Linux操作系统实验指导书

机房环境

使用机房的计算机进行实验，开机选择win10_3系统，打开VMware后选择Ubuntu 18.04 Server虚拟机。开机后等待显示登录界面，登录信息如下

用户名：bjut

密码：admin

root密码：root

实验一 添加一个新的系统调用

一、实验目的

理解操作系统内核与应用程序的接口关系；加深对内核空间和用户空间的理解；学会增加新的系统调用。

二、实验内容与要求

首先增加一个系统调用函数，然后连接新的系统调用，重建新的Linux内核（版本4.20.13），用新的内核启动系统，并使用新的系统调用。

三、实验步骤

输入如下命令查看当前内核版本：

```
cat /proc/version
或者
uname -a
```

1.拷贝并解压缩内核源代码

本次实验将要编译的内核版本为4.20.13，所有实验操作均在root用户下进行，输入`sudo su`和普通用户密码切换到root，也可以自行设置root密码。机房环境已经设置好root密码，可直接通过输入`su`命令和root密码切换权限。

在控制台输入如以下命令拷贝并解压缩源代码，注意：如果使用WinSCP传输文件，由于该软件支持的文件传输协议（SFTP，FTP，SCP）均不支持在会话中切换到root用户，而Linux默认不允许root用户直接登录，因此无法将源代码压缩包文件直接拷贝到目标地址，请拷贝到其他普通权限的目录（如/home/usrname），然后同样按照下方命令操作。

```
cp -ri 共享文件夹路径/linux-4.20.13.tar.xz /usr/src # 将源码压缩包拷贝到目标地址
cd /usr/src # 可以用ls命令查看源码文件是否成功拷贝到该目录下
xz -d linux-4.20.13.tar.xz # 解压缩
tar xvf linux-4.20.13.tar  # 解包
cd /usr/src/linux-4.20.13  # 切换到源码所在路径
```

2.添加系统调用

修改源码文件夹中的三个文件 sys.c 、syscalls.h 和 syscall_64.tbl

（1）修改 sys.c 文件

在虚拟机控制台用vim打开并编辑文件。使用WinSCP拷贝文件或使用VS Code编辑文件的情况，同样需要注意上文提到的权限问题。在修改前建议对原文件做备份。

```
cd /usr/src/linux-4.20.13/kernel
cp sys.c sys.c.bak # 备份原文件 该命令仅在此处执行一次，防止重复拷贝导致备份被覆盖
vim sys.c
```

在首行加入 linkage.h 的头文件声明，然后在文件末尾加入用户定义的系统调用函数

```
#include <linux/linkage.h>
//其他代码
//···
//···
//文件末尾
SYSCALL_DEFINE1(mycall, int, number)
{
    printk("This is my first system call %d ", number);
    return number;
}
```

> 注：进入vim编辑器界面，按 **G** 可跳至文件末尾；**PageDown/PageUp** 快速滚动 ； 按 **i** 进入编辑状态；**ESC** 退出编辑状态；
>
> 保存并退出：按**ESC**退出编辑状态进入命令状态后，输入 **:wq**
>
> 只退出不保存：按**ESC**进入命令状态后，输入 **:q!**

（2）修改 syscalls.h 文件

```
cd /usr/src/linux-4.20.13/arch/x86/include/asm/
cp syscalls.h syscalls.h.bak # 备份原文件
vim syscalls.h
```

在文件最末尾另起一行键入以下内容

```
asmlinkage long sys_mycall(int);
```

（3）编辑 syscall_64.tbl 文件，添加系统调用号

```
cd /usr/src/linux-4.20.13/arch/x86/entry/syscalls
cp syscall_64.tbl syscall_64.tbl.bak # 备份原文件
vim syscall_64.tbl
```

在文件中添加自己的系统调用号，假设原来common的系统调用号到334号，我们设置自己的系统调用号为335。在334号后面一行添加如下代码

```
335    64    mycall                __x64_sys_mycall
# 格式为：335[TAB]64[TAB]mycall[TAB][TAB][TAB]__x64_sys_mycall
```

（4）若要将修改后的文件恢复原样，在文件相应路径下输入以下命令用原文件将修改后的文件覆盖

```
cp 原文件名.bak 原文件名
```

3.编译内核

输入如下命令编译内核，若编译过程中出错，请仔细检查每段命令是否输入正确，然后**从第一个命令开始重新执行一遍**，若仍然报相同错误，请检查第2步中修改文件时是否有拼写或空格错误

```
cd /usr/src/linux-4.20.13
make mrproper    # 清理源代码目录中以前构建内核时遗留的文件和.config文件
make clean       # 清理遗留的目标文件和可执行文件
make menuconfig # 出现图形界面后，用方向键控制光标选择保存，确认，退出
                 # 若此步骤报错并提示 Your display is too small to run Menuconfig! 请调大
                 # 系统设置中的分辨率，然后重新执行命令

sed -ri '/CONFIG_SYSTEM_TRUSTED_KEYS/s/=.+/=""/g' .config  # 修改.config文件 注意拼写和空格
make -j4         # 耗时最长，不少于60分钟，根据自己计算机的配置启用多线程-jN
```

内核编译的时间较长且不能中断，请做好安排预留足够的时间完成编译，编译过程会占用较多CPU和内存资源，尽量不要运行其他资源密集型程序。根据每个人硬件设备和虚拟机设置的不同，编译时间从1小时~2小时不等，若发现编译结束时间过早（如十几分钟就回到命令行模式），一般是编译过程出错，请注意控制台是否输出 Error; fatal; failed; 等信息，或 no space left on device 提示硬盘空间不足。仔细检查每一步骤是否执行正确，可以通过`history`命令检查控制台命令历史记录，尤其注意空格、大小写、单词拼写等容易出错的部分。

4.安装内核

输入以下命令安装内核

```
make modules_install
make install
```

执行完毕后，重启虚拟机，自动进入新的内核。输入以下命令查看内核版本。对于原内核版本高于4.20.13的情况，请按照附录4的方法手动选择启动内核。

```
cat /proc/version
或者
uname -a
```

5.测试新系统调用是否成功

（1）编写测试程序

```c
/* testcall.c */

#include<stdio.h>
#include<linux/kernel.h>
#include<unistd.h>
#include<sys/syscall.h>

int main()
{
    long a;
    a = syscall(335, 666); //调用第335号系统调用，即sys_mycall();
    printf("The number from syscall is %ld\n",a);
    return 0;
}
```

编译程序

```
gcc testcall.c -o testcall # 编译
./testcall # 运行
```

若控制台输出 The number from syscall is 666 表明系统调用添加成功。

实验二 内核模块的添加和卸载

一、实验目的

了解模块的编程方法

二、实验内容与要求

编写HelloWorld模块，实现编译、加载和卸载模块

三、实验步骤

（1）编写模块程序

```c
/* HelloWorld.c */

#include <linux/module.h>
#include <linux/init.h>

/* 模块加载时运行的代码，可自定义函数名 */
static int hellomodule_init(void)
{
    printk("HelloWorld!\n");
    return 0;
}

/* 模块卸载时运行的代码 */
static void hellomodule_exit(void)
{
    printk("Goodbye!\n");
}

/* 模块加载宏和卸载宏，括号里填写上面两个函数名 */
module_init(hellomodule_init);
module_exit(hellomodule_exit);

/* 模块遵循的公共许可证 */
MODULE_LICENSE("GPL");
```

（2）编译内核模块

先编写Makefile文件。在与HelloWorld.c相同的目录下，新建文件名为Makefile的文件编写以下内容。

```
obj-m := HelloWorld.o
all:
        make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
clean:
        make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

> 注：Makefile的内容由一组规则组成，每个规则都具有以下形式：
>
> > target: requirements
> > 	target build instructions
>
> 其中 target build instructions 前**必须使用TAB缩进**，否则会出现 Makefile:2: *** missing separator. Stop. 等错误信息。
>
> obj-m 的目的是让内核中的 kbuild 系统通过 mod_name.c 文件编译出 mod_name.o，连接这些文件后得到内核模块 mod_name.ko。因此请确保你的 obj-m := 后面的模块名称与你的.c源代码文件相同。

编译内核模块。在同目录下执行命令`make`，成功执行后，会在当前目录下生成许多新文件：

> HelloWorld.o HelloWorld.ko HelloWorld.mod.o HelloWorld.mod.c Modules.symvers

其中 HelloWorld.ko 就是我们要加载的模块。

（3）加载模块

在同目录下输入命令`insmod ./HelloWorld.ko`加载模块，然后输入`lsmod`查看能否找到名为HelloWorld的模块。如果输出内容超出屏幕显示范围，可以用`shift + PgUp/PgDown`键翻页，或者输入`lsmod | grep 'HelloWorld\|Module'`查看是否能搜索到该模块。

```
root@linux_lab:/home/bjut/exp2# insmod ./HelloWorld.ko
root@linux_lab:/home/bjut/exp2# lsmod
Module                  Size  Used by
HelloWorld             16384  0
binfmt_misc            24576  1
snd_intel8x0           45056  0
snd_ac97_codec        135168  1 snd_intel8x0
ac97_bus               16384  1 snd_ac97_codec
snd_pcm               102400  2 snd_intel8x0,snd_ac97_codec
snd_timer              36864  1 snd_pcm
joydev                 24576  0
```

在同目录下输入命令`dmesg`查看最后一行，会出现模块加载时调用的函数输出。

（4）卸载模块

```
rmmod HelloWorld # 卸载模块
lsmod # 查看模块列表
dmesg # 查看模块卸载时调用的函数输出
```

```
[   18.013747] NET: Registered protocol family 40
[   18.810232] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[   18.818991] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[   18.819795] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[   20.565617] new mount options do not match the existing superblock, will be ignored
[  187.237495] HelloWorld: loading out-of-tree module taints kernel.
[  187.237545] HelloWorld: module verification failed: signature and/or required key missing - tainting kernel
[  187.237962] HelloWorld!
[  194.019917] GoodBye!
```

可以用`make clean`命令清除编译生成的文件， 以便重新编译。

实验三 编写系统时钟模块

一、实验目的

理解/proc文件系统的作用、原理和机制

二、实验内容

编写一个时钟模块读取系统当前时间，并在/proc目录下创建用于交换数据的虚拟文件，使用户可以通过读取该文件获得模块在内核模式读取到的时间数据

三、实验步骤

1.编写并加载clock模块

新建clock文件夹，并在新文件夹路径下编写clock.c源程序

```
/* clock.c */

#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>

/* prototypes */
static int proc_clock_show(struct seq_file *m, void *v);
static int proc_clock_open(struct inode *inode, struct file *file);
static int init_clock(void);
static void cleanup_clock(void);

/* 定义模块信息 */
#define USER_MODULE_VERSION "1.0"
#define USER_ROOT_DIR "clock"
#define MODULE_NAME "my_clock"

/*自定义/proc父目录*/
struct proc_dir_entry *my_clock_dir;
struct proc_dir_entry *my_clock_file;

/*定义一个简单的fops结构体，除open方法以外直接使用seq_file接口已实现的方法*/
static const struct file_operations my_clock_fops =
{
    .owner = THIS_MODULE,
    .open = proc_clock_open,
    .read = seq_read,
    .llseek = seq_lseek,
    .release = single_release,
};

/*输出读取的当前系统时间*/
static int proc_clock_show(struct seq_file *m, void *v)
```

```
{
    struct timeval tv;
    do_gettimeofday(&tv);
    seq_printf(m, "%ld %ld\n", tv.tv_sec, tv.tv_usec);
    return 0;
}

/*open方法通过*show方法一次性输出虚拟文件中的内容*/
static int proc_clock_open(struct inode *inode, struct file *file)
{
    return single_open(file, proc_clock_show, NULL);
}

/*加载模块时创建/proc下的虚拟文件*/
static int __init init_clock(void)
{
    printk("clock: init_module()\n");
    my_clock_dir = proc_mkdir(USER_ROOT_DIR, NULL);
    my_clock_file = proc_create(MODULE_NAME, 0, my_clock_dir, &my_clock_fops);
    printk(KERN_INFO"%s %s has initialized.\n", MODULE_NAME, USER_MODULE_VERSION);
    return 0;
}

/*卸载模块时将proc entry移除*/
static void __exit cleanup_clock(void)
{
    printk("clock: cleanup_module()\n");
    remove_proc_entry(MODULE_NAME, my_clock_dir);
    remove_proc_entry(USER_ROOT_DIR, NULL);
    printk(KERN_INFO"%s %s has removed.\n", MODULE_NAME, USER_MODULE_VERSION);
}

module_init(init_clock);
module_exit(cleanup_clock);

MODULE_DESCRIPTION("clock module for gettimeofday of proc.");
MODULE_LICENSE("GPL");
```

按照与实验二步骤（2）（3）相同的方法编译、加载模块。加载clock模块成功后，进入`/proc`目录，会发现存在一个`clock`文件夹

（2）编写测试函数testclock.c

```
/* testclock.c */

#include<sys/time.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
```

```c

int main()
{
    struct timeval tv;
    FILE *fp;
    char info[128];
    fp=fopen("/proc/clock/my_clock","r");
    if(fp==NULL)
    {
        printf("clock module doesn't exist.\n");
        exit(0);
    }
    fgets(info, 30, fp);
    printf("my_clock time:\n%s\n", info);
    gettimeofday(&tv, 0);
    printf("system time:\n%ld %ld\n", tv.tv_sec, tv.tv_usec);
    fclose(fp);
}
```

编译testclock.c测试程序，运行并查看测试结果。也可以输入命令`cat /proc/clock/my_clock`查看控制台的输出内容。

（3）卸载模块

模块卸载方法同实验二步骤（4），卸载模块后，再查看`/proc`目录，将找不到clock模块

附录1 clock.c程序分析

在 Linux 中输入命令`man proc`可以查看其官方定义：process information pseudo-filesystem，也就是说 /proc 文件系统是一个伪文件系统，目的是为内核数据结构提供交换数据的接口。之所以将 /proc 称为伪文件系统，是因为 /proc 下的文件和正常的文件不同，不会占用任何的硬盘空间。内核通过实现所有的文件操作符创建一个虚拟的文件，而文件内容则由内核模块动态生成。由于内核模块运行在高权限和内核地址空间中，因此可以访问所有的内核数据结构，也就是实际上能够访问整个系统。/proc 文件系统通过这种方式提供了一个在用户空间和内核空间之间的接口，可以将它看作是内核的控制和信息中心。

实际上，很多 Linux 系统工具都是通过简单地调用此目录下的文件实现的。

例如实验一中查看内核版本使用的命令`cat /proc/version`；实验二中查看模块列表的命令`lsmod`等同于`cat /proc/modules`；如果我们通过`ls`命令查看文件，则会发现这些文件的大小为0：

```
bjut@linux_lab:~$ ls -l /proc/version
-r--r--r-- 1 root root 0 Jan 11 13:07 /proc/version
```

如果想了解 /proc 中其他目录的内容和功能，可以通过`man proc`提供的官方手册查看。

1.模块程序

模块加载时调用`proc_create`函数

```
static inline struct proc_dir_entry *proc_create(
const char *name,
umode_t mode,
struct proc_dir_entry *parent,
const struct file_operations *proc_fops)
{return proc_create_data(name, mode, parent, proc_fops, NULL);}
/*
name：模块名
mod：模块模式
parent：父entry，当设为NULL时，默认为/proc
proc_fops：操作函数表
*/
```

该函数会创建一个 PROC entry，用户可以通过此文件和内核进行数据交换。详细内容请查看`/usr/src/linux-内核版本号/include/linux/proc_fs.h`文件。

2.时间数据结构

```
struct timespec{
    time_t tv_sec; //秒
    long tv_nsec; //纳秒
}
struct timeval{
    time_t tv_sec; //秒
    long tv_usec; //微秒
}
```

Linux在源码`/include/linux/time.h`中定义了关于时间的数据结构，其中定义了一个全局系统变量`struct timeval xtime`用来保存当前时间；`proc_read_clock`函数的第二个参数指向了`/proc/clock/my_clock`文件的缓冲区，因此调用`printf`后，系统时间`xtime`的值被格式化输出到my_clock文件中。

附录2 实验报告内容

实验报告请按照如下结构编写：

实验目的
功能要求
主要功能设计说明
主程序函数与参数说明
程序框图
程序设计实现说明
测试结果与说明

附录3 如何将主机中的文件传入Linux虚拟机

在实验过程中需要将新的内核代码文件传入Linux虚拟机中，在此提供两种方法:

1. 在虚拟机管理器中设置共享文件夹并挂载到Linux系统中
2. 使用WinSCP软件连接虚拟机

1.1 设置共享文件夹

（1）以VMware为例，在虚拟机开启时选择菜单栏 虚拟机=>设置=>选项=>共享文件夹，开启文件夹共享，并选择想要共享的主机文件夹路径和共享文件夹名称（在Linux中显示的文件夹名，建议设置为全英文）。关闭设置，切换到虚拟机，输入命令`cd /mnt/hgfs/共享文件夹名`进入共享文件夹。

（2）以VirtualBox为例，首先需要在Linux中安装增强功能。Desktop版可以直接通过菜单 设备->安装增强功能 进行安装，Server版请按以下步骤操作:

```
apt-get install dkms # 安装依赖
reboot # 重启虚拟机
===
mount /dev/cdrom /mnt/ # 挂载光盘镜像
```

若控制台输出`mount: [path] is write-protected,  mounting read-only`表示挂载成功，若挂载失败，请检查菜单 设备->分配光驱 中是否有`VBoxGuestAdditions.iso`镜像文件，没有请在`C:\Program Files\Oracle\VirtualBox`或者你自定义的安装路径下寻找。

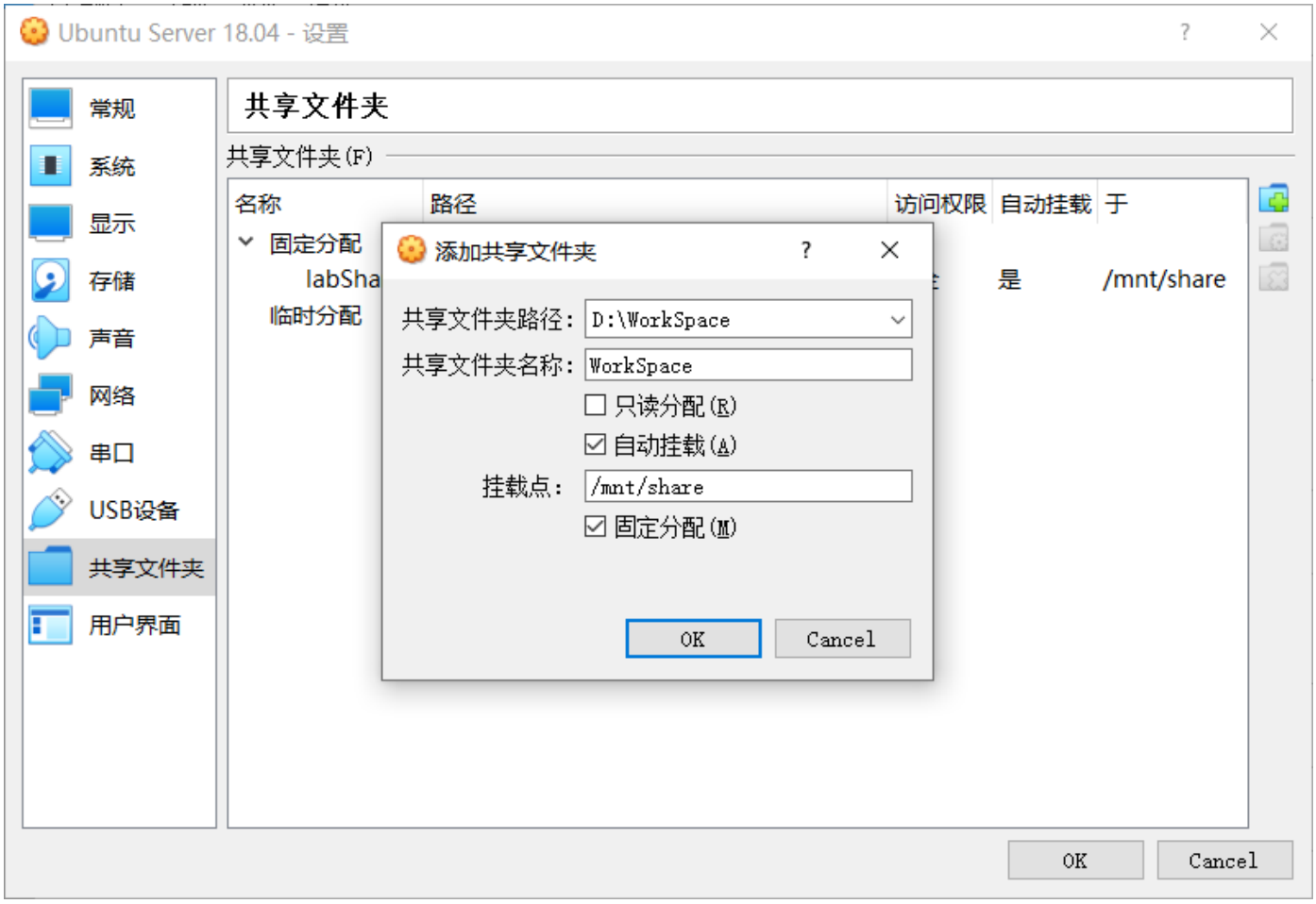
挂载成功后，执行如下命令

```
/mnt/VBoxLinuxAdditions.run # 执行安装命令
umount /mnt/ # 安装完成后卸载光盘
```

在虚拟机中创建共享目录，之后通过此目录访问主机中的共享文件夹(需要root权限)，这里以 /share 为例

```
mkdir /mnt/share
```

在虚拟机设置或设备菜单中找到共享文件夹，选择添加共享文件夹，选择要共享的文件夹路径，勾选[√]自动挂载和[√]固定分配，填写Linux系统中的挂载点，本文设置为 /mnt/share



启动虚拟机，访问共享文件夹`cd /mnt/share`然后输入`ls`命令查看文件夹中的内容

1.2 通过WinSCP连接到虚拟机

（1）检查连接环境

安装ssh服务

```
apt-get install ssh
```

查看服务是否启动

```
ps -elf | grep ssh
```

查看防火墙状态

```
ufw status
```

若显示active，需要输入以下命令关闭防火墙

```
ufw disable
```

（2）检查虚拟机是否分配了独立的内网IP

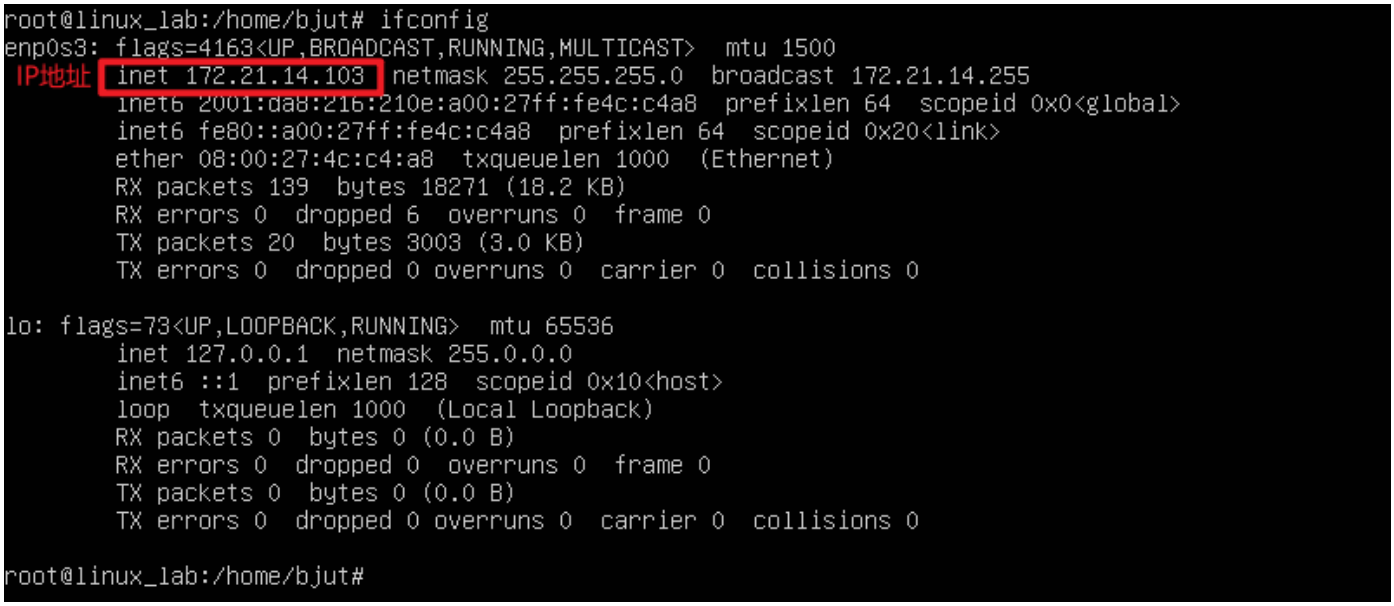
登录虚拟机后会出现如下系统信息，或使用`ifconfig`命令查看网络设置。若 IP address/inet 后面不是192或172开头的内网ipv4地址，请按照步骤（3）进行设置。

```
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-208-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Day Mon  d hour:min:sec UTC year

  System load:  0.0                Processes:           123
  Usage of /:   15.1% of 37.11GB   Users logged in:     1
  Memory usage: 5%                 IP address for ens33: 192.168.25.130
  Swap usage:   0%
```

（3）修改虚拟机网络设置

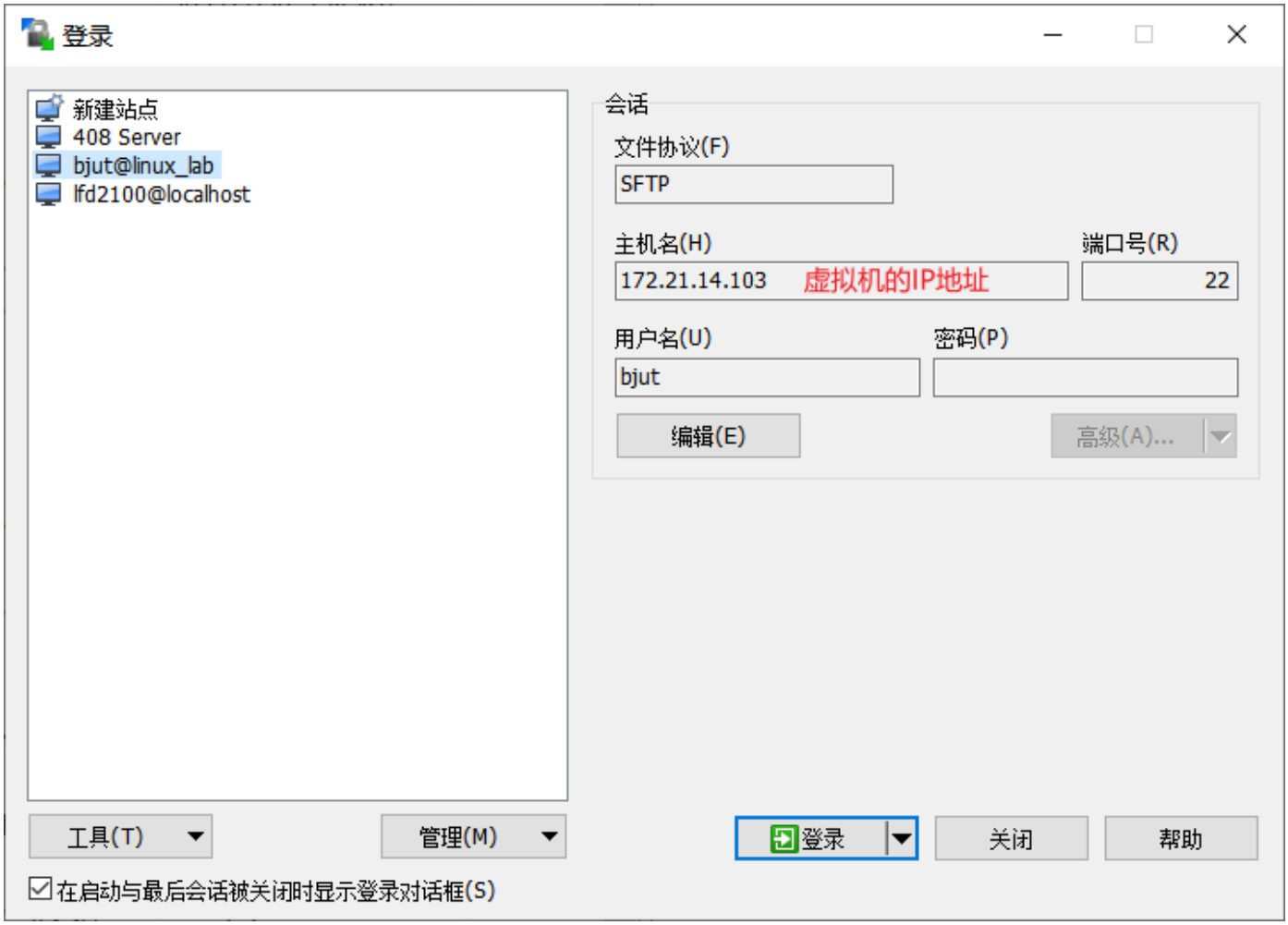
打开虚拟机设置，找到网络/网络适配器选项，将NAT模式改为仅主机模式。注意修改为仅主机模式后虚拟机可能无法连接互联网，请在完成安装相关编译支持库后再进行此小节的操作。

重启虚拟机，输入`ifconfig`查看IP分配信息



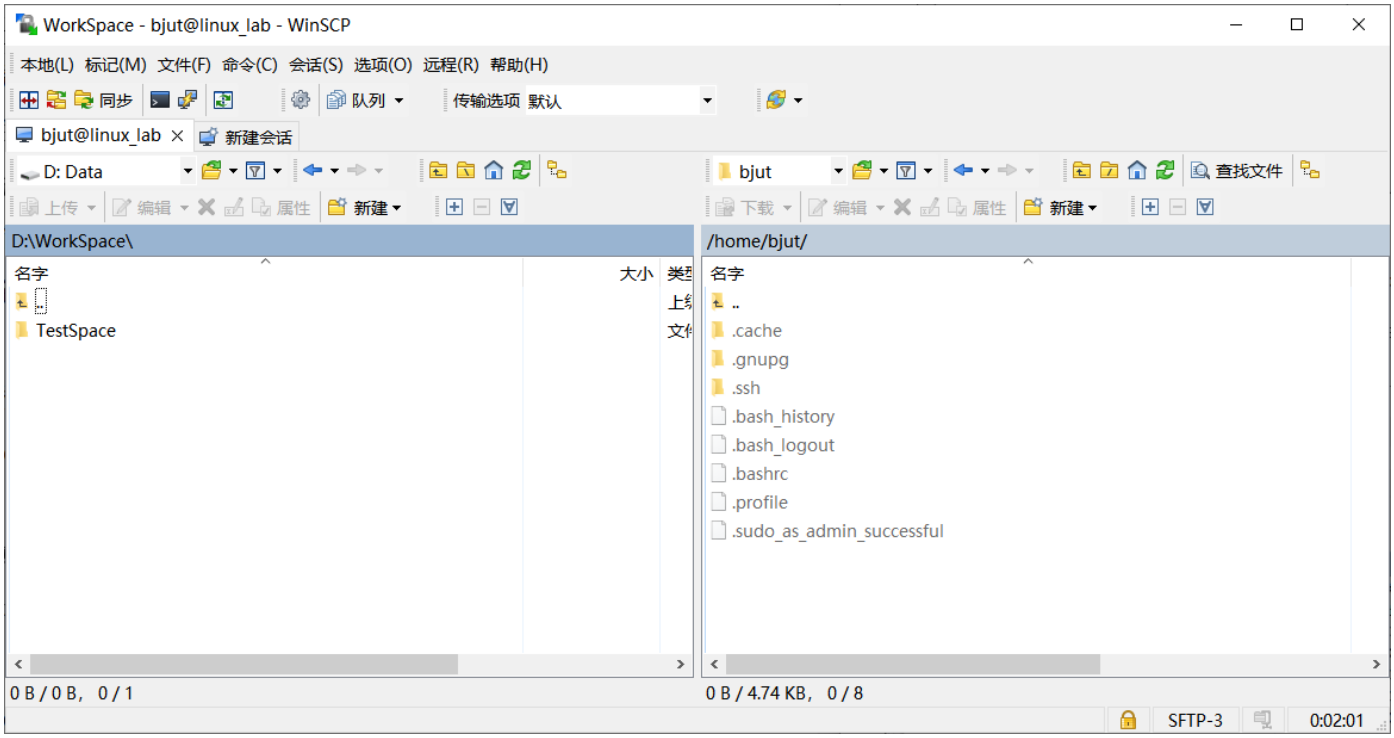
在WinSCP官网下载软件:https://winscp.net/eng/download.php

打开软件，弹出登录窗口，左侧选择新建站点，右侧填写虚拟机的IP地址，端口号默认22，用户名密码即Linux虚拟机的用户名和密码.

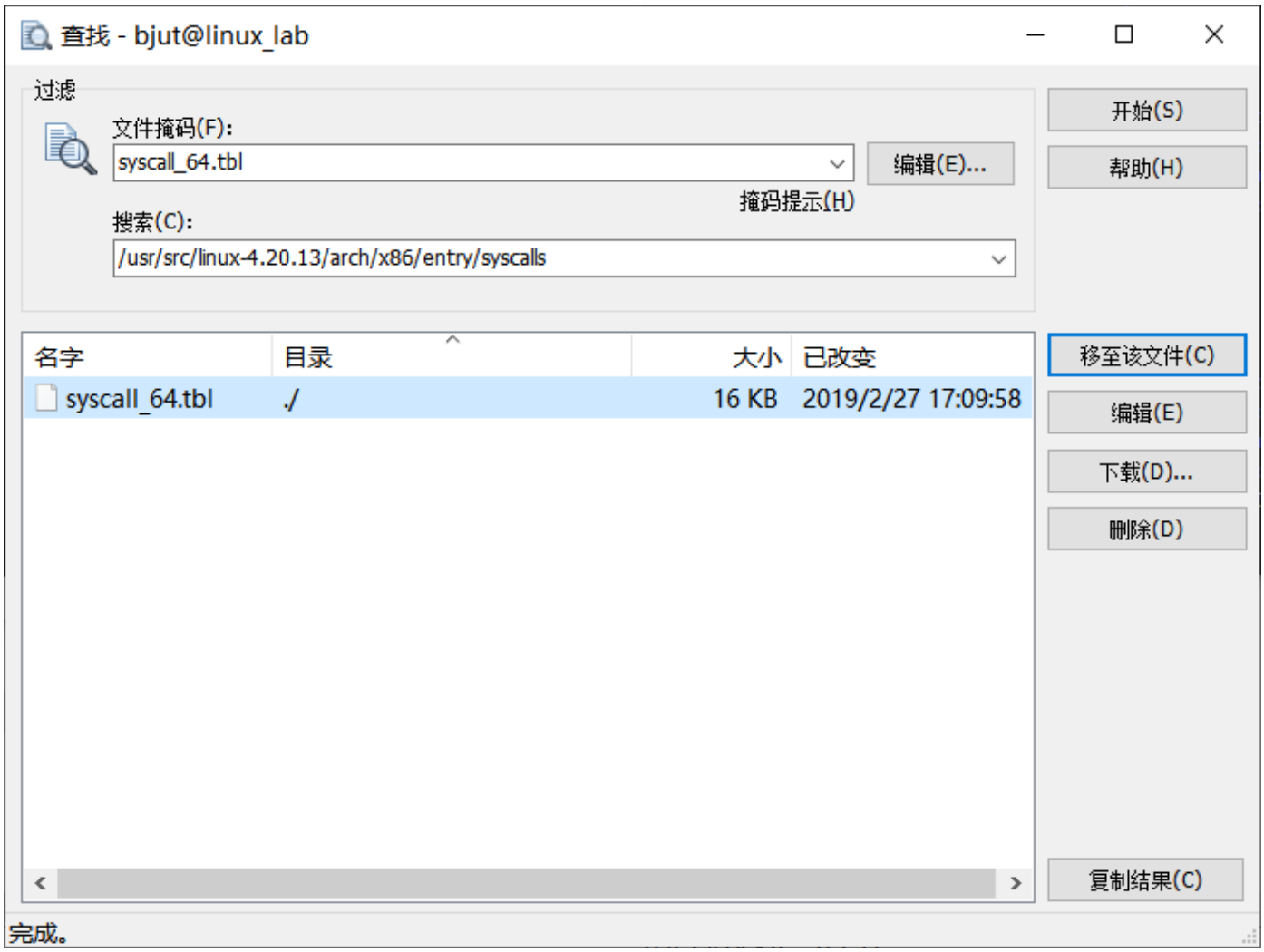


填写完毕后，点击登录即可连接到虚拟机.

软件主界面左侧栏为主机文件系统，右侧栏为虚拟机文件系统，可以通过复制粘贴/拖拽等方法互相传输文件.如果传输文件失败且提示 Permission denied 说明目标文件夹由root用户创建，解决办法为：切换到虚拟机控制台，用root用户在目标文件夹的上级路径输入命令`chmod -R o+rwx 文件夹名`修改文件夹权限，或者将文件复制到其他路径，然后在控制台用root用户的`cp`命令拷贝到目标路径.



通过右侧的查找文件功能可以快速定位文件



附录4 如何使用VS Code/PuTTY远程连接虚拟机

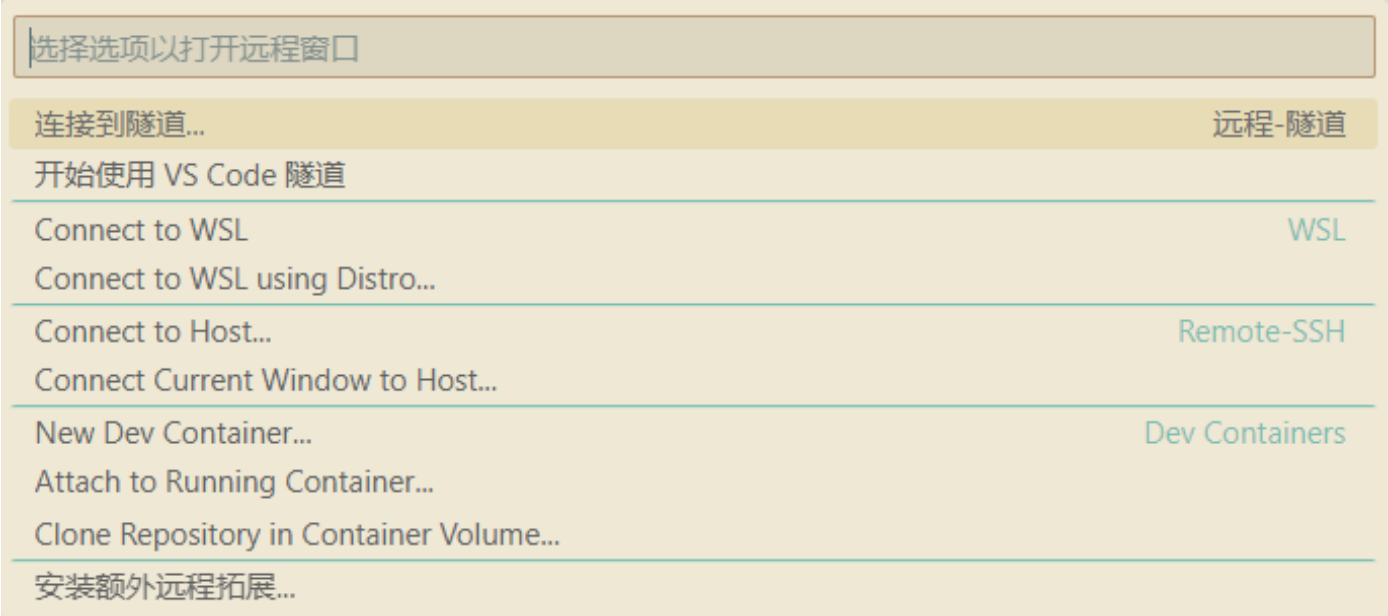
如果出现虚拟机无法调整分辨率、显示不清晰或者无法粘贴主机中的文本内容等问题，可以通过远程连接方式登录虚拟机，本文提供2种软件连接方法的说明：

1. VS Code SSH远程连接扩展
2. PuTTY 一个免费的SSH和Telnet客户端

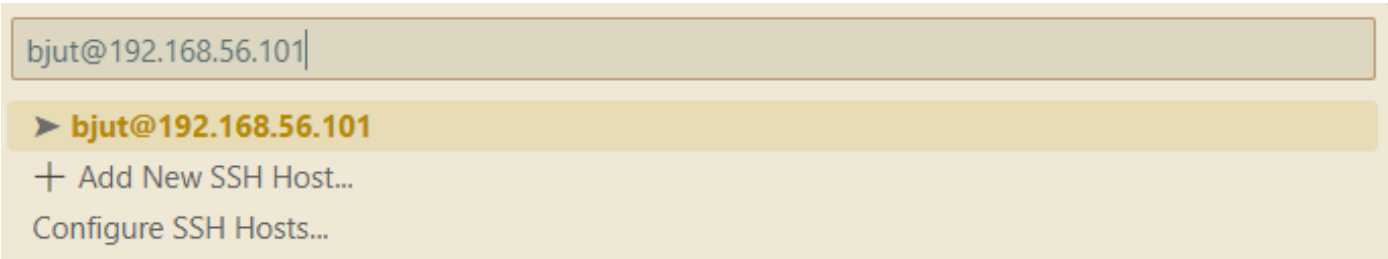
2.1 VS Code SSH远程连接

按照上文WinSCP连接虚拟机的步骤中修改网络设置的部分执行命令，并记下虚拟机的IP地址。

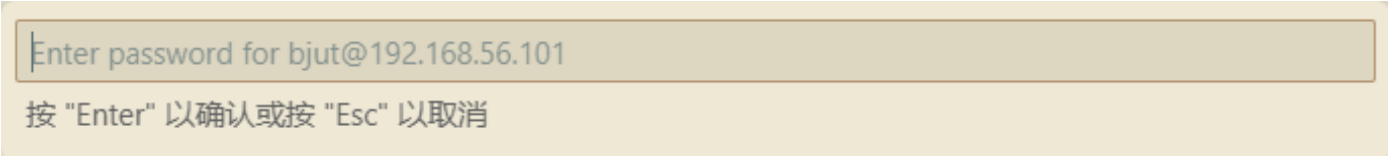
在 VS Code 中安装 Remote-SSH extension，安装完成后左下角出现“打开远程窗口”的图标，点击图标出现命令面板，选择 Remote-SSH 栏中任意一种连接方式（打开新窗口连接或从当前窗口连接）



点击后出现输入框，要求输入登录远程主机的用户名和主机IP地址，格式为 user@host IP



按 Enter 键确认后，左下角显示“正在打开远程...”，若弹出窗口要求选择远程主机的操作系统，请选择 Linux。连接成功后出现密码输入框，按提示操作。



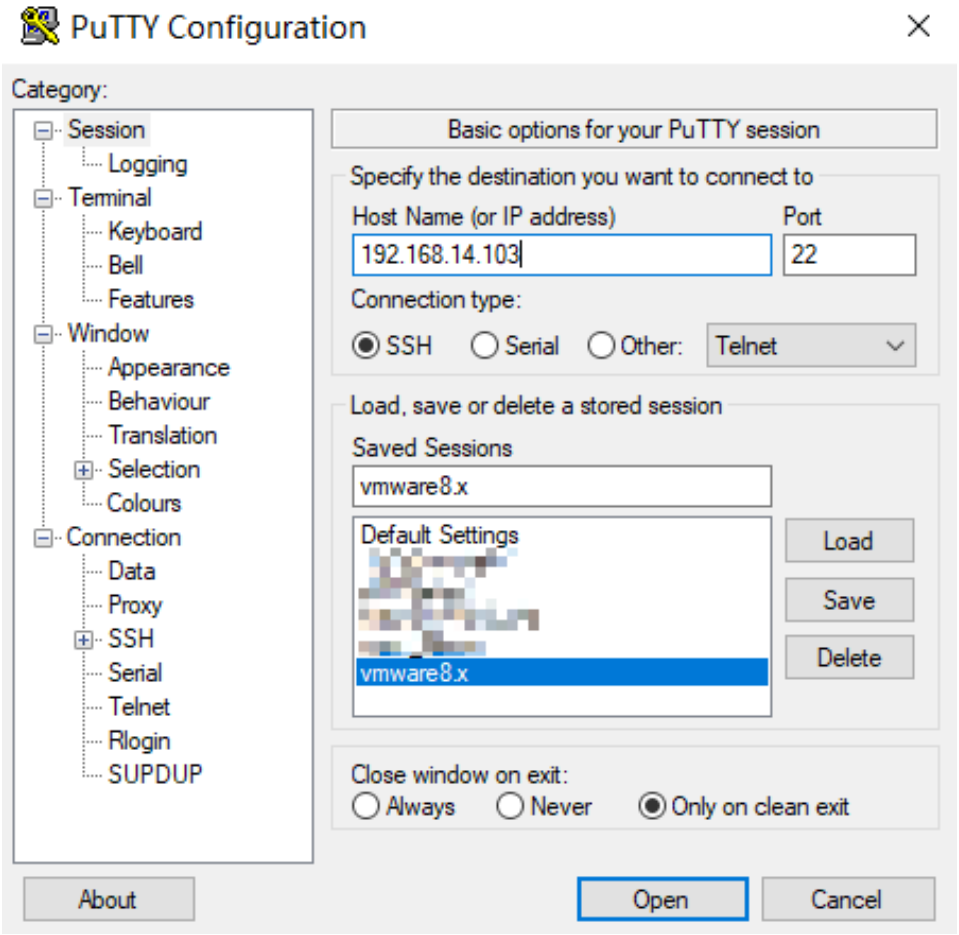
登录成功后，左下角的远程窗口图标会变成显示 `SSH: your host IP` 之后可以打开终端输入要执行的命令，或者点击左侧的资源管理器，选择要打开的文件夹。

2.2 PuTTY远程连接

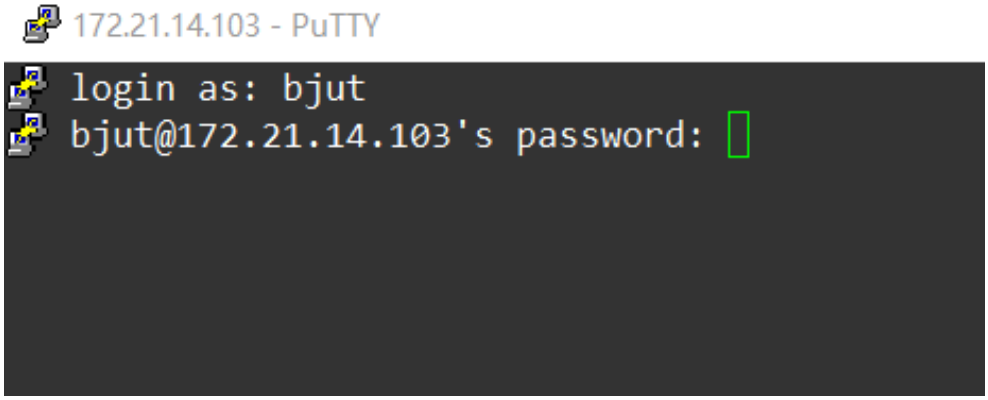
PuTTY是一个免费的SSH和Telnet客户端，用PuTTY连接到虚拟机请按照上文WinSCP连接虚拟机的步骤中修改网络设置的部分执行命令，并记下虚拟机的IP地址。

在PuTTY官网下载软件：https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

打开软件，在Host Name栏输入虚拟机的IP地址，Port默认22，连接方式默认选择SSH。如果想保存设置，在Saved Sessions下方输入框键入会话名称，然后点击Save，可以看到会话列表增加了当前条目。下次打开软件重新连接时，选择该会话然后点击Load，或直接双击会话开始连接。左侧的菜单列表可以更改其他选项，如控制台的字体、使用SSH Key登录等。



点击最下方的Open连接到虚拟机，出现与虚拟机相同的登录界面。



在PuTTY会话窗口中，鼠标单击右键即可在空白处粘贴文本，选中文本后单击右键则会复制文本。

附录5 系统原内核版本较高时手动切换内核的方法

由于bootloader默认启动版本号更高的内核，若使用的系统内核版本号高于4.20.13，需要手动设置启动内核，按如下步骤操作（需要root权限）

（1）打开grub配置文件

```
vim /boot/grub/grub.cfg
```

找到`submenu 'Advanced options for Ubuntu'`开头的配置行，此处列出了系统可用的不同版本的 Linux 内核。若成功编译安装了新内核，可以在这里看到系统原本的内核版本，和新编译的4.20.13版本内核。

```
submenu 'Advanced options for Ubuntu' $menuentry_id_option 'gnulinux-advanced-e05126ed-131d-4967-a06c-84dcbba8d03c' {
    menuentry 'Ubuntu, with Linux 4.20.13' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gn
        ...
    }
    menuentry 'Ubuntu, with Linux 4.20.13 (recovery mode)' --class ubuntu --class gnu-linux --class gnu --class os $menuent
'gnulinux-4.20.13-recovery-e05126ed-131d-4967-a06c-84dcbba8d03c' {
        ...
    }
    menuentry 'Ubuntu, with Linux 4.15.0-171-generic' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id
'gnulinux-4.15.0-171-generic-advanced-e05126ed-131d-4967-a06c-84dcbba8d03c' {
        ...
    }
    menuentry 'Ubuntu, with Linux 4.15.0-171-generic (recovery mode)' --class ubuntu --class gnu-linux --class gnu --class
'gnulinux-4.15.0-171-generic-recovery-e05126ed-131d-4967-a06c-84dcbba8d03c' {
        ...
    }
    menuentry 'Ubuntu, with Linux 4.15.0-167-generic' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id
'gnulinux-4.15.0-167-generic-advanced-e05126ed-131d-4967-a06c-84dcbba8d03c' {
        ...
    }
    menuentry 'Ubuntu, with Linux 4.15.0-167-generic (recovery mode)' --class ubuntu --class gnu-linux --class gnu --class
'gnulinux-4.15.0-167-generic-recovery-e05126ed-131d-4967-a06c-84dcbba8d03c' {
        ...
    }
    menuentry 'Ubuntu, with Linux 4.15.0-156-generic' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id
'gnulinux-4.15.0-156-generic-advanced-e05126ed-131d-4967-a06c-84dcbba8d03c' {
        ...
    }
    menuentry 'Ubuntu, with Linux 4.15.0-156-generic (recovery mode)' --class ubuntu --class gnu-linux --class gnu --class
'gnulinux-4.15.0-156-generic-recovery-e05126ed-131d-4967-a06c-84dcbba8d03c' {
        ...
    }
}
```

（2）输入如下命令用vim打开grub默认文件：

```
vim /etc/default/grub
```

> ...
>
> GRUB_DEFAULT=0
>
> GRUB_TIMEOUT_STYLE=hidden
>
> GRUB_TIMEOUT=2
>
> ...

找到以上配置项，将`GRUB_DEFAULT=0`改为`GRUB_DEFAULT='Advanced options for Ubuntu>Ubuntu, with Linux 4.20.13'`

（3）更新grub并重启

```
update-grub
reboot
```

（4）重启后再次查看当前的内核版本

```
uname -a
```

附录6 搭建实验环境

本节内容介绍在自己的计算机上安装实验环境的步骤。开始前请从在线课程资料下载地址下载镜像文件ubuntu-18.04.6-live-server-amd64.iso(969MB)和内核源代码linux-4.20.13.tar.xz(99.4MB)。

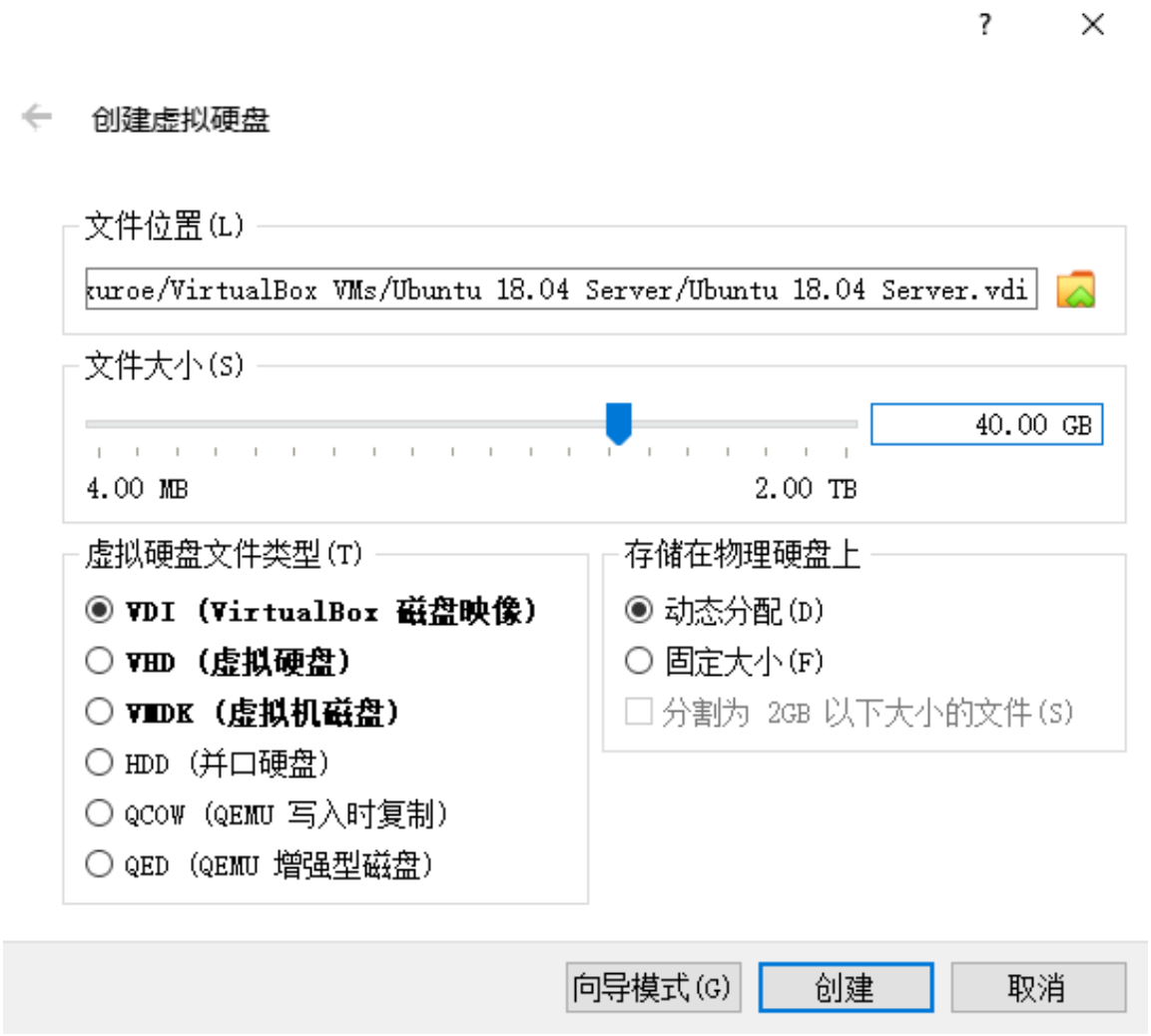
> 镜像文件也可在 https://ubuntu.com/download/server 找到 Alternative downloads 并选择下载 Ubuntu Server 18.04 LTS。LTS版本在停止支持前可能会更新内核版本，请关注官网信息注意下载的系统内核版本，若高于本实验将要编译的4.20.13版本，在安装内核时必须执行额外的步骤切换内核（见附录4）。

推荐安装Server版系统减少占用的硬盘和内存资源。

> 若希望使用图形界面系统，请在 https://ubuntu.com/download/desktop 下载，注意：Desktop版系统的内核版本高于本实验将要编译的4.20.13版本，在安装内核时必须执行额外的步骤切换内核（见附录4）。

（1）新建虚拟机

安装VMware或VritualBox虚拟机管理器软件，演示截图以VritualBox为例，VMware界面与操作流程与VritualBox基本一致。



VritualBox选择创建新的虚拟机，图中的文件夹路径为虚拟硬盘将要存放的位置，不是iso镜像文件的存放位置。虚拟机类型选择Ubuntu 64-bit版本。VMware在选择启动盘/镜像文件页面，请选择之前下载的.iso系统镜像文件。

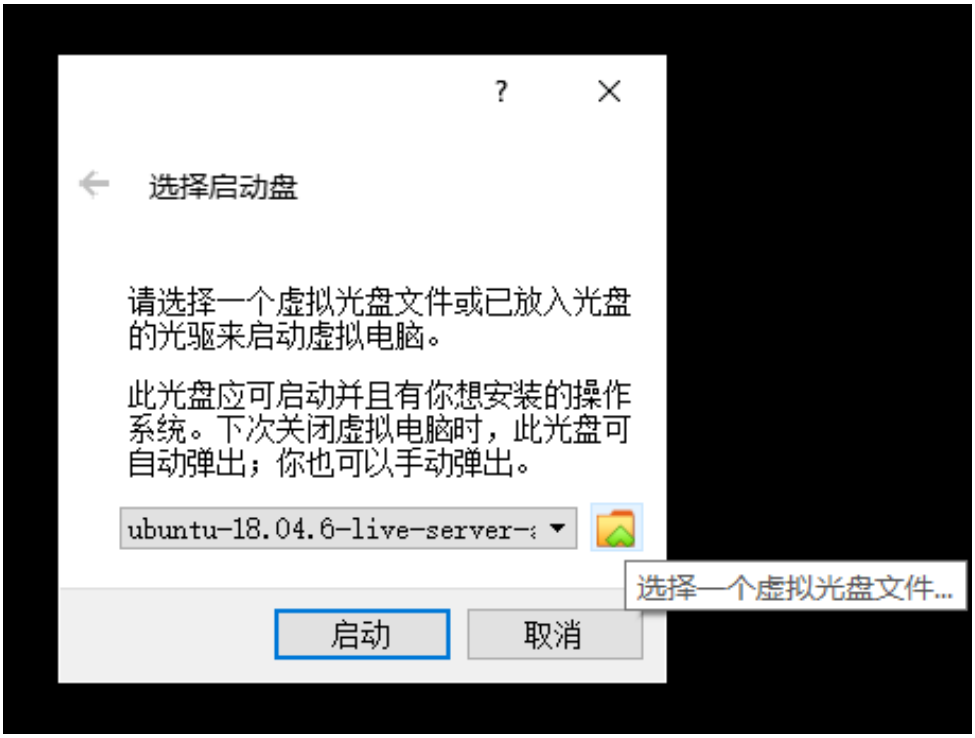
推荐为虚拟机分配资源:

2G以上内存

35G以上硬盘容量

内存小于2G可能在启动新内核时出现`kernel panic — not syncing: system is deadlocked`报错，硬盘小于30G可能出现编译内核时没有足够的空间存放编译生成的文件的问题.

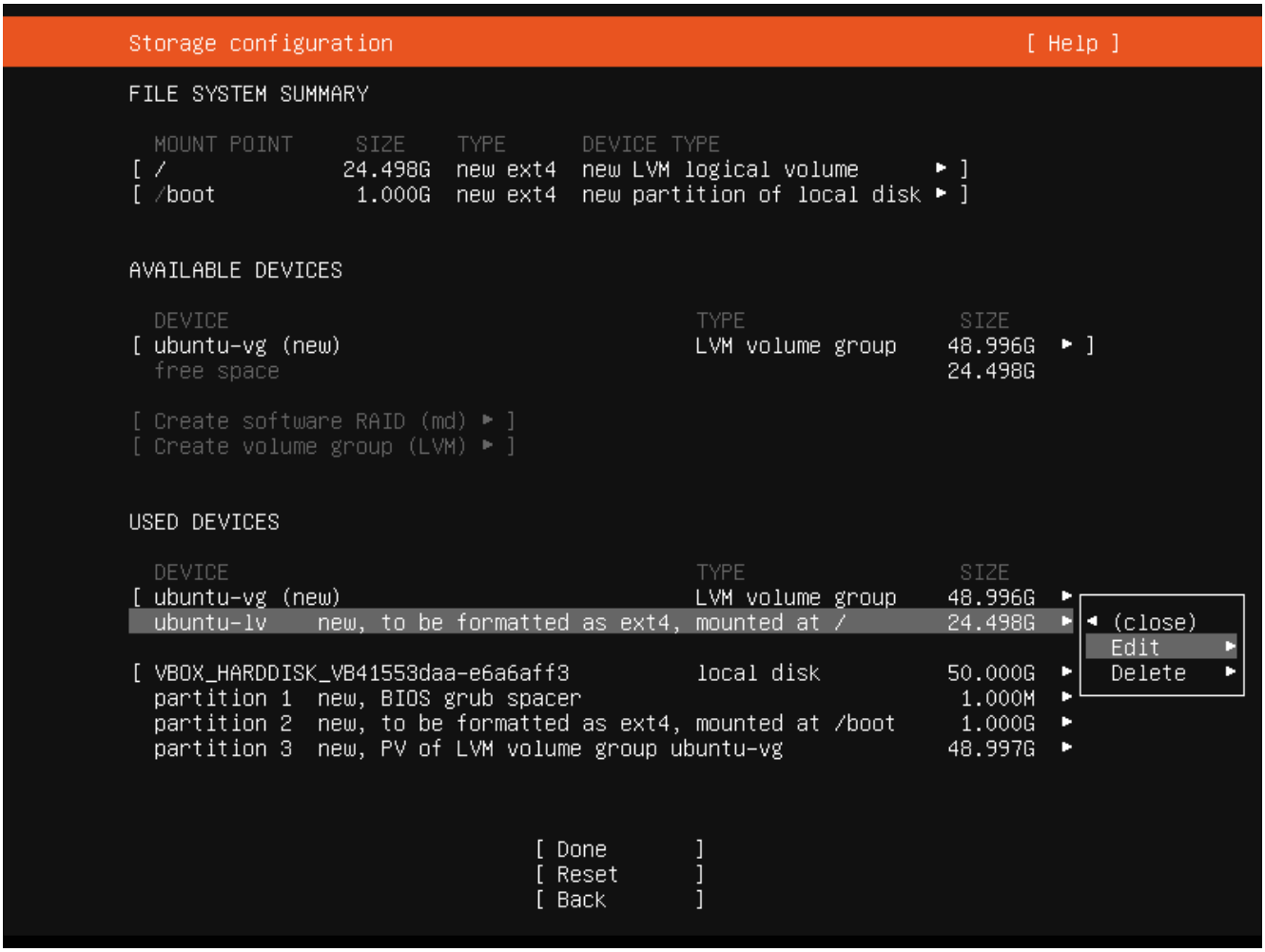
（2）安装虚拟机系统

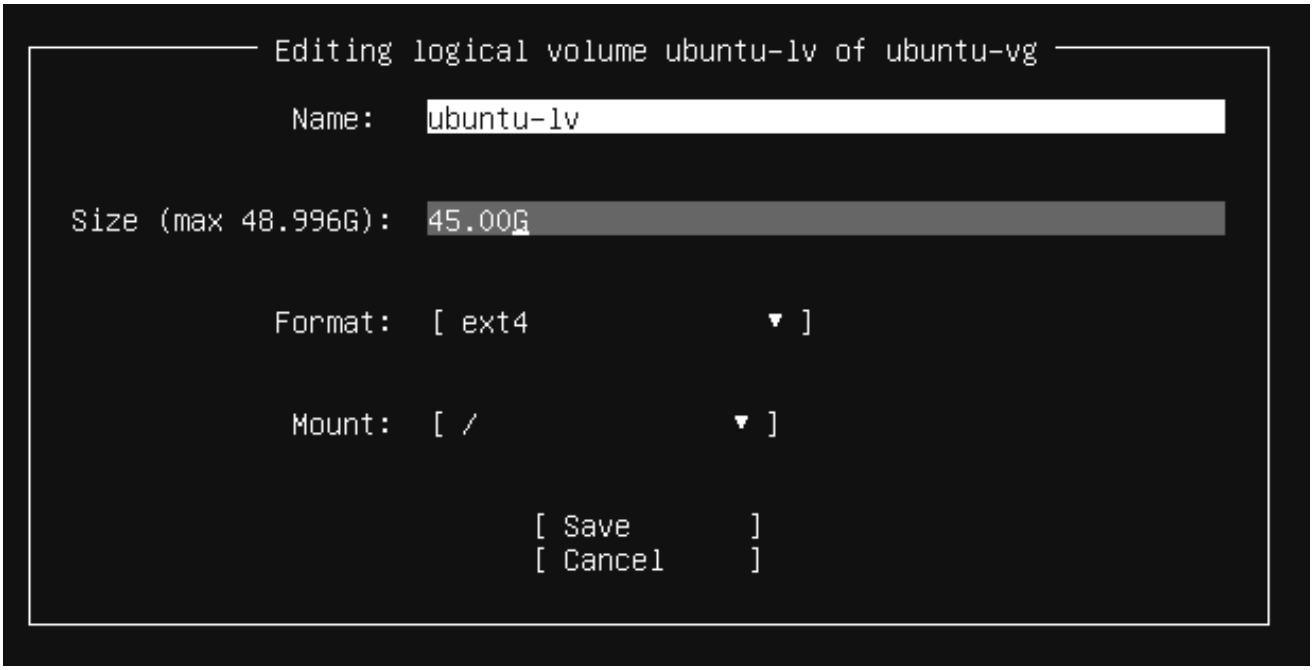


VirtualBox在虚拟机第一次启动时选择启动盘/镜像文件，请选择之前下载的.iso系统镜像文件。点击启动后，开始安装Linux系统。

本文以ubuntu-18.04.6-server版为例：

- 大部分设置保持默认，用方向键选择 [Done]，按下Enter进入下一步
- 提示 installer update available，选择 Continue without updating
- Storage configuration 页面修改文件系统容量，Size**建议设置35G以上**，其他部分保持默认即可。此处一般默认值为20G，如果不进行修改，后续若硬盘空间不够需要手动扩展硬盘





- 设置用户名和密码
- SSH Setup页面推荐选择 [X] Install OpenSSH server，之后可以用PuTTY 和 WinSCP 等软件传输命令和文件.

> Desktop 版安装时推荐在 *更新和其他软件* 页面选择最小安装，并**取消勾选** *其他选项* 下的 *安装Ubuntu时下载更新* 和 *安装第三方软件* 两个选项。

(3) 重启虚拟机

等待系统安装完毕，完成后下方出现 [Reboot Now] 选项，选择并按下Enter重启。

重启后出现如下方所示登录界面 ，输入安装系统时设定的用户名，然后输入用户密码，**密码在键入时不会在屏幕上显示占位符**.

```
Ubuntu 18.04.6 LTS linux_lab tty1

linux_lab login: bjut
Password:
```

本实验全程在root权限下进行，输入`sudo su`切换到root，根据提示输入用户密码。若设置了root密码，则输入命令`su`和root密码，**密码在键入时不会在屏幕上显示占位符**。命令行开头从用户名变为root表示切换成功。

```
bjut@linux_lab:~$ sudo su
[sudo] password for bjut:
root@linux_lab:/home/bjut#
```

（4）安装运行库

键入以下命令，安装相关编译支持库

```
apt-get update
apt install libncurses5-dev libssl-dev -y
apt install build-essential openssl -y
apt install zlibc minizip -y
apt install libidn11-dev libidn11 -y
apt install bison -y
apt install flex -y
```

在安装过程中，注意命令执行结束后控制台是否输出 Error: Err: E: failed 等字样提示安装失败，请仔细检查输入命令是否正确，或者尝试检查网络连接重试.

安装vim

```
apt-get install vim vim
```