Linux;
- 学习Linux (Ubuntu) 的使用方法;
- 学习并熟练使用若干Linux指令;
- 掌握Linux内核编译方法;
- 掌握使用gdb调试内核的方法及步骤。

实验环境

- 虚拟机：VMware/VirtualBox
- 操作系统：Ubuntu 20.04.6 LTS

系统的安装形式可以自由选择，双系统，虚拟机都可以，系统版本则推荐使用本文档所用版本。注意：由于Linux各种发行版非常庞杂且存在较大差异，因此本试验在其他Linux发行版可能会存在兼容性问题。如果想使用其他环境（如vlab）或系统（如Arch、WSL等），请根据自己的系统自行调整实验步骤以及具体指令，达成实验目标即可，但其中出现的兼容性问题助教**无法**保证能够一定解决。

实验时间安排

注：此处为实验发布时的安排计划，请以课程主页和课程群内最新公告为准

- 4.4 晚实验课，讲解实验、检查实验
- 4.11 晚实验课，检查实验
- 4.18 晚实验课，检查实验
- 4.25 晚及之后实验课（含4.25），补检查实验

补检查分数照常给分，但会**记录**此次检查未按时完成，此记录在最后综合分数时作为一种参考（即：最终分数可能会低于当前分数）。

检查时间、地点：周二晚18:30~21:30，电三楼406/408。

如何提问

- 请同学们先阅读《提问的智慧》。[原文链接](#)
- 提问前，请先[阅读报错信息](#)、查询在线文档，或百度。[在线文档链接](#)；
- 在向助教提问时，请详细描述问题，并提供相关指令及相关问题的报错截图；
- 在QQ群内提问时，如遇到长时未收到回复的情况，可能是由于消息太多可能会被刷掉，因此建议在在线文档上提问；
- 如果助教的回复成功地帮你解决了问题，请回复“问题已解决”，并将问题及解答更新到在线文档。这有助于他人解决同样的问题。

为什么要做这个实验

- 为什么要学会使用Linux?
 - Linux的安全性、稳定性更好，性能也更好，配置也更灵活方便，所以常用于服务器和开发环境。实验室和公司的服务器一般也都用Linux；
 - Linux是开源系统，代码修改方便，很多学术成果都基于Linux完成；
 - Windows是闭源系统，代码无法修改，无法进行后续实验。
- 为什么要使用虚拟机?
 - 虚拟机对你的电脑影响最低。双系统若配置不正确，可能导致无法进入Windows，而虚拟机自带的快照功能也可以解决部分误操作带来的问题。
 - 本实验并不禁止其他环境的使用，但考虑其他环境（如WSL）变数太大，比如可能存在兼容性或者其他配置问题，会耽误同学们大量时间浪费在实验内容以外的琐事，因此建议各位同学尽量保持与本试验一致或类似的环境。
- 为什么要学会编译Linux内核?
 - 这是后续实验的基础。在后续实验中，我们会让大家通过阅读Linux源码、修改Linux源码、编写模块等方式理解一个真实的操作系统是怎么工作的。

其他友情提示

- **合理安排时间，强烈不建议在ddl前赶实验。**
- 本课程的实验实践性很强，请各位大胆尝试，适当变通，能完成实验任务即可。
- pdf上文本的复制有时候会丢失或者增加不必要的空格，有时候还会增加不必要的回车，有些指令一行写不下分成两行了，一些同学就漏了第二行。如果出了bug，建议各位先仔细确认自己输入指令是否正确。要**逐字符**比对。每次输完指令之后，请观察一下指令的输出，检查一下这个输出是不是报错。请在复制文档上的指令之前先理解一下指令的含义。我们在检查实验时会抽查提问指令的含义。
- 如果你想问“为什么PDF的复制容易出现问題”，请参考 [此文章](#)。
- 如果同学们遇到了问题，请先查询在线文档。在线文档地址：[链接](#)

第一部分：在虚拟机下安装Linux系统

提示：

- 本部分属于初学者指南。我们不限环境的使用。你可以使用双系统或其他linux发行版完成实验。虽然理论上影响不大，**但若其他版本的操作系统在后续实验中出现兼容性问题，可能需要你自己解决。**
- 你可以使用VMware/VirtualBox完成实验。推荐使用Virtualbox，因为它开源且免费。相比之下，VMware的免费版(VMware Workstation Player)不具备快照功能。如果你的虚拟机不慎挂掉，解决起来可能会比较麻烦。但在一些其他课程的实验上，VirtualBox可能有bug。二者的使用方法大同小异，请各位自行权衡。

1.0 若干名词解释

宿主机(host)：主机，即物理机器。

虚拟机：在主机操作系统上运行的一个“子机器”。

Linux发行版：Linux内核与应用软件打包构成的可以使用的操作系统套装。常见的有Ubuntu、Arch、CentOS甚至Android等。

1.1 下载

虚拟机软件（二选一）：

- VMware Workstation Player的下载链接：
<https://customerconnect.vmware.com/zh/downloads/details?downloadGroup=WKST-PLAYER-1622&productId=1039&rPid=82555>
- VirtualBox的下载链接：<https://www.virtualbox.org/wiki/Downloads>，注意根据你的宿主机(host)选取合适的安装包。
- macOS宿主机若想使用其他虚拟机软件，请自行搜索安装教程。

Ubuntu 20.04.6 LTS 安装镜像文件（下载完成之后，你不需要打开镜像文件）：

- 官网链接：<https://releases.ubuntu.com/20.04/ubuntu-20.04.6-desktop-amd64.iso>
- LUG校内镜像，校内下载速度可达几十MB/s：<http://mirrors.ustc.edu.cn/ubuntu-releases/20.04/ubuntu-20.04.6-desktop-amd64.iso>

安装VMware/VirtualBox的步骤较为简单，运行安装程序即可，在此不表。

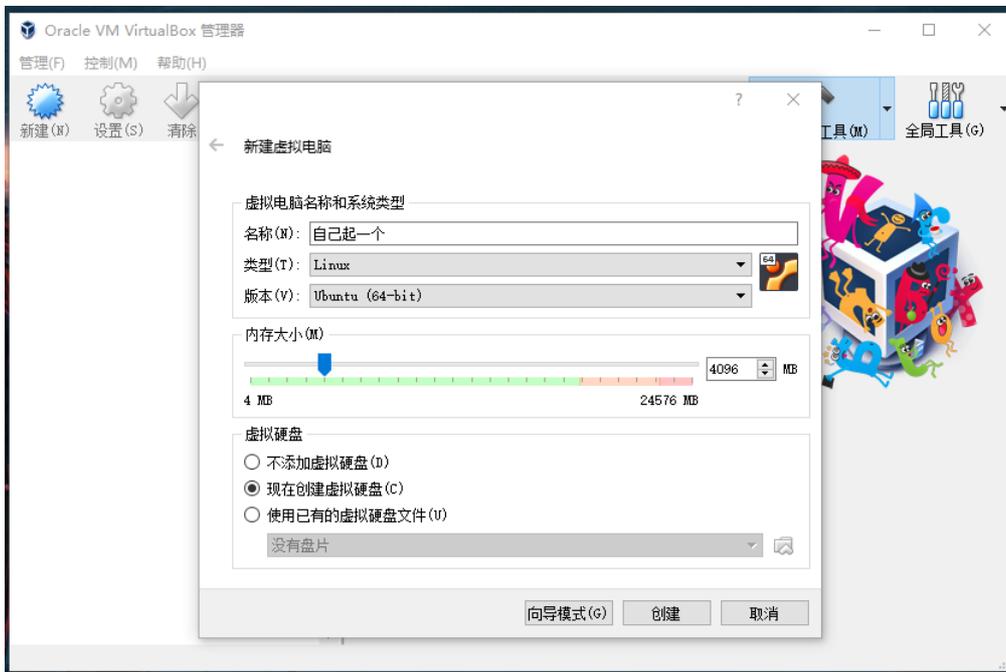
1.2 创建、安装虚拟机 (VirtualBox)

下面介绍VirtualBox创建、安装虚拟机的过程。如果你使用VMware，请直接看下一节。

1.2.1 新建虚拟机

点击“新建”创建虚拟机。设置虚拟机的名称、类型、分配内存等。同时要选择“现在创建虚拟硬盘”。

请至少分配2GB以上的内存给虚拟机。同时建议分配至少1/4主机内存给虚拟机。



1.2.2 创建虚拟硬盘

- 文件位置：考虑到虚拟磁盘大小动辄几十GB，建议将其放在空间有富余的磁盘分区上。如果你有很多不常用的文件占用大量磁盘空间，可以考虑将其转移到 **睿客云盘** 上保存。
- 文件大小：建议30~40G。我们的实验会占用较多的磁盘空间。20GB **非常紧张**。

警告：如果磁盘空间不够，Linux启动会黑屏进不去图形界面，需要在命令模式下删除一些文件后重启才能进入图形界面。一些虚拟机具备“扩展磁盘容量”的功能，但是根据实际测试，发现很多时候反而会让虚拟机直接黑屏。

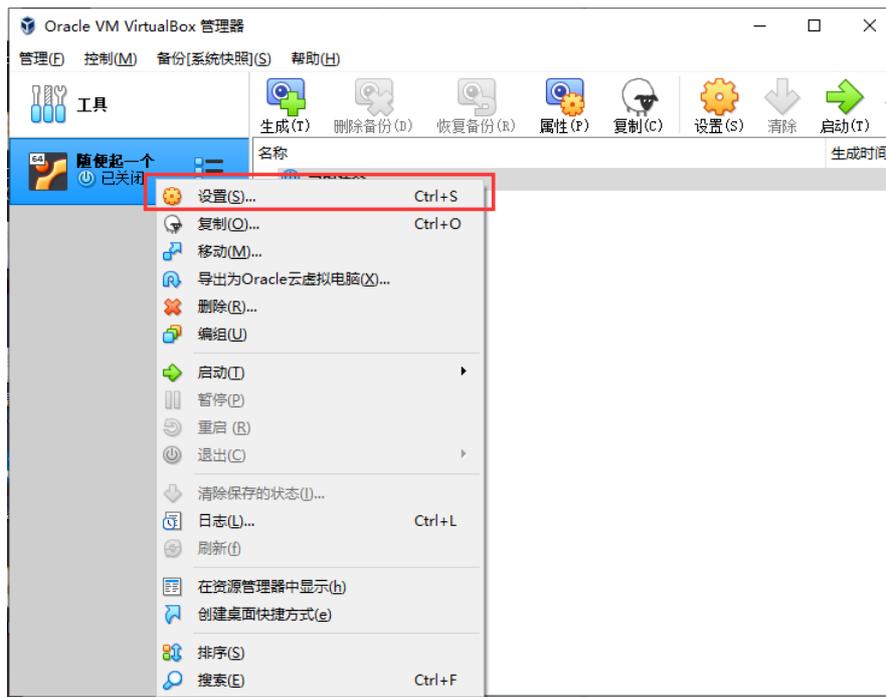
- 虚拟硬盘文件类型：默认即可。
- 动态分配/固定大小：默认即可。
- 最后点击创建。



至此，虚拟机已经创建完毕。点击“启动”即可进入虚拟机。

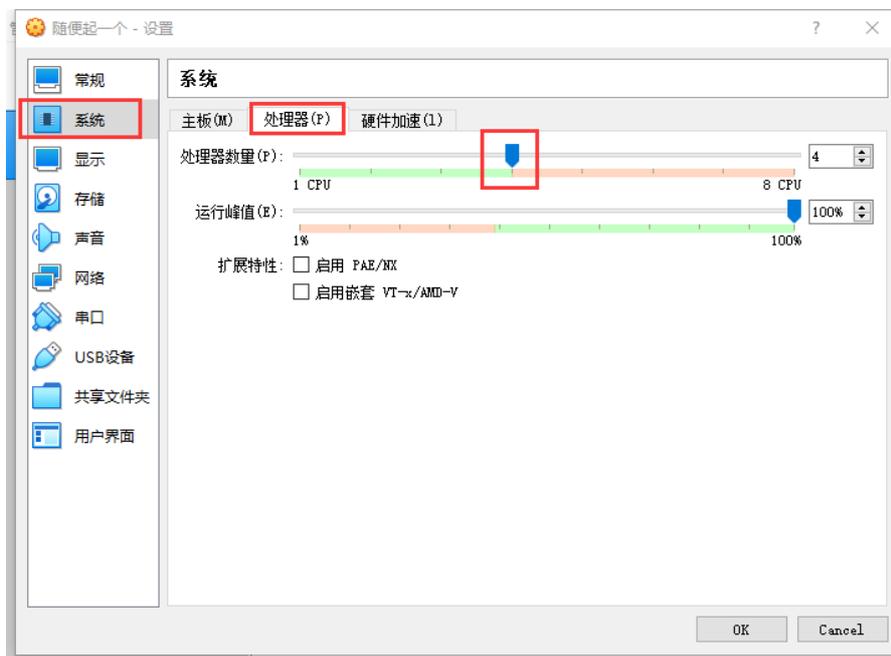
1.2.3 设置CPU数量

选中新建的虚拟机，右键-设置，打开设置界面。



在系统-处理器中将处理器数量调至合适的数量。该选项请根据自己电脑的处理器核数自行调整。

为虚拟机分配更多的CPU内核数量有助于提高虚拟机的性能。注意，给虚拟机分配的内核不是被虚拟机独占的。就算为虚拟机分配宿主机相同的内核数量，也毫无问题。



1.2.4 启动、挂载启动盘

然而，此时的虚拟机只是个空壳——硬盘是空的，里面什么都没有。所以我们需要安装操作系统。

启动虚拟机。虚拟机程序发现磁盘是空的，会自动提示我们挂载启动盘。选择我们下载的Ubuntu安装镜像文件作为启动盘。点击“启动”即可进入安装程序。



1.2.5 安装Ubuntu

1. 等一会之后会进入安装界面。你可以在左边把安装界面切成中文，然后点“安装Ubuntu”。

如果在安装时发现“继续”、“后退”、“退出”等按钮在屏幕外，请先按 **Alt+F7**，然后松开键盘，再移动鼠标以拖动窗口。点击鼠标会使窗口拖动停止。

2. 键盘布局选择Chinese即可。
3. 这里不建议选“安装Ubuntu时下载更新”。因为国内默认的下载源速度较慢，换源之后速度才快。



4. 因为虚拟机的磁盘本来就是空的，所以安装类型选择“清除整个磁盘并安装Ubuntu”。然后点现在安装-继续。

警告：在安装双系统时，不要选这个，否则后果自负。

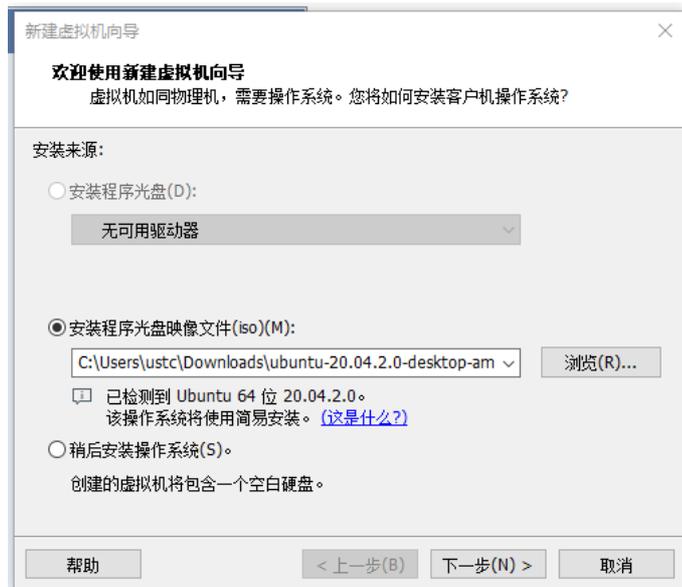
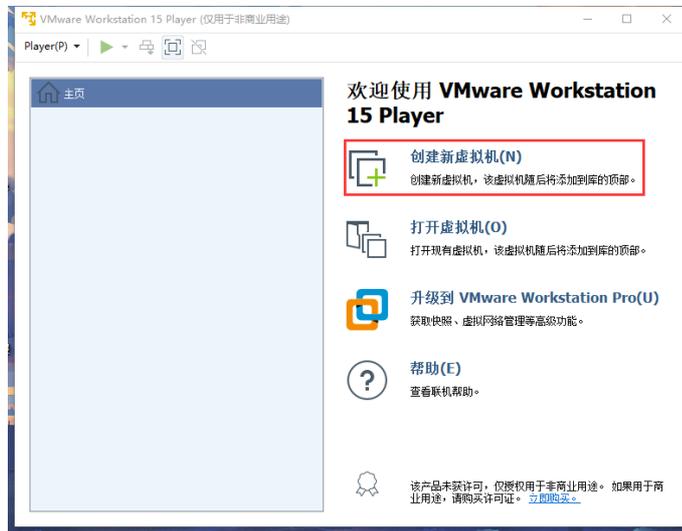
5. 时区位置默认上海即可。
6. 随便编一个姓名、计算机名、用户名，然后设置密码。

警告：请一定要记住密码。否则会进不去系统。

7. 等安装完即可。安装完成之后系统会提示重启。如果重启之后系统提示需要移除安装光盘，对于VirtualBox，在上方控制-设置-存储里检查“控制器：IDE”里有没有安装镜像即可。如果有，就在右侧“分配光驱”里移除虚拟盘；如果是“没有盘片”，就直接在虚拟机中回车重启。

1.3 创建、安装虚拟机 (VMware)

1. 直接使用下载的Ubuntu镜像文件进行简易安装。



2. 设置计算机名、用户名、密码。

警告：请一定要记住密码。否则会进不去系统。



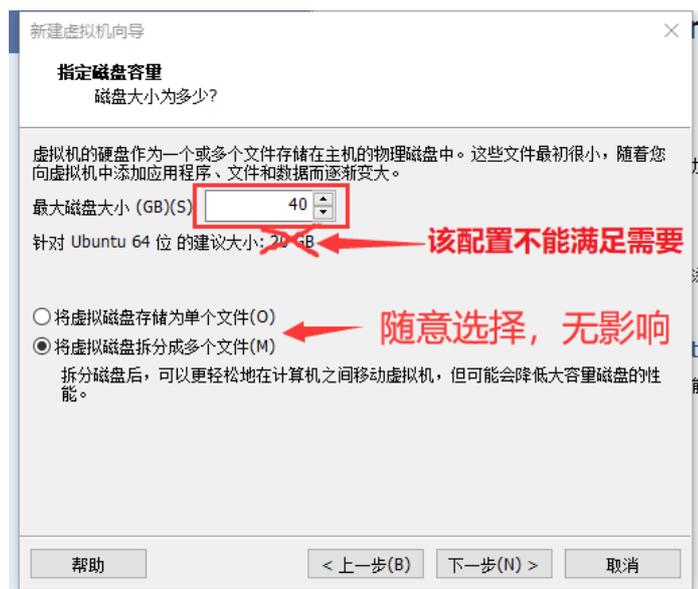
3. 设置虚拟机名称和文件存放位置。

考虑到虚拟磁盘大小动辄几十GB，建议将其放在空间有富余的磁盘分区上。



4. 最大磁盘大小：建议30~40G。我们的实验会占用较多的磁盘空间。20GB **非常紧张**。你可以随意选择是否拆分磁盘的选项。如果你有很多不常用的文件占用大量磁盘空间，可以考虑将其转移到 **睿客云盘** 上保存。

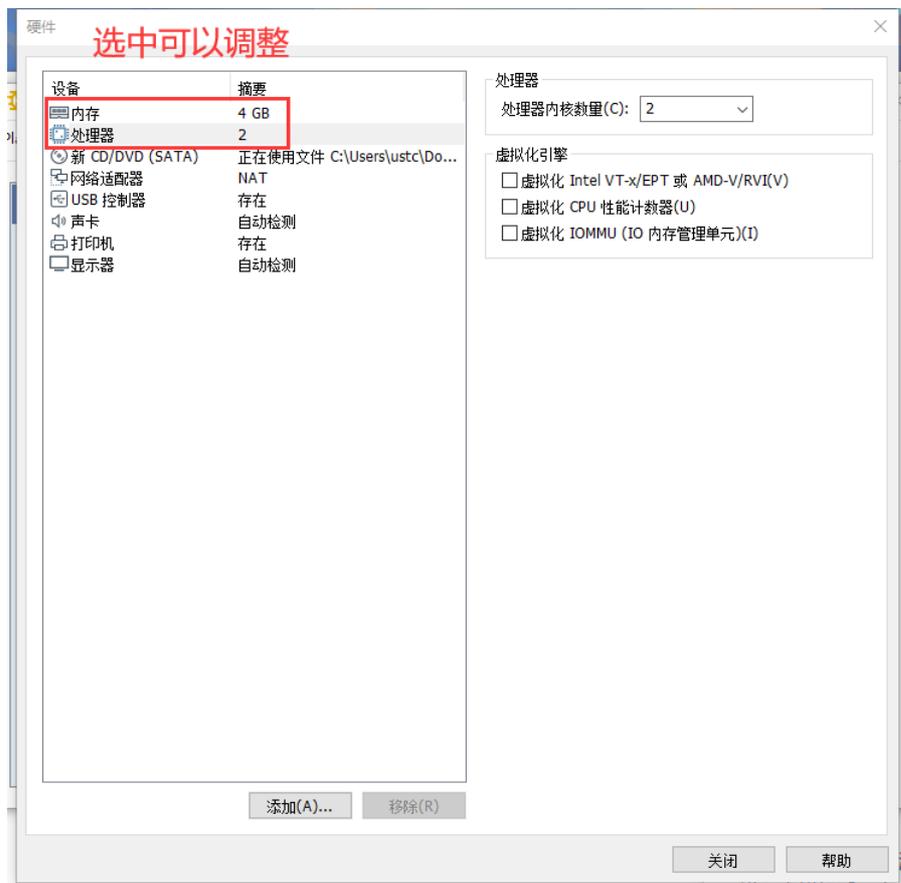
警告：如果磁盘空间不够，Linux启动会黑屏进不去图形界面，需要在命令模式下删除一些文件后重启才能进入图形界面。一些虚拟机具备“扩展磁盘容量”的功能，但是根据实际测试，发现很多时候反而会让虚拟机直接黑屏。



5. 可以在“自定义硬件”内自己设置内存、处理器核数等设置。

请至少分配2GB以上的内存给虚拟机。同时建议分配至少1/4主机内存给虚拟机。

为虚拟机分配更多的CPU内核数量有助于提高虚拟机的性能。注意，给虚拟机分配的内核不是被虚拟机独占的。就算为虚拟机分配宿主主机相同的内核数量，也毫无问题。



6. 点击“完成”即可自动安装Ubuntu系统。之后就不需要操作了，直接等安装完毕后虚拟机重启就可以愉快地使用Ubuntu了。

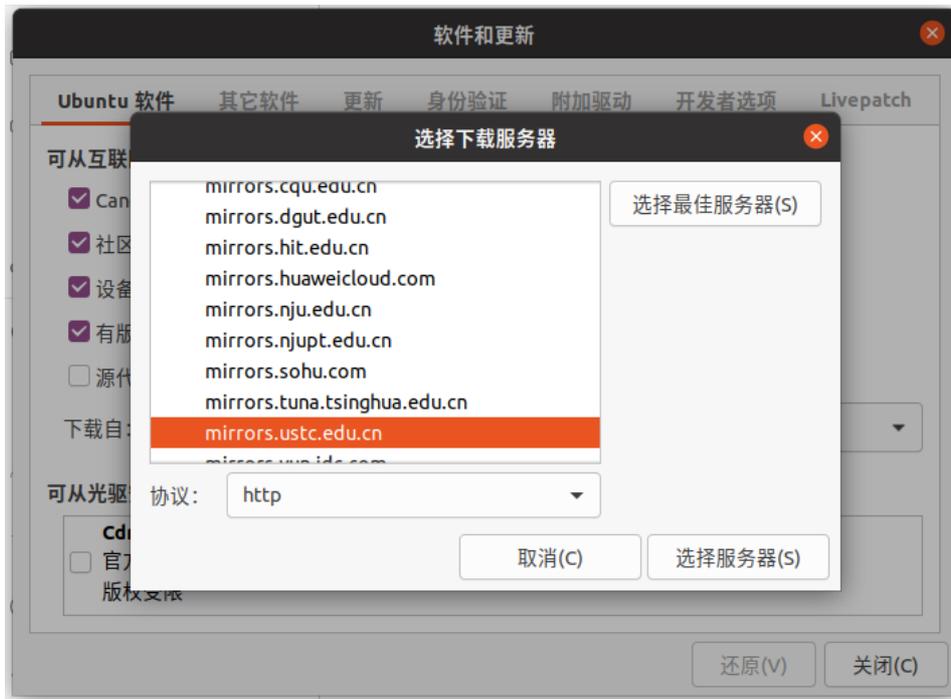
1.4 其他必要设置和使用注意事项

1.4.1 换源

Ubuntu自带的软件源较慢，这会导致我们安装软件包时花更多的时间下载。所以要更换软件源为科大镜像。进入虚拟机后，点击左下角的进入应用菜单，找到并进入“软件更新器”。进入之后它会检查更新，最后会跳出一个“是否向安装更新”的提示。**不要安装**，并点击“设置”。



更改“Ubuntu”软件选项卡的“下载自”为“其他站点”，在弹出的“选择下载服务器”窗口中选择“中国-mirrors.ustc.edu.cn”。输入密码即可完成修改。



设置之后，如果提示更新软件包缓存，请选择更新，并等待更新结束再安装其他软件包/语言包。如果提示更新系统，也可以放心地选择更新而不必担心用时过长。

1.4.2 设置自动调整分辨率、文件拖放

- 对于VirtualBox，若要设置自动调整分辨率，请安装增强功能。可以参考此链接：<https://ywnz.com/linuxjc/2410.html>
- 对于VirtualBox，若要实现文件拖放，请安装扩展包。到官网下载页面(<https://www.virtualbox.org/wiki/Downloads>) 找“VirtualBox 6.1.32 Oracle VM VirtualBox Extension Pack”，下载后双击安装即可。然后在上面的“设备”菜单中设置拖放和剪贴板为“双向”。

虽然不支持将文件拖放到桌面，但你可以尝试打开Ubuntu的文件管理器，并将文件拖放至他处。

- VMware无需执行此步骤。因为VMware会自动安装VMware tools，但是如果发现调整不了虚拟机分辨率、无法共享粘贴板等情况，是自动安装失败（比如网络问题），需要手动安装。请参考此链接：https://blog.csdn.net/qq_64092369/article/details/123050107

VirtualBox/VMware的文件拖放经常会出问题，目前并没有通用的解决方案，因此建议用U盘、共享文件夹、睿客云盘之类实现文件中转。

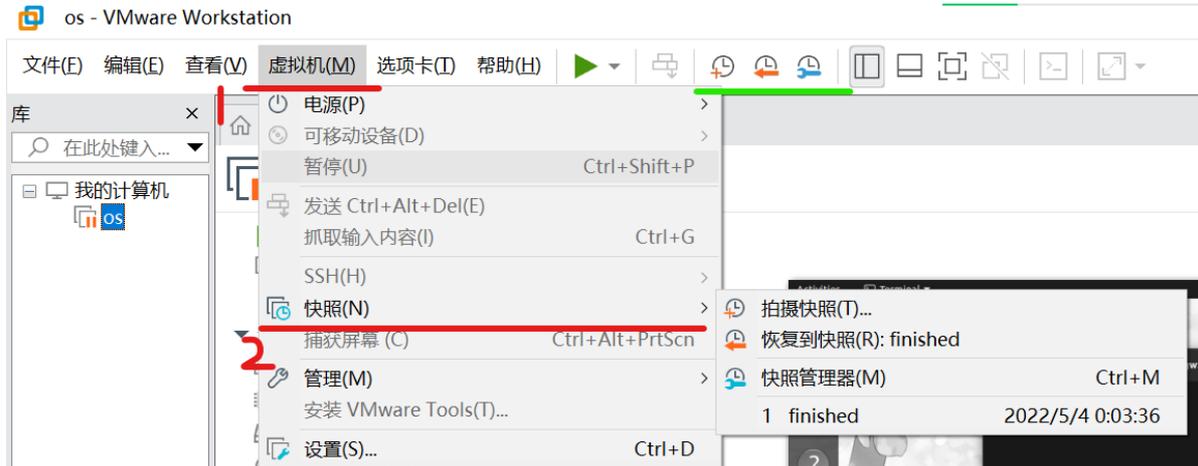
1.4.3 修改语言 (VMware 简易安装)

- 请参考此链接：https://blog.csdn.net/langshi_2011/article/details/78993781
- 请尤其注意链接的第六步。

1.4.4 快照是个好东西

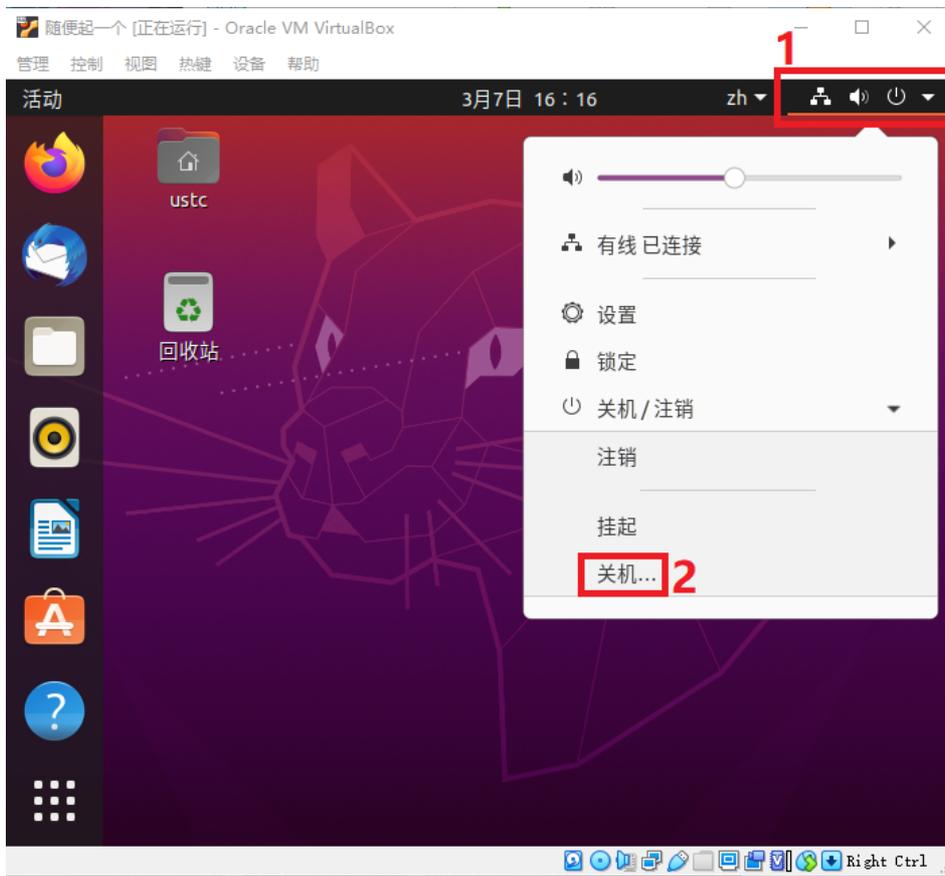
VMware提供了一个方便的功能——**快照**功能。**快照**，即虚拟机磁盘文件在某个时间点的副本，可以理解为保存了当时虚拟机的状态。后续如果代码改得乱七八糟或者虚拟机出问题了，可以直接选择之前保存的快照进行恢复，这样就可以让虚拟机回到你保存的那个状态，和当时的虚拟机完全一样。

具体操作非常简单，如下图，在菜单栏中找到 **虚拟机** 选项，在该菜单中有 **快照** 选项，在这里就可以对快照进行操作（图中红线）。同时菜单栏中也有快捷按钮（下图中绿线）：



1.4.5 如何关机

如何关闭Ubuntu：如下图所示，点屏幕右上角-关机。



直接点虚拟机右上角的叉也可以关机。

- “快速休眠”是保存虚拟机状态，下次启动虚拟机时可以直接恢复上次状态；（建议使用）

- “正常关闭”相当于按电脑的电源键关机；
- “强制退出”相当于拔掉电脑电源进行硬关机。

第二部分：初探Linux

考虑到很多同学在本次实验之前没有使用过Linux系统，因此今年增加此部分来速成Linux。

2.1 Ubuntu GUI的使用

打开虚拟机进入到Ubuntu之后，即可看到Ubuntu的GUI界面。默认左侧是Dock（类似Windows的任务栏），里面有若干内置软件。左下角是菜单（类似Windows的开始菜单）。考虑到部分同学首次接触Ubuntu，因此建议各位依次点击所有的软件、按钮，以进一步了解Ubuntu并熟悉其中的软件。

屏幕最右上角有几个图标，可以调整音量、网络设置、语言、输入法等，还可以关机。在菜单里找到“设置”，里面可以调整系统设置，如分辨率、壁纸等。Dock里有一个文件夹形状的图标，它是文件管理器，可以像windows一样图形化地浏览文件。

提示：如果在执行某个操作时报错文件/文件夹不存在，可以在UI界面内手动复制粘贴文件到目标位置（文件所有者为root用户的除外）。

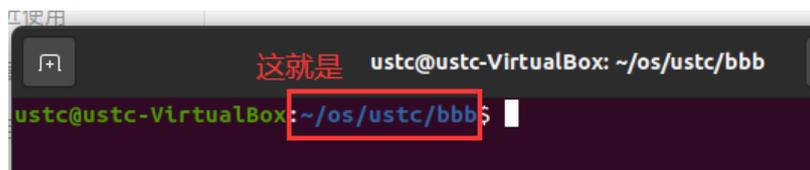
2.2 终端（命令行）的使用

2.2.1 打开终端

在桌面/文件管理器的空白处右键即可出现“在终端打开”按钮，点击此处即可呼出终端进而在终端中执行相关的Linux指令。注意：终端也是有工作位置的。简单而言，在哪个目录下打开终端，命令就会在哪个目录下执行。终端当前目录一般称为“**工作目录**”。

举例：`ls` 指令可以显示工作目录下的文件。在不同的目录下运行此指令的结果显然是不一样的。

终端会显示工作目录，如图所示：



2.2.2 目录与路径

Linux与Windows不同，Windows一般会分有多个逻辑磁盘，每个逻辑磁盘各有一棵目录树，但Linux只有一个目录树，磁盘可以作为一棵子树**挂载**到目录树的某个节点。在Linux操作系统中，整个目录树的根节点被称为**根目录**。每个用户拥有一个**主目录**，或称**家目录**，类似windows的 `C:\Users\用户名`。从根目录看，除root用户之外，每个用户的家目录为 `/home/用户名`。

Linux终端里使用的路径分为**绝对路径**和**相对路径**两种。绝对路径指从根目录算起的路径，相对路径指从工作目录算起的路径。其中，以根目录算起的绝对路径以 `/` 开头，以家目录算起的绝对路径以 `~` 开头，相对路径不需要 `/` 或 `~` 开头。

举例：一个名为ustc的用户在他的家目录下创建了一个名为os的目录，在os目录下面又创建了一个名为lab1的目录，则该lab1目录可以表示为：

- `/home/ustc/os/lab1`
- `~/os/lab1`
- 如果工作目录在家目录：`os/lab1`
- 如果工作目录在os目录：`lab1`

一些特殊的目录：

- `.` 代表该目录自身。例：`cd .` 代表原地跳转（`cd` 是切换目录的指令）；
- `..` 代表该目录的父目录。特别地，根目录的父目录也是自身。
- 在Linux里，文件名以 `.` 开头的文件是隐藏文件（或目录），如何显示它们请参考2.3.3。
 - `.` 和 `..` 都是隐藏的目录。

举例：以下几个路径是等价的：

- `/home/ustc/os/lab1`
- `/home/ustc/os/././././././lab1`
- `/home/ustc/os/./os/lab1`

2.2.3 运行指令

在终端中输入可执行文件的路径即可。终端所在路径是程序运行时的路径。需要注意的是，如果运行的二进制文件就在工作目录下，需要在文件名前加上 `./`。

提示：与windows可执行文件扩展名为.exe不同，Linux中，可执行文件一般没有扩展名。例外：`gcc`在不指定输出文件名的情况下，编译出的可执行文件会带有.out的扩展名。但你也不需要管这个.out，直接运行也是一样的。

例：有一个可执行文件，其路径为 `~/os/lab1/testprog`。它在运行时会读取一个相对路径为 `a.txt` 的文件。

- 在家目录下运行：需要执行 `os/lab1/testprog`（或 `~/os/lab1/testprog` 等绝对路径），程序读取的 `a.txt` 在家目录下；
- 在 `~/os/lab1` 下运行：需要执行 `./testprog`（当然你用绝对路径也无所谓），程序读取的 `a.txt` 在 `~/os/lab1` 目录下。

相关问题：为什么在运行 `sudo`、`man` 等命令时，只需要输入指令名而不需要输入这些指令对应的二进制文件所在的路径？

一种情况是，这是因为这些指令所在的路径（一般是 `/usr/bin`）被加入到了该用户的**环境变量**中。当终端读取到一个不带路径的命令之后，系统只会在环境变量中搜索，从而方便用户使用。当然，默认被加入环境变量里的路径一般只有一些系统路径，除非自行设置，家目录下面没有目录默认在环境变量中。如感兴趣，修改环境变量的方法可自行搜索了解。

另一种情况是，部分指令是Shell内建指令（如 `cd`），它们的意义直接由Shell解释，没有对应的二进制文件。此情况可能会在下一实验中详细阐述。

2.2.4 指令及其参数

无论是Linux还是Windows，一条完整的命令都由命令及其参数构成。你们可以通过 `man 指令名` 来自了解指令语法。一般来说，指令语法里带有 '[']' 的是可选参数，其他是必选参数。不同的参数的顺序一般是可以互换的。下面以 `gcc`（一种编译器）为例，介绍实验文档描述指令的方式，以及如何按需构造一条指令。

本次要用到的 `gcc` 指令的一部分语法是：`gcc [-static] [-o outfile] infile`。下面是各参数介绍：

参数	含义
<code>-static</code>	静态编译选项（此处参数仅为示例，参数详细含义请自行上网搜索）。
<code>-o outfile</code>	指定输出的可执行文件的文件名为outfile。如果不指定，会输出为a.out。
<code>infile</code>	要编译的gcc文件名。注意绝对路径/相对路径的问题。

- 如果我们编译test.c，不指定输出文件名，命令就只是 `gcc test.c`；（这种情况下，gcc会自动命名输出文件为a.out）

- 如果我们编译test.c, 输出二进制文件名为test, 命令就是 `gcc -o test test.c` ;
 - 一般来说参数的顺序是无所谓的。所以使用 `gcc test.c -o test` 也一样能编译。
- 如果我们编译test.c, 输出二进制文件名为test, 且要使用静态编译, 那么构造出的编译指令就是 `gcc -static -o test test.c` 。

2.2.5 终端使用小技巧

- 按键盘的 `↑↓` 键可以切换到之前输入过的指令;
- 按键盘的 `Tab` 键可以自动补全。如果按一下Tab之后没反应, 说明候选项太多。再按一下Tab可以显示所有候选项。
- 在shell里, `Ctrl+C` 是终止不是复制。复制的快捷键是 `Ctrl+Insert` 或 `Ctrl+Shift+C` , 粘贴的快捷键是 `Shift+Insert` 或 `Ctrl+Shift+V` 。

2.3 Linux常用指令

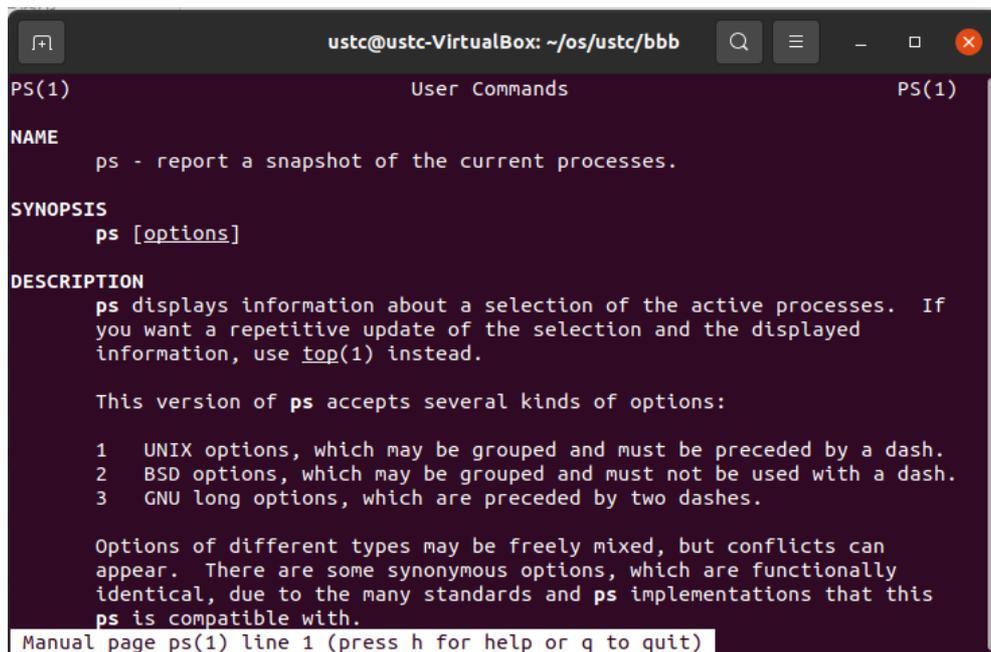
注: 一些指令的使用方法详见提供的链接。本部分涉及测验考察, 测验方式见文档4.2。

2.3.1 man

英文缩写: manual

如果你不知道一条命令的含义, 使用 `man xxx` 可以显示该命令的使用手册。

举例: `man ps` 可以显示 `ps` 指令的使用方法。指令输出如下图。按q退出手册。



```
ustc@ustc-VirtualBox: ~/os/ustc/bbb
PS(1) User Commands PS(1)
NAME
  ps - report a snapshot of the current processes.
SYNOPSIS
  ps [options]
DESCRIPTION
  ps displays information about a selection of the active processes.  If
  you want a repetitive update of the selection and the displayed
  information, use top(1) instead.

  This version of ps accepts several kinds of options:

  1  UNIX options, which may be grouped and must be preceded by a dash.
  2  BSD options, which may be grouped and must not be used with a dash.
  3  GNU long options, which are preceded by two dashes.

  Options of different types may be freely mixed, but conflicts can
  appear.  There are some synonymous options, which are functionally
  identical, due to the many standards and ps implementations that this
  ps is compatible with.
Manual page ps(1) line 1 (press h for help or q to quit)
```

2.3.2 sudo

TL;DR: `sudo` = “以管理员模式运行”。A joke

Linux采用**用户组**的概念实现访问控制, 其中, 只有root用户组才具备管理系统的权限。在 `sudo` 出现之前, 一般用户管理linux系统的方式是, 先用 `su` 指令切换到root用户, 然后在root用户下进行操作。但是使用 `su` 的缺点之一在于必须先告知root用户的密码, 且这种控制方式不够精细。

为了方便操作, 并更加精确地控制权限, `sudo` 用来将root用户的部分权限让渡给普通用户。普通用户只需要输入自己的密码, 确认“我是我自己”, 就能执行自己拥有的那部分管理员权限。特别地, 在Ubuntu下, 普通用户默认可以通过 `sudo` 取得root用户的所有权限。若想精确地控制每个用户能用 `sudo` 干什么, 可以参阅 `visudo` 。

如果在输入某个指令后，系统提示权限不够(Permission Denied)，那么在指令前加上 `sudo` 一般都能解决问题。

注意1: root用户具备很高的权限。一些需要管理员权限才能执行的指令（如，删除整个磁盘上的内容）会破坏系统。所以在使用 `sudo` 时，请务必确认输入的指令没问题。

注意2: 为了防止旁窥者获知密码长度，linux下输入密码不会在屏幕上给出诸如***的回显。输完密码敲回车就行了。

常见使用方法: `sudo command`

例: 以普通用户运行 `apt install vim`，会被告知没有权限，因为只有root用户（管理员）才有资格安装软件包。正确的用法是 `sudo apt install vim`。

2.3.3 ls

英文全拼: list files

显示特定目录中的文件列表。常见的一部分语法是: `ls [-a] [name]`。参数含义详见 [Linux ls命令](#)。

2.3.4 cd

英文全拼: change directory

切换工作目录。常见的一部分语法是: `cd [name]`。参数含义详见 [Linux cd命令](#)。特别地，常见使用 `cd -` 来返回上次到达的目录。

2.3.5 pwd

英文全拼: print work directory

输出工作目录的绝对路径。常见的使用方法是: `pwd`。

2.3.6 rm

英文全拼: remove

删除一个文件或目录。常见的一部分语法是: `rm [-rf] name`。参数含义详见 [Linux rm命令](#)。

2.3.7 mv

英文全拼: move

移动一个文件或目录，或重命名文件。常见的一部分语法是: `mv source dest`。参数含义详见 [Linux mv命令](#)。

2.3.8 cp

英文全拼: copy

复制一个文件或目录，或重命名文件。常见的一部分语法是: `cp [-r] source dest`。参数含义详见 [Linux cp命令](#)。

2.3.9 mkdir

英文全拼: make directory

创建目录。常见的一部分语法是: `mkdir [-p] dirName`。

参数含义详见 [Linux mkdir命令](#)。

2.3.10 cat

英文全拼: concatenate

一般用于将文件内容输出到屏幕。常见的使用方法是: `cat fileName`。

2.3.11 kill

用于将指定的信息发送给程序, 通常使用此指令来结束进程。强制结束进程的信号编号是9, 所以强制结束进程的使用方法是: `kill -9 [进程编号]`。

其他参数的使用方法可以参考 [Linux kill命令](#)。

2.3.12 ps

英文全拼: process status

用于显示进程状态 (任务管理器)。常见的使用方法:

- `ps`: 显示**当前用户**在当前终端控制下的进程。考虑到用户通常想看不止当前用户和当前终端下的进程, 所以不加参数的用法并不常用。
- `ps aux`: 展示所有用户所有进程的详细信息。注意, a前面带一个横线是严格意义上不正确的使用方法。
- `ps -ef`: 也是展示所有用户所有进程的详细信息。就输出结果而言和 `ps aux` 无甚差别。

2.3.13 wget

wget是一个在命令行下下载文件的工具。常见的使用方法是: `wget [-O FILE] URL`。

例如, 我们要下载 <https://git-scm.com/images/logo@2x.png> 到本地。

1. 直接下载: `wget https://git-scm.com/images/logo@2x.png`, 文件名是logo@2x.png。
2. 直接下载并重命名: `wget -O git.png https://git-scm.com/images/logo@2x.png`, 文件名是git.png。注: -O中的O表示字母O而不是数字0。

注意, 如果发现有同名文件, 1所示方法会在文件名后面加上.1的后缀进行区分, 而2所示方法会直接覆盖。

2.3.14 tar

用于压缩、解压压缩包。常见使用方法:

- 把某名为source的文件或目录压缩成名为out.tar.gz的gzip格式压缩文件: `tar zcvf out.tar.gz source`
- 解压缩某名为abc.tar.gz的gzip格式压缩文件: `tar zxvf abc.tar.gz`

其他使用方法详见 [Linux tar命令](#)

2.3.15 包管理器 (apt等)

在Linux下, 如何安装软件包? 每个Linux发行版都会自带一个**包管理器**, 类似一个“软件管家”, 专门下载免费软件。

不同Linux发行版附带的包管理器是不一样的。如, Debian用apt (Ubuntu是基于Debian的, 所以也用apt), ArchLinux用Pacman, CentOS用yum, 等等。

apt的常见用法:

- 安装xx包, yy包和zz包: `sudo apt install xx yy zz`
- 删除xx包, yy包和zz包: `sudo apt remove xx yy zz`
- 更新已安装的软件包: `sudo apt upgrade`
- 从软件源处检查系统中的软件包是否有更新: `sudo apt update`

其他使用方法参见 [Linux apt命令](#)。

其他提示：

- 如果报错“无法获得锁 /var/lib/dpkg/lock.....”，这是因为系统在同一时刻只能运行一个 apt，请耐心等待另一边安装/更新完。Ubuntu的软件包管理器也是一个apt。若你确信没有别的 apt在运行，可能是因为没安装完包就关掉了终端或apt。请重启Linux或参考 [此链接](#)。
- 如果报错“下列软件包有未满足的依赖关系.....”或“没有可用的软件包.....”，但是它被其他软件包引用了.....，可能是因为刚换了源，没等包刷新完就关闭窗口，请手动 `sudo apt-get update`。

2.3.16 文字编辑器 (vim、gedit等)

Linux系统中，常见的使用命令行的文字编辑器是vim。系统一般并不自带vim，需要使用包管理器安装。由于vim的使用方法与我们习惯的GUI编辑方式有很大差异，所以在这里不详细介绍它的用法。若想了解请参考 <https://www.runoob.com/linux/linux-vim.html>。

gedit是Ubuntu使用的Gnome桌面环境自带的一款文本编辑器，其使用方法与Windows的记事本(Notepad)大同小异。在这里也不多介绍。在终端里输入 `gedit` 回车即可启动gedit。编辑特定的文件的使用方法是 `gedit 文件名`，若输入的文件名不存在，将自动创建新文件。在Ubuntu的图形化界面中直接双击文本文件也可以编辑文件。但需要注意的是，在编辑一些需要root权限的文件时，直接双击文件不能编辑，只能在终端里 `sudo gedit 文件名`。

在命令行下启动gedit会在终端里报warning，忽略即可。 [参考链接](#)

但是gedit实际上并不是写代码的最佳选择，如果想使用一些更高级的编辑器，可以考虑安装vscode。安装方法请自行到网上搜索。

如果想安装高级IDE的话（不推荐这么做），只能用CLion等。Visual Studio不支持Linux。

在后续的实验中，如果某步骤写着 `vim xxx`，说明这是让你编辑某文件。编辑文件的方式不仅局限于vim，用gedit等也可。

2.3.17 编译指令 (gcc、g++、make等)

我们在编译代码时，需要使用编译器。`gcc` 和 `g++` 是常见的编译命令。其中：

- `gcc` 会把 `.c` 文件当作C语言进行编译，把 `.cpp` 文件当作C++语言编译。
- `g++` 会把 `.c` 和 `.cpp` 文件都当作C++语言编译。

考虑到复杂工程需要编译的文件数量会很多，此时每次都手输编译命令较为繁琐，为此GNU提供了一个make工具，可以按照一个编写好的Makefile文件来完成编译任务。本课程实验应该不会涉及让学生自行从头编写一个Makefile，但涉及使用 `make` 命令。make的工作原理、Makefile的编写方法可以参考 [这个链接](#)。

注意：

1. 这里的“C语言”是C90标准的C语言，不能使用STL、类、引用、for内定义变量等C++特性。
2. Linux内核是使用上面所述的C语言编写的，而不是C++。
3. 在编写代码时，请注意代码文件的扩展名命名，和编译指令的选择。
4. `gcc` 和 `g++` 默认是不安装的。如果你想使用，请先使用包管理器安装 `build-essential` 包，里面包括 `gcc`、`g++` 等常见编译器，和make工具等。

简单地使用 `gcc / g++` 编译的语法是：`gcc [-o outfile] infile`。下面是指令的各个参数介绍。如需了解更详尽的使用方法，可参考gcc官方手册：<http://www.gnu.org/software/gcc/>。

参数	含义
-o outfile	指定输出的可执行文件的文件名为outfile。如果不指定，会输出为a.out。
infile	要编译的gcc文件名。注意绝对路径/相对路径的问题。

使用 `make` 的方法是：在工程目录下直接运行 `make` 即可。一些Makefile会提供多种编译选项，如删除编译好的二进制文件(`make clean`)、编译不同的文件等。这时候需要根据Makefile来确定不同的编译选项。在后面的实验中，如有此方面的需要，我们将给出使用方法。

PS：Makefile文件编写的简单介绍。

虽然本课程可能不涉及Makefile文件的编写，但是作为工程项目非常实用的一项工具，打算在这里进行简单的介绍。（要是学习任务紧张，可以直接跳过，接着看2.4）

Makefile文件的格式比较简单，由若干个如下的指令块组成：

```
target ... : prerequisites ...
    command
    ...
    ...
```

target即为当前指令块要生成的目标文件；prerequisites为当前目标所依赖的文件，可以是文件系统中实际存在的文件，也可以是当前Makefile文件中其他目标；command为任意shell指令，一般是 `gcc` 或 `g++` 等编译指令，当然有时候也会用到其他指令。

举一个简单例子大家就会更清楚其工作过程了：

```
foo.o: foo.c
    gcc -Wall -c foo.c -o foo.o
bar.o: bar.c
    gcc -Wall -c bar.c -o bar.o
main.o: main.c
    gcc -Wall -c main.c -o main.o
app.executable: foo.o bar.o main.o
    gcc main.o foo.o bar.o -lpthread -o app.executable
```

本来我们在调试的过程中需要反复执行这四条指令，但是现在只需要执行 `make app.executable` 一条指令即可，`make` 工具会根据我们的目标所需的依赖，自动执行对应的指令，这样可以节省我们大量的时间。

当然，对于一个庞大的工程，一个程序可能有无数依赖，程序之间的依赖关系可能极其复杂，自己编写Makefile也会成为一项复杂的工作，这时就会用到 `CMakeLists` 工具自动生成Makefile，这里就不详细介绍了，推荐一个教程 [CMake应用：CMakeLists.txt完全指南](#)，大家可以自行学习。

2.4 Shell 脚本

考虑到往届在检查实验时，大多数时间都浪费在了敲指令上，今年尝试教学生如何编写Shell脚本。

2.4.1 Shell脚本的编写

Shell脚本类似于Windows的.bat批处理文件。一个最简单的Shell脚本长这样：

```
#!/bin/bash
第一条命令
第二条命令
第三条命令，以此类推
```

其中，第一行的 `#!/bin/bash` 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。Ubuntu下默认的shell是bash。脚本一般命名为 `xxx.sh`。

2.4.2 脚本的运行

运行Shell脚本有两种方式。

1. 通过执行 `sh xxx.sh` 来直接调用解释器运行脚本。其中，`sh`就是我们的Shell。这种方式运行的脚本，不需要在第一行指定解释器信息。
2. 将该脚本视为可执行程序。首先，保存脚本之后，要通过 `chmod +x xxx.sh` 给脚本赋予执行权限，然后直接 `./xxx.sh` 运行脚本。

注意：脚本的执行和在终端下执行这些语句是完全一致的，依然需要注意绝对路径和相对路径的问题。

举例：首先编译`abc.cpp`并输出一个名为`aabbcc`的二进制可执行文件，然后执行`aabbcc`，最后删掉`aabbcc`，上述操作可以使用如下脚本来实现：

```
#!/bin/bash
gcc -o aabbcc abc.cpp
./aabbcc
rm aabbcc
```

之后的实验会介绍更多Shell脚本的语法。

2.5 参考资料

- Linux命令大全：<https://www.runoob.com/linux/linux-command-manual.html>
- Shell教程：<https://www.runoob.com/linux/linux-shell.html>

第三部分：编译、调试Linux内核

3.1 先导知识

3.1.1 系统内核启动过程

Linux kernel在自身初始化完成之后，需要能够找到并运行第一个用户程序（此程序通常叫做“init”程序）。用户程序存在于文件系统之中，因此，内核必须找到并挂载一个文件系统才可以成功完成系统的引导过程。

在grub中提供了一个选项“root=”用来指定第一个文件系统，但随着硬件的发展，很多情况下这个文件系统也许是存放在USB设备，SCSI设备等等多种多样的设备之上，如果需要正确引导，USB或者SCSI驱动模块首先需要运行起来，可是不巧的是，这些驱动程序也是存放在文件系统里，因此会形成一个悖论。

为解决此问题，Linux kernel提出了一个RAM disk的解决方案，把一些启动所必须的用户程序和驱动模块放在RAM disk中，这个RAM disk看上去和普通的disk一样，有文件系统，有cache，内核启动时，首先把RAM disk挂载起来，等到init程序和一些必要模块运行起来之后，再切换到真正的文件系统之中。

但是，这种RAM disk的方案（下称initrd）虽然解决了问题但并不完美。比如，disk有cache机制，对于RAM disk来说，这个cache机制就显得很多余且浪费空间；disk需要文件系统，那文件系统（如ext2等）必须被编译进kernel而不能作为模块来使用。

Linux 2.6 kernel提出了一种新的实现机制，即initramfs。顾名思义，initramfs只是一种RAM filesystem而不是disk。initramfs实际是一个cpio归档，启动所需的用户程序和驱动模块被归档成一个文件。因此，不需要cache，也不需要文件系统。

3.1.2 什么是initramfs

initramfs 是一种以 cpio 格式压缩后的 rootfs 文件系统，它通常和 Linux 内核文件一起被打包成 boot.img 作为启动镜像。

BootLoader 加载 boot.img，并启动内核之后，内核接着就对 cpio 格式的 initramfs 进行解压，并将解压后得到的 rootfs 加载进内存，最后内核会检查 rootfs 中是否存在 init 可执行文件（该init文件本质上是一个执行的 shell 脚本），如果存在，就开始执行 init 程序并创建 Linux 系统用户空间 PID 为 1 的进程，然后将磁盘中存放根目录内容的分区真正地挂载到 / 根目录上，最后通过 `exec chroot . /sbin/init` 命令来将 rootfs 中的根目录切换到挂载了实际磁盘分区文件系统中，并执行 /sbin/init 程序来启动系统中的其他进程和服务。

基于ramfs开发的initramfs取代了initrd。

3.1.3 什么是initrd

initrd代指内核启动过程中的一个阶段：临时挂载文件系统，加载硬盘的基础驱动，进而过渡到最终的根文件系统。

initrd也是早期基于ramdisk生成的临时根文件系统的名称。现阶段虽然基于initramfs，但是临时根文件系统也依然存在某些发行版称其为initrd。例如，CentOS 临时根文件系统命名为 `initramfs-uname -r.img`，Ubuntu 临时根文件系统命名为 `initrd-uname -r.img`（`uname -r`是系统内核版本）。

3.1.4 QEMU

QEMU是一个开源虚拟机。可以在里面运行Linux甚至Windows等操作系统。

本次实验需要在虚拟机中安装QEMU，并使用该QEMU来运行编译好的Linux内核。这么做的原因如下：

- 在Windows下编译Linux源码十分麻烦，且QEMU在Windows下速度很慢；
- 之后的实验会涉及修改Linux源码。如果直接修改Ubuntu的内核，改完代码重新编译之后需要重启才能完成更改，但带GUI的Ubuntu系统启动速度较慢。另外，操作失误可能导致Ubuntu系统损坏无法运行。

3.2 下载、安装

为防止表述混乱，本文档指定 `~/oslab` 为本次实验使用的的目录。同学们也可以根据需要在其他目录中完成本次实验。如果在实验中遇到“找不到 `~/oslab` ”的报错，请先创建相关目录。

3.2.1 下载 Linux 内核源码

- 下载Linux内核源码，保存到 `~/oslab/` 目录（提示：使用 `wget` ）。源码地址：
 - <https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.9.263.tar.xz>
 - 备用链接1：<https://od.srpr.cc/acgg0/linux-4.9.263.tar.xz>
 - 备用链接2：<http://home.ustc.edu.cn/~maohaoyu/linux-4.9.263.tar.xz>
- 解压Linux内核源码（提示：使用 `tar` 。xz格式的解压参数是 `Jxvf` ，参数区分大小写）。

3.2.2 下载 busybox

- 下载busybox源码，保存到 `~/oslab/` 目录（提示：使用 `wget` ）。源码地址：
 - <https://busybox.net/downloads/busybox-1.32.1.tar.bz2>
 - 备用链接1：<https://od.srpr.cc/acgg0/busybox-1.32.1.tar.bz2>
 - 备用链接2：<http://home.ustc.edu.cn/~maohaoyu/busybox-1.32.1.tar.bz2>
- 解压busybox源码（提示：使用 `tar` 。bz2格式的解压参数是 `jxvf` ）。

如果3.2.1和3.2.2执行正确的话，你应该能在 `~/oslab/` 目录下看到两个子目录，一个叫 `linux-4.9.263` ，是Linux内核源码路径。另一个叫 `busybox-1.32.1` ，是busybox源码路径。

3.2.3 安装 qemu 和 Linux 编译依赖库

使用包管理器安装以下包：

- `qemu-system-x86` (Ubuntu 20.04以上) 或 `qemu` (Ubuntu 其他版本)
- `git`
- `build-essential` (里面包含了`make/gcc/g++`等，省去了单独安装的麻烦)
- `libelf-dev`
- `xz-utils`
- `libssl-dev`
- `bc`
- `libncurses5-dev`
- `libncursesw5-dev`

3.2.4 编译 Linux 源码

1. 精简配置：将我们提供的 `.config` 文件下载到Linux内核源码路径（提示：该路径请看3.2.2最后一段的描述，使用 `wget` 下载）。`.config` 文件地址：
 - <http://home.ustc.edu.cn/~maohaoyu/.config>
 - 备用链接：https://git.lug.ustc.edu.cn/gloomy/ustc_os/-/raw/master/term2021/lab1/.config

提示：以 `.` 开头的文件是隐藏文件。如果你下载后找不到它们，请参考2.2.2和2.3.3。

2. 内核配置：在Linux内核源码路径下运行 `make menuconfig` 以进行编译设置。本次实验直接选择 Save，然后Exit。
3. 编译：在Linux内核源码路径下运行 `make -j $((`nproc`-1))` 以编译内核。作为参考，助教台式机CPU为i5-7500(4核4线程)，使用VirtualBox虚拟机，编译用时5min左右。

提示：

1. `nproc` 是个shell内置变量，代表CPU核心数。如果虚拟机分配的cpu数只有1（如Hyper-V默认只分配1核），则需先调整虚拟机分配的核心数。
2. 如果你的指令只有 `make -j`，后面没有加上处理器核数，那么编译器会无限开多线程。因为Linux内核编译十分复杂，这会直接吃满你的系统资源，导致系统崩溃。
3. `nproc` 前的 ``` 是反引号，位于键盘左上侧。不是enter键旁边那个。

若干其他问题及其解决方案：

问题1: 编译内核时遇到 `make[1]: *** No rule to make target 'debian/canonical-certs.pem', needed by 'certs/x509_certificate_list'. Stop.`

解决方案：用文本编辑器(vim 或 gedit)打开 `PATH-TO-linux-4.9.263/.config`文件, 找到并注释掉包含`CONFIG_SYSTEM_TRUSTED_KEY`和 `CONFIG_MODULE_SIG_KEY`的两行即可。

解决方案链接：<https://unix.stackexchange.com/questions/293642/attempting-to-compile-kernel-yields-a-certification-error>

问题2: `make menuconfig`时遇到以下错误：`Your display is too small to run Menuconfig! It must be at least`

`19 lines by 80 columns.`

解决方案：请阅读报错信息并自行解决该问题。

4. 如果编译成功，我们可以在Linux内核源码路径下的 `arch/x86_64/boot/` 下看到一个 `bzImage` 文件，这个文件就是内核镜像文件。

若怀疑编译/安装有问题，可以先在Linux内核源码路径下运行 `make clean` 之后从3.2.4.1开始。

3.2.5 编译busybox

1. 在busybox的源码路径下运行 `make menuconfig` 以进行编译设置。修改配置如下：(空格键勾选)

```
Settings ->
Build Options
[*] Build static binary (no share libs)
```

2. 编译：在busybox的源码路径下运行 `make -j $((`nproc`-1))` 以编译busybox。本部分的提示与3.2.4.3相同。
3. 安装：在busybox的源码路径下运行 `sudo make install`。

若怀疑编译/安装有问题，可以先在busybox的源码路径下运行 `make clean` 之后从3.2.5.1开始。

3.2.6 制作根文件系统

1. 将工作目录切换为busybox源码目录下的 `_install` 目录。
2. 使用 `sudo` 创建一个名为 `dev` 的文件夹（提示：参考2.3.9）。
3. 使用以下指令创建 `dev/ram` 和 `dev/console` 两个设备文件：

- `sudo mknod dev/console c 5 1`
- `sudo mknod dev/ram b 1 0`

4. 使用 `sudo` 创建一个名为 `init` 的文件（提示：参考2.3.16）。使用文本编辑器编辑文件，文件内容如下：

特别提示：这一步 **不是** 让你在终端里执行这些命令。在复制粘贴时，请注意空格、回车是否正确地保留。

在命令行下启动gedit会在终端里报warning，忽略即可。 [参考链接](#)

```
#!/bin/sh
echo "INIT SCRIPT"
mkdir /proc
mkdir /sys
mount -t proc none /proc
mount -t sysfs none /sys
mkdir /tmp
mount -t tmpfs none /tmp
echo -e "\nThis boot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
exec /bin/sh
```

5. 赋予 `init` 执行权限：`sudo chmod +x init`
6. 将x86-busybox下面的内容打包归档成cpio文件，以供Linux内核做initramfs启动执行：

```
find . -print0 | cpio --null -ov --format=newc | gzip -9 > 你们自己指定的cpio文件路径
```

注意1：该命令一定要在busybox的 `_install` 目录下执行。

注意2：每次修改 `_install`，都要重新执行该命令。

注意3：请自行指定cpio文件名及其路径。示例：`~/oslab/initramfs-busybox-x64.cpio.gz`

3.2.7 运行qemu

我们本次实验需要用到的qemu指令格式：

```
qemu-system-x86_64 [-s] [-S] [-kernel ImagePath] [-initrd InitPath] [--append Para] [-nographic]
```

参数	含义
-s	在3.3.2介绍
-S	在3.3.2介绍
-kernel ImagePath	指定系统镜像的路径为ImagePath。在本次实验中，该路径为3.2.4.4所述的bzImage文件路径。
-initrd InitPath	指定initramfs(3.2.6生成的cpio文件)的路径为InitPath。在本次实验中，该路径为3.2.6.6所述的cpio文件路径。
--append Para	指定给内核启动赋的参数。
-nographic	设置无图形化界面启动。

在 `--append` 参数中，本次实验需要使用以下子参数：

参数	含义
nokaslr	关闭内核地址随机化，便于调试。
root=/dev/ram	指定启动的第一个文件系统为ramdisk。我们在3.2.6.3创建了 <code>/dev/ram</code> 这个设备文件。
init=/init	指定init进程为/init。我们在3.2.6.4创建了init。
console=ttyS0	对于无图形化界面启动 （如WSL），需要使用本参数指定控制台输出位置。

这些参数之间以空格分隔。

总结(TL;DR):

运行下述指令前，请注意先**理解指令含义，注意换行问题**。

如果报错找不到文件，请先检查：**是否把pdf的换行和不必要的空格复制进去了？前几步输出的东西是不是在期望的位置上？** 我们不会提示下述指令里哪些空格/换行符是应有的，哪些是不应有的。请自行对着上面的指令含义排查。

如果报错Failed to execute /init (error -8)，建议从3.2.5开始重做。若多次重做仍有问题，建议直接删掉解压好的busybox和Linux源码目录，从3.2.4重做。目前已知init文件配置错误、Linux内核编译出现问题都有可能致本错误。

- 以图形界面，弹出窗口形式运行内核：

```
qemu-system-x86_64 -kernel ~/oslab/linux-4.9.263/arch/x86_64/boot/bzImage -initrd
~/oslab/initramfs-busybox-x64.cpio.gz --append "nokaslr root=/dev/ram init=/init"
```

- Ubuntu 20.04/20.10 环境下如果出现问题，可执行以下指令：

```
qemu-system-x86_64 -kernel ~/oslab/linux-4.9.263/arch/x86_64/boot/bzImage -initrd
~/oslab/initramfs-busybox-x64.cpio.gz --append "nokaslr root=/dev/ram init=/init"
```

- 如不希望qemu以图形界面启动，希望以无界面形式启动（如WSL），输出重定向到当前shell，使用以下命令：

```
qemu-system-x86_64 -kernel ~/oslab/linux-4.9.263/arch/x86_64/boot/bzImage -initrd
~/oslab/initramfs-busybox-x64.cpio.gz --append "nokaslr root=/dev/ram init=/init
console=ttyS0 " -nographic
```

其他常见问题：

1. 如何检查运行是否正确？

弹出的黑色窗口就是qemu。因为系统内核在启动的时候会输出一些log，所以qemu界面里偶尔会蹦出一两条log是正常的，**只要这些log不是诸如"kernel panic"之类的报错**即可。建议尝试输入一条命令，比如在弹出的窗口里面输入 `ls` 回车，如果能够显示相关的文件列表,即说明运行正确。

2. 鼠标不见了，该怎么办？

请观察窗口上方的标题栏，"Press (请自行观察标题栏上的说明) to release grab"

3. 如何关闭qemu？

对于图形化界面，直接点击右上角的叉就行了。对于非图形化界面，请先按下Ctrl+A，松开这两个键之后再按X。



```
QEMU - Press [redacted] to release grab
Machine View
[ 1.647979] Magic number: 14:432:523
[ 1.648788] console [netcon0] enabled
[ 1.648867] netconsole: network logging started
[ 1.653986] ALSA device list:
[ 1.654053] No soundcards found.
[ 1.726126] Freeing unused kernel memory: 1184K
[ 1.726311] Write protecting the kernel read-only data: 14336k
[ 1.767307] Freeing unused kernel memory: 708K
[ 1.779569] Freeing unused kernel memory: 1316K
INIT SCRIPT

This boot took 1.88 seconds

/bin/sh: can't access tty: job control turned off
/ # [ 1.953757] tsc: Refined TSC clocksource calibration: 3408.358 MHz
[ 1.954164] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x31212603
e88, max_idle_ns: 440795247449 ns
[ 2.231628] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042
/serio1/input/input3
[ 3.034533] clocksource: Switched to clocksource tsc

ls
bin      init     proc     sbin     tmp
dev      linuxrc root     sys      usr
/ # [ 17.050770] random: crng init done
```

一次成功的ls，说明启动成功

偶尔会蹦出几条log，这是正常的

3.3 使用 gdb调试内核

`gdb`是一款命令行下常用的调试工具，可以用来打断点、显示变量的内存地址，以及内存地址中的数据等。使用方法是 `gdb 可执行文件名`，即可在gdb下调试某二进制文件。

一般在使用gcc等编译器编译程序的时候，编译器不会把调试信息放进可执行文件里，进而导致gdb知道某段内存里有内容，但并不知道这些内容是变量a还是变量b。或者，gdb知道运行了若干机器指令，但不知道这些机器指令对应哪些C语言代码。所以，在使用gdb时需要在编译时加入 `-g` 选项，如：

`gcc -g -o test test.c` 来将调试信息加入可执行文件。而Linux内核采取了另一种方式：它把符号表独立成了另一个文件，在调试的时候载入符号表文件就可以达到相同的效果。

- `gdb`里的常用命令

```
r/run # 开始执行程序
b/break <location> # 在location处添加断点，location可以是代码行数或函数名
b/break <location> if <condition> # 在location处添加断点，仅当condition条件满足才中断运行
c/continue # 继续执行到下一个断点或程序结束
n/next # 运行下一行代码，如果遇到函数调用直接跳到调用结束
s/step # 运行下一行代码，如果遇到函数调用则进入函数内部逐行执行
ni/nexti # 类似next，运行下一行汇编代码（一行c代码可能对应多行汇编代码）
si/stepi # 类似step，运行下一行汇编代码
list # 显示当前行代码
p/print <expression> # 查看表达式expression的值
q # 退出gdb
```

3.3.1 安装 gdb

使用包管理器安装名为gdb的包即可。

3.3.2 启动 gdb server

使用3.2.7所述指令运行qemu。但需要加上 `-s` 和 `-S` 两个参数。这两个参数的含义如下：

参数	含义
<code>-s</code>	启动gdb server调试内核，server端口是1234。若不想使用1234端口，则可以使用 <code>-gdb tcp:xxxx</code> 来取代此选项。
<code>-S</code>	若使用本参数，在qemu刚运行时，CPU是停止的。你需要在gdb里面使用c来手动开始运行内核。

3.3.3 建立连接

另开一个终端，运行gdb，然后在gdb界面里运行如下命令：

```
target remote:1234 # 建立gdb与gdb server间的连接。这时候我们看到输出带有??，还报了一条warning
# 这是因为没有加载符号表，gdb不知道运行的是什么代码。
c # 手动开始运行内核。执行完这句后，你应该能发现旁边的qemu开始运行了。
q # 退出gdb。
```

3.3.4 重新编译Linux内核使其携带调试信息

刚才因为没有加载符号表，所以gdb不知道运行了什么代码。所以我们要重新编译Linux来使其携带调试信息。

1. 进入Linux源码路径。
2. 执行下列语句(这条语句没有回车，注意文档显示时产生的额外换行问题)：

```
./scripts/config -e DEBUG_INFO -e GDB_SCRIPTS -d DEBUG_INFO_REDUCED -d DEBUG_INFO_SPLIT -
d DEBUG_INFO_DWARF4
```

3. 重新编译内核。 `make -j $((`nproc`-1))`

作为参考，助教台式机CPU为i5-7500(4核4线程)，使用VirtualBox虚拟机，编译用时8.5min左右。

3.3.5 加载符号表、设置断点

1. 重新执行3.3.2.
2. 另开一个终端，运行gdb，然后在gdb界面里运行如下命令：

```
target remote:1234 # 建立gdb与gdb server间的连接。这时候我们看到输出带有??，
# 这是因为没有加载符号表，gdb不知道运行的是什么代码。
file Linux源码路径/vmlinux # 加载符号表。（不要把左边的东西原封不动地复制过来）
n # 单步运行。此时可以看到右边不是??，而是具体的函数名了。
break start_kernel # 设置断点在start_kernel函数。
c # 运行到断点。
l # 查看断点代码。
p init_task # 查看断点处名为"init_task"的变量值。这个变量是个结构体，里面有一
大堆成员。
```

```
ustc@ustc-VirtualBox: ~/oslab/busybox-1.32.1/_install
info "(gdb)Auto-loading safe path"
(gdb) n
Single stepping until exit from function nmi_print_seq,
which has no line number information.
0x00000000000000e05b in cpu_hw_events ()
(gdb) break start_kernel
Breakpoint 1 at 0xffffffff81f55a04: file init/main.c, line 486.
(gdb) c
Continuing.

Breakpoint 1, start_kernel () at init/main.c:486
486     set_task_stack_end_magic(&init_task);
(gdb) l
481     asmlinkage __visible void __init start_kernel(void)
482     {
483         char *command_line;
484         char *after_dashes;
485
486         set_task_stack_end_magic(&init_task);
487         smp_setup_processor_id();
488         debug_objects_early_init();
489
490         /*
(gdb) p init_task
$1 = {thread_info = {flags = 0, status = 0}, state = 0,
      stack = 0xffffffff81e00000 <init_thread_union>, usage = {counter = 2},
      flags = 2097152, ptrace = 0, wake_entry = {next = 0x0 <irq_stack_union>},
      on_cpu = 0, cpu = 0, wakee_flips = 0, wakee_flip_decay_ts = 0,
      last_wakee = 0x0 <irq_stack_union>, wake_cpu = 0, on_rq = 0, prio = 120,
      static_prio = 120, normal_prio = 120, rt_priority = 0,
      sched_class = 0x0 <irq_stack_union>, se = {load = {weight = 0,
        inv_weight = 0}, run_node = {__rb_parent_color = 0,
          rb_right = 0x0 <irq_stack_union>, rb_left = 0x0 <irq_stack_union>},
        group_node = {next = 0xffffffff81e0f568 <init_task+168>,
          prev = 0xffffffff81e0f568 <init_task+168>}, on_rq = 0, exec_start = 0,
        sum_exec_runtime = 0, vruntime = 0, prev_sum_exec_runtime = 0,
        nr_migrations = 0, statistics = {wait_start = 0, wait_max = 0,
```

单步运行

设置断点

运行到断点

查看断点处代码

查看变量

第四部分：检查要求

本次实验无实验报告。

本次实验共10分。

4.1 第一部分检查要求

检查学生是否安装了Linux系统（不局限于Ubuntu）。本部分不计分。

4.2 第二部分检查要求

1. 我们现场随便给出一条本实验文档未涉及的指令，你需要自己使用 `man` 指令阅读其手册简要解释该指令的含义，并简要介绍任意一个参数的含义。本部分共1分。若遇到不认识的英文单词，可以现场搜索。
2. 在助教的电脑上或机房的电脑上，在助教的监督下，使用实验提供的测试程序进行现场测试，每人最多尝试2次，取最高分。**答题时不得打小抄、查阅实验文档。**从题库中抽4题，每题抽4个选项，选项顺序会随机打乱。答题总时间120秒。每题1分。满分4分。为防止同一学生在不同助教处刷次数，每一次尝试都记录成绩。

题库见提供的csv文件，三个csv都是题库。csv文件里，第一列是题目，第二列是正确答案，其余是干扰项。公布的题库、自测程序和考察时使用的完全相同。csv文件是UTF-8 with BOM格式，可以直接用Excel打开。但Excel可能会将部分选项解释异常，因此建议使用文本编辑器打开。自测程序由Python语言编写，在Linux下，在自测程序目录下执行 `python3 test.py` 即可运行测试程序。Ubuntu自带Python3。若未安装Python3，可用apt自行安装。强烈建议大家检查之前自测几次，熟悉程序的工作流程。

为防止替考及背完就忘的情况，在后续实验中，如果发现有同学忘了指令含义，我们可能会考虑让其重做一遍测试题，并按照错误个数扣实验分。

因为往年很多同学实验做到最后还是对Linux指令很不熟悉，所以只能出此下策强迫大家进行记忆。

为保证本评测程序在多种操作系统下均可运行，秒数刷新时会覆盖掉已输入的数字。但这些数字确实是已经输入了的，只是被后续读秒的输出覆盖而不可见。所以，若确信输入正确，直接回车即可。若怀疑输入错误，请长按 `Backspace` 保证之前输入的全被删掉之后再输入新的。

4.3 第三部分检查要求

现场检查能否启动虚拟机并启动gdb调试，即现场执行3.3.5。本部分共5分。本部分检查中，允许学生对照实验文档操作。

- 你需要能够简要解释符号表的作用。若不能解释，本部分减1分。
- 此外，**为提高检查效率，请自己编写一个【shell脚本】启动qemu（即，使用2.4所述方法完成3.3.5的第一步）**。若不能使用脚本启动qemu，本部分减1分。你不需要使用脚本启动或操作gdb。

