中山大學
SUN YAT-SEN UNIVERSITY

LAB1 实验报告

实验课程: 操作系统原理实验

任课教师: 刘宁

实验题目: 编译内核/利用已有内核

专业名称: 计算机科学与技术

学生姓名:

学生学号:

实验地点: 实验中心 B202

实验时间: 2025.2.26

Section 1 实验概述

本实验需要熟悉现有 Linux 内核的编译过程和启动过程，并在编译内核的基础上构建简单应用并启动。同时，要利用精简的 Busybox 工具集构建简单的 OS，熟悉现代操作系统的构建过程。此外，还需要熟悉编译环境、相关工具集，并能够实现内核远程调试。具体内容如下：

(1) 搭建 OS 内核开发环境包括：代码编辑环境、编译环境、运行环境、调试环境等。
(2) 下载并编译 i386（32 位）内核，并利用 qemu 启动内核。
(3) 熟悉制作 initramfs 的方法，并编写简单应用程序随内核启动运行。
(4) 编译 i386 版本的 Busybox，随内核启动，构建简单的 OS。
(5) 开启远程调试功能，进行调试跟踪代码运行。

Section 2 预备知识与实验环境

- 预备知识：Linux 操作系统基础，编译工具使用，C/C++编程基础，调试工具使用（qemu，gdb），内核启动与初始化，虚拟化技术基础等
- 实验环境：
  - 虚拟机版本/处理器型号：VMware-Ubuntu22.04.5/i386（32 位）
  - 代码编辑环境： 编辑器：VSCode
    插件：C/C++插件，汇编插件
  - 代码编译工具： 编译器：gcc
    工具链：binutils、make、qemu、nasm 等
    调试工具：gdb
  - 重要三方库信息：libncurses5-dev：用于终端界面
    bison：用于编译器构建
    flex：词法分析工具
    libssl-dev：SSL 支持库
    libc6-dev-i386：i386 架构的 C 库开发工具 等

Section 3 实验任务

- 实验任务 1：搭建 OS 内核开发环境包括：代码编辑环境、编译环境、运行环境、调试环境等。
- 实验任务 2：下载并编译 i386（32 位）内核，并利用 qemu 启动内核。

- 实验任务 3：制作 initramfs 的方法，编写简单应用程序随内核启动运行。
- 实验任务 4：编译 i386 版本的 Busybox，随内核启动，构建简单的 OS。
- 实验任务 5：开启远程调试功能，进行调试跟踪代码运行。

Section 4 实验步骤与实验结果

---------- **实验任务 1** ----------

- 任务要求：搭建一个完整的操作系统内核开发环境，包括以下几个方面：

(1) 配置运行环境，包括虚拟机（如 VirtualBox）和操作系统（如 Ubuntu）。

(2) 配置编译环境，安装编译工具（如 GCC）以支持 Linux 内核的编译。

(3) 配置代码编辑环境，如安装适合的代码编辑器（例如 VSCode）。

(4) 配置调试环境，安装调试工具（如 QEMU 和 GDB）以支持内核调试。

- 思路分析：

(1) 虚拟机配置：使用 VMwaare 来创建 Ubuntu 虚拟机。

(2) 操作系统设置：安装 Ubuntu 操作系统，配置适当的开发工具。需要注意版本选择，**版本太高会出现不稳定或兼容问题**，使用 Ubuntu 18.04 作为开发平台。

(3) 编译环境配置：安装必要的编译工具（如 GCC、Make）和调试工具（如 QEMU 和 GDB）。确保系统中的环境变量正确配置。**需要注意确定编译器安装以及版本**，可以输入相关指令检查。

(4) 调试工具配置：QEMU 和 GDB 结合使用，QEMU 作为虚拟机运行内核，而 GDB 负责远程调试。

(5) 内核配置：选择合适的内核版本，配置内核的选项，确保启用调试功能，以便在调试过程中获取足够的调试信息。。

- 实验步骤：

1. 安装并配置虚拟机环境：

如果在非 Linux 环境下使用虚拟机，安装并配置 VirtualBox。

在虚拟机上安装 Ubuntu 18.04，并更改下载源为清华镜像源，以提高下载速度：

```
sudo mv /etc/apt/sources.list /etc/apt/sources.list.backup
sudo gedit /etc/apt/sources.list
 # 粘贴清华源配置
```

```
sudo apt update
```

2. 配置代码编辑环境：

安装 VSCode 编辑器，并安装相关插件（C/C++、汇编等）：

```
sudo apt install code
```

3. 安装编译器和必需的开发工具：

```
sudo apt install binutils gcc qemu cmake libncurses5-dev bison flex libssl-dev libc6-dev-i386 gcc-multilib g++-multilib
sudo apt install qemu gdb
```

- 实验结果展示：输入以下指令检查安装情况：

```
lsb_release -a
gcc -v
make -v
qemu-system-i386 --version
gdb --version
code --version
```

```
mafq5@mafq5-virtual-machine:~/桌面$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.5 LTS
Release:        22.04
Codename:       jammy
mafq5@mafq5-virtual-machine:~/桌面$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/11/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none:amdgcn-amdhsa
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 11.4.0-1ubuntu1~22.04' --with-bugurl=file:///
ogram-suffix=-11 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/u
xx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-v
-objc-gc=auto --enable-multiarch --disable-werror --enable-cet --with-arch-32=i686 --with-abi=m64 --with-mul
nvptx/usr,amdgcn-amdhsa=/build/gcc-11-XeT9lY/gcc-11-11.4.0/debian/tmp-gcn/usr --without-cuda-driver --enable
k-serialization=2
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.4.0 (Ubuntu 11.4.0-1ubuntu1~22.04)
mafq5@mafq5-virtual-machine:~/桌面$ make -v
GNU Make 4.3
为 x86_64-pc-linux-gnu 编译
Copyright (C) 1988-2020 Free Software Foundation, Inc.
许可证：GPLv3+：GNU 通用公共许可证第 3 版或更新版本<http://gnu.org/licenses/gpl.html>。
本软件是自由软件：您可以自由修改和重新发布它。
在法律允许的范围内没有其他保证。
mafq5@mafq5-virtual-machine:~/桌面$ qemu-system-i386 --version
QEMU emulator version 6.2.0 (Debian 1:6.2+dfsg-2ubuntu6.25)
Copyright (c) 2003-2021 Fabrice Bellard and the QEMU Project developers
mafq5@mafq5-virtual-machine:~/桌面$ code --version
1.97.2
e54c774e0add60467559eb0d1e229c6452cf8447
x64
```

---------- 实验任务 2 ----------

- 任务要求：

(1) 下载并编译 i386（32 位）内核。

(2) 利用 QEMU 启动内核。

(3) 配置内核的编译选项以支持调试功能。

(4) 生成内核镜像文件 bzImage 和符号表 vmlinux。

- 思路分析：

(1) 下载内核：首先从 Linux Kernel 官网下载合适的内核版本（例如 5.10）。

(2) 解压并进入目录：解压下载的内核源代码并进入该目录。

(3) 配置内核：使用 make i386_defconfig 配置内核为 32 位版本，使用 make menuconfig 进行进一步的调试功能配置。

(4) 编译内核：使用 make -j8 编译内核，生成所需的内核镜像和符号表。

(5) 验证结果：检查生成的内核镜像文件 bzImage 和符号表文件 vmlinux

- 实验步骤：

1. 创建并进入 lab1 目录：

```
mkdir ~/lab1
cd ~/lab1
```

2. 下载 Linux 内核源代码（版本 5.10.19）：

```
wget
https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.10.19.tar.xz
```

3. 解压并进入内核目录：

```
xz -d linux-5.10.19.tar.xz
tar -xvf linux-5.10.19.tar
cd linux-5.10.19
```

4. 配置内核为 i386 32 位版本：

```
make i386_defconfig
```

5. 打开内核配置菜单并启用调试信息：

```
make menuconfig
```

6. 在图形界面中选择 Kernel hacking → Compile-time checks and compiler options，勾选 Compile the kernel with debug info。保存并退出配置界面。

7. 编译内核：

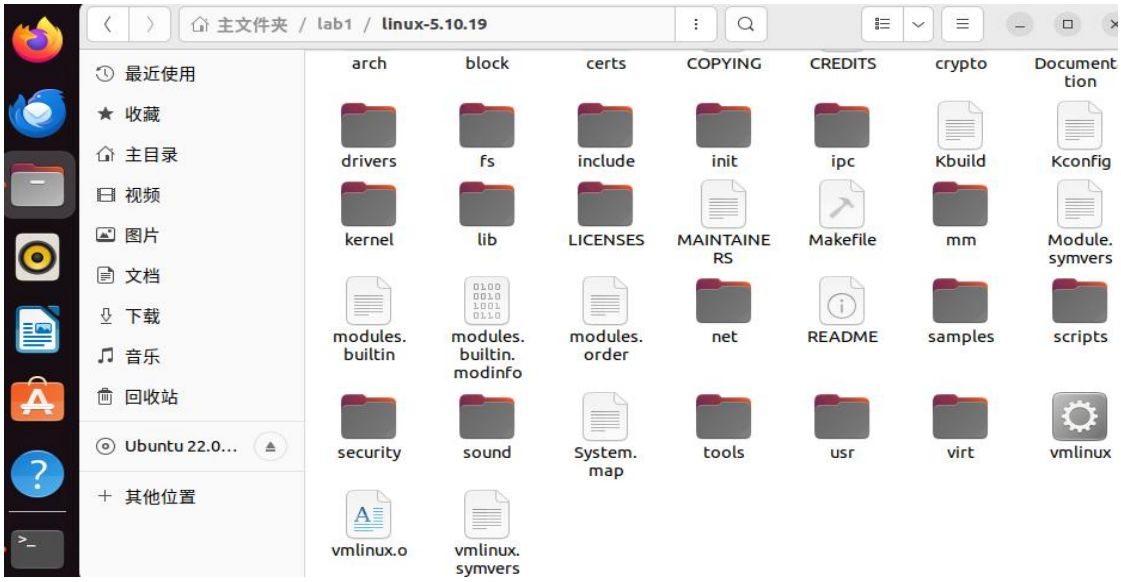
```
make -j8
```

8. 检查生成的内核镜像和符号表：

```
ls
```

- 实验结果展示：通过执行前述代码，可得下图结果。

```
mafq5@mafq5-virtual-machine:~/lab1/linux-5.10.19$ make -j8
  CALL    scripts/atomic/check-atomics.sh
  CALL    scripts/checksyscalls.sh
  CHK     include/generated/compile.h
Kernel: arch/x86/boot/bzImage is ready  (#1)
mafq5@mafq5-virtual-machine:~/lab1/linux-5.10.19$ ls
arch    certs     CREDITS  Documentation  fs       init  Kbuild   kernel  LICENSES
     Makefile  modules.builtin          modules.order   net     samples  securit
y  System.map  usr   vmlinux    vmlinux.symvers
block   COPYING  crypto   drivers        include  ipc   Kconfig  lib     MAINTAIN
ERS  mm         modules.builtin.modinfo  Module.symvers  README  scripts  sound
   tools        virt  vmlinux.o
```



上图说明内核编译已经完成并生成了可用的内核镜像文件以及符号表。

---------- 实验任务 3 ----------

- 任务要求：

(1) 使用 QEMU 启动已编译的内核并开启远程调试功能。

(2) 使用 GDB 连接到 QEMU 进行调试。

(3) 在 GDB 中设置断点，检查内核启动过程，并通过调试定位到内核问题（如 initrd_load 错误）。

- 思路分析：

(1) 启动 QEMU：使用 qemu-system-i386 启动内核，并启用调试功能（-s -S），等待 GDB 的连接。

(2) GDB 调试：通过 GDB 连接到 QEMU，加载内核符号表，并设置调试断点（如在 start_kernel 函数处）。

(3) 调试过程：在调试过程中，检查内核启动时的调用堆栈，排除启动过程中出现的问题。Kernel panic 错误表明内核未能成功启动，通常是由于缺少 initrd 文件，导致无法加载 initramfs 文件系统。

- 实验步骤：

（1）进入实验目录：

cd ~/lab1

（2）启动 QEMU： 使用以下命令启动 QEMU 并开启远程调试：

```
   qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -s -S
-append "console=ttyS0" -nographic
```

此时 QEMU 不会输出信息，因为它在等待 GDB 输入。

（3）启动 GDB： 在另一个终端中启动 GDB：

gdb

（4）加载符号表： 在 GDB 中加载内核符号表：

```
   file linux-5.10.19/vmlinux
```

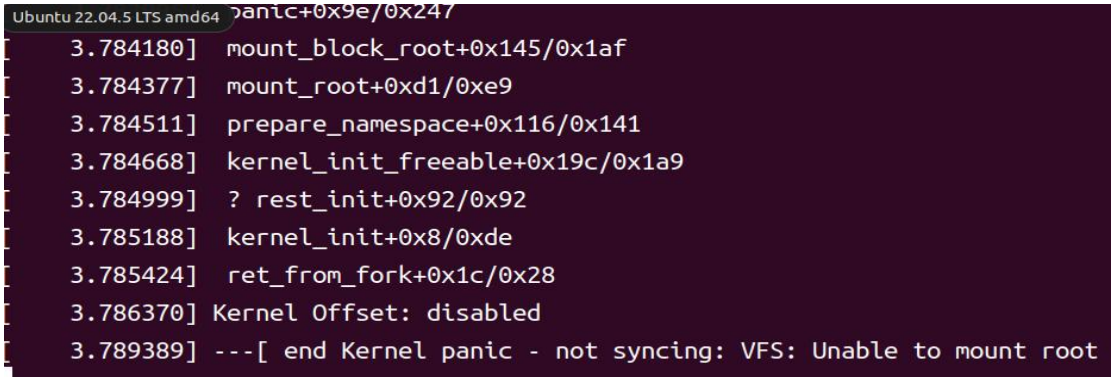
（5）连接到 QEMU： 在 GDB 中连接到 QEMU：

```
   target remote:1234
   break start_kernel
```

（6）继续执行： 输入以下命令继续执行内核：

```
   c
```

- 实验结果展示：



调试输出： 当内核启动时，如果出现 Kernel panic 错误，GDB 会显示调用栈信息，定位到出错的位置（如 initrd_load）。这表明系统未能加载 initramfs 文件，原因是没有指定 initrd 文件。

- 实验注意：

确保 QEMU 和 GDB 都在不同的终端中运行，且不要关闭 QEMU 所在的终端。在调试过程中，如果出现 Kernel panic 错误是正常结果，等待后续实验。

---------- 实验任务 4 ----------

- 任务要求：

(1) 创建一个简单的 initramfs，并在其中包含一个 "Hello World" 程序。

(2) 使用 cpio 将程序打包成 initramfs 文件。

(3) 使用 QEMU 启动内核，并加载该 initramfs。

(4) 在启动过程中，通过 GDB 调试，验证 initramfs 中的程序是否能够成功执行，输出 "Hello World"。

- 思路分析：

(1) 创建 Hello World 程序：首先创建一个简单的 C 程序，通过 printf 输出 "Hello World"。

(2) 编译为 32 位可执行文件：将程序编译为静态链接的 32 位可执行文件，确保它可以在 initramfs 中运行。

(3) 打包 initramfs：使用 cpio 将编译好的可执行文件打包成 initramfs，以便内核启动时加载。

(4) 启动内核：使用 QEMU 启动内核，并加载自定义的 initramfs 文件，确保系统能够在启动过程中执行该程序。

- 实验步骤：

（1）进入实验目录：

```
cd ~/lab1
```

编写 Hello World 程序： 创建一个简单的 helloworld.c 文件，内容如下：

```
#include <stdio.h>
void main() {
    printf("lab1: Hello World\n");
    fflush(stdout);
    /* 让程序打印完后继续维持在用户态 */
```

```
        while(1);
    }
```

（2）编译程序： 将 helloworld.c 编译为 32 位静态链接的可执行文件：

```
    gcc -o helloworld -m32 -static helloworld.c
```

（3）打包 initramfs： 使用 cpio 将 helloworld 可执行文件打包成 initramfs 文件：

```
    echo helloworld | cpio -o --format=newc > hwinitramfs
```

（4）启动内核并加载 initramfs： 使用 QEMU 启动内核并加载创建的 initramfs 文件：

```
    qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd
hwinitramfs -s -S -append "console=ttyS0 rdinit=helloworld" -nographic
```

（5）调试过程：

启动 GDB，连接到 QEMU 进行远程调试。

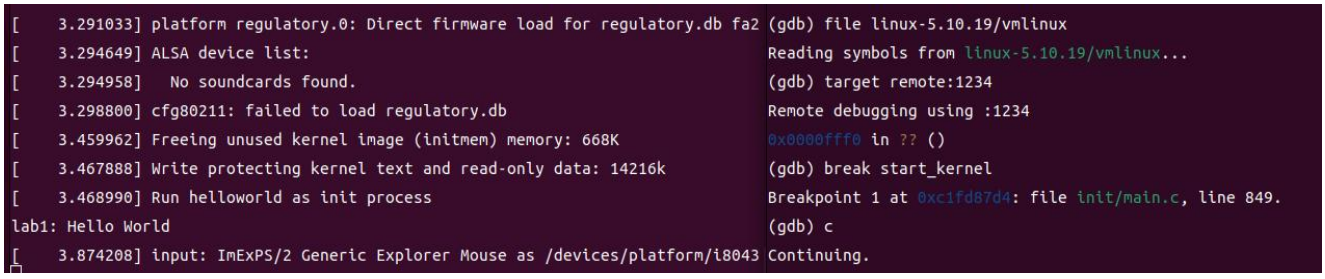
设置断点并继续执行，检查输出：

```
    gdb
    (gdb) file linux-5.10.19/vmlinux
    (gdb) target remote:1234
    (gdb) break start_kernel
    (gdb) c
```

验证输出： 在 GDB 调试过程中，验证是否在终端中成功输出：

lab1: Hello World

- 实验结果展示：通过执行前述指令，可得下图结果。



---------- 实验任务 5 ----------

- 任务要求：

(1) 下载并编译 BusyBox 作为静态二进制文件。

(2) 配置并生成 initramfs，其中包含 BusyBox 和一个简单的 init 程序。

(3) 使用 QEMU 启动内核，并加载生成的 initramfs，在启动过程中执行 BusyBox。

- 思路分析：

(1) 下载并编译 BusyBox：首先下载 BusyBox 源代码，并通过 make 编译成静态二进制文件。

(2) 配置 BusyBox：在编译前，需要设置编译选项，确保编译生成静态二进制文件，并为 i386 架构配置适当的编译标志。

(3) 制作 initramfs：将编译好的 BusyBox 安装到指定目录，并创建一个简单的 init 脚本。然后将其打包成 initramfs 文件，供 QEMU 启动时使用。

(4) 启动内核并加载 initramfs：通过 QEMU 启动内核，并加载刚刚制作的 initramfs 文件，确保内核可以正常加载 BusyBox 并执行初始化。

- 实验步骤：

（1）进入实验目录，下载并解压 BusyBox： 从课程网站下载 BusyBox 源代码并解压：

```
cd ~/lab1
tar -xf Busybox_1_33_0.tar.gz
```

（2）编译 BusyBox： 配置并编译 BusyBox 为静态二进制文件：

```
make defconfig
make menuconfig
```

（3）在菜单中，进入 Settings，选择 Build BusyBox as a static binary (no shared libs)，并在 Additional CFLAGS 和 Additional LDFLAGS 中设置为：

Additional CFLAGS: -m32 -march=i386

Additional LDFLAGS: -m32

（4）保存并退出后，编译 BusyBox：

```
make -j8
make install
```

（5）制作 initramfs： 将编译好的 BusyBox 文件复制到目标目录，并创建 initramfs：

```
mkdir -pv mybusybox/{bin,sbin,etc,proc,sys,usr/{bin,sbin}}
cp -av busybox-1_33_0/_install/* mybusybox/
```

（6）创建 init 脚本： 使用 gedit 创建一个简单的 init 脚本作为启动时执行的程序：

```
gedit mybusybox/init
```

脚本内容如下：

```
    mount -t proc none /proc
    mount -t sysfs none /sys
    echo -e "\nBoot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
    exec /bin/sh
```

设置 init 脚本权限：

```
    chmod u+x init
```

（7）打包 initramfs： 使用 cpio 将 BusyBox 和 init 脚本打包成 initramfs 文件：

```
    find . -print0 | cpio --null -ov --format=newc | gzip -9 >
~/lab1/initramfs-busybox-x86.cpio.gz
```

（8）启动内核并加载 initramfs： 使用 QEMU 启动内核并加载生成的 initramfs：

```
    qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bzImage -initrd
initramfs-busybox-x86.cpio.gz -nographic -append "console=ttyS0"
```

（9）验证输出： 在 QEMU 启动过程中，使用 ls 等命令查看文件夹内容，验证 BusyBox 是否正常加载并执行：

```
    ls
    pwd
    Echo”hello ,busybox”
```

- 实验结果展示：通过执行前述代码，可得下图结果。

```
    [*] Build static binary (no shared libs)
    [ ] Force NOMMU build
    ()  Cross compiler prefix
    ()  Path to sysroot
    (-m32 -march=i386) Additional CFLAGS
    (-m32) Additional LDFLAGS
    ()  Additional LDLIBS
```

```
Please press Enter to activate this console. [    3.725956] input: ImExPS/2 Gen3

/ # ls
bin      etc      linuxrc  root     sys
dev      init     proc     sbin     usr
/ # pwd
/
/ # echo "Hello, BusyBox!"
Hello, BusyBox!
```

● Section 5 实验回顾总结与心得体会

（1)深入理解操作系统启动过程：通过编译内核和制作 initramfs，我对 Linux 系统的启动过程有了更清晰的认识。

（2）调试技能的提升：使用 QEMU 和 GDB 进行远程调试，让我掌握了内核调试的技巧，能够分析和解决启动过程中的问题。

（3）编译与配置能力：通过配置编译选项和调试环境，我熟悉了如何编译内核并解决相关配置问题，增强了系统定制能力。

（4）附上具体遇到的问题

```
mafq5@OSlab:~/桌面/lab1$ qemu-system-i386 -kernel linux-5.10.19/arch/x86/boot/bz
Image -s -S -append "console=ttyS0" -nographic
找不到命令 “qemu-system-i386”，但可以通过以下软件包安装它：
sudo apt install qemu-system-x86      # version 1:6.2+dfsg-2ubuntu6.23, or
sudo apt install qemu-system-x86-xen  # version 1:6.2+dfsg-2ubuntu6.23
```

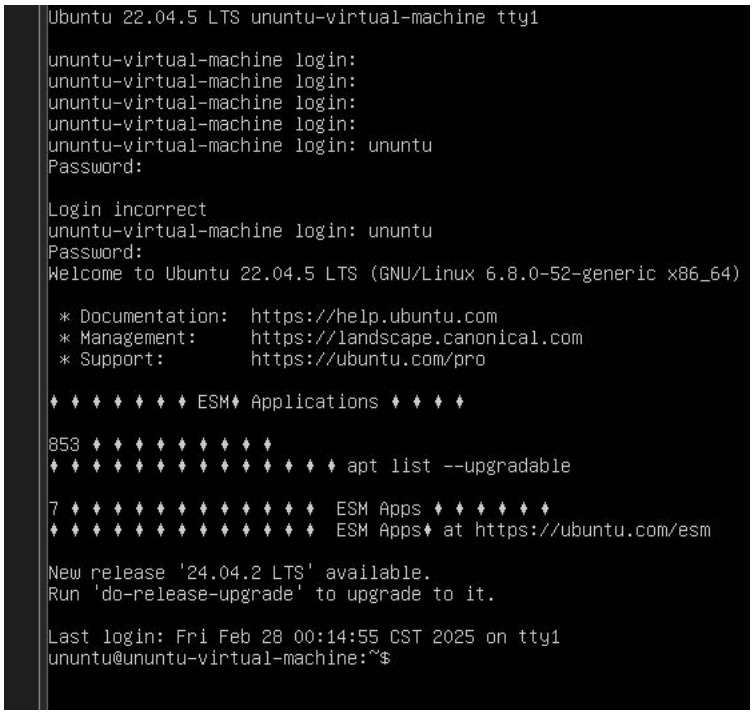
以上是由于 ubuntu 版本指令引起，更换合适指令即可

```
mafq5@OSlab:~/桌面$ df -h
文件系统        大小  已用  可用 已用% 挂载点
tmpfs           476M  2.0M  474M    1% /run
/dev/sda3        20G   18G  155M  100% /
tmpfs           2.4G     0  2.4G    0% /dev/shm
tmpfs           5.0M  4.0K  5.0M    1% /run/lock
tmpfs           2.4G     0  2.4G    0% /run/qemu
/dev/sda2       512M  6.1M  506M    2% /boot/efi
tmpfs           476M  108K  475M    1% /run/user/1000
/dev/sr0        4.5G  4.5G     0  100% /media/mafq5/Ubuntu 22.04.5 LTS amd64
```

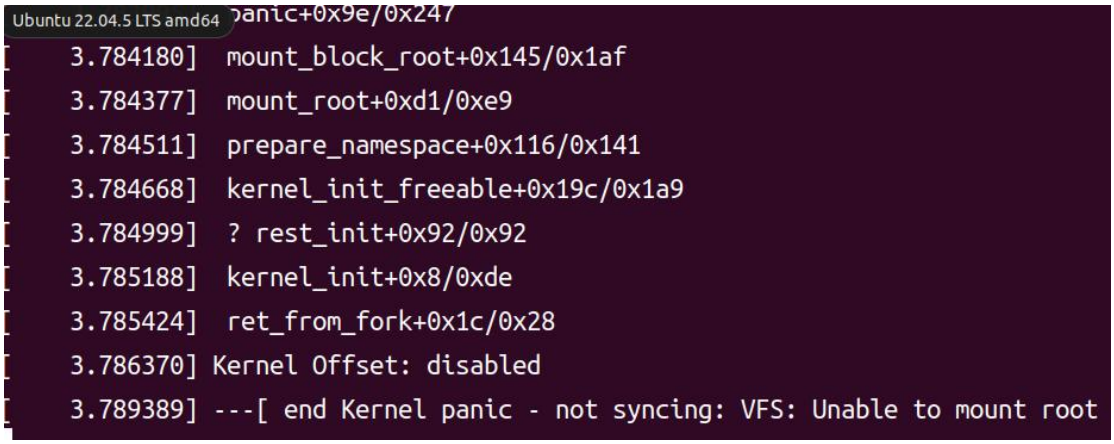
以上是由于建虚拟机时内存分配不足引起，解决方法：

1 重新安装

2 扩展，但是实践发现容易把系统原有的包搞掉，如下：



这时候需要重装图形界面及相关包（比较麻烦）。



出现 Kernel panic 错误，GDB 会显示调用栈信息，定位到出错的位置（如 initrd_load）。这表明系统未能加载 initramfs 文件，原因是没有指定 initrd 文件。

Section 6 附录：参考资料清单

1 除了实验指导书，部分指令参考了 chatgpt.

2 参考了 csdn 等相关文章

https://lists.busybox.net/pipermail/busybox-cvs/2024-January/041752.html

https://blog.csdn.net/2202_75820736/article/details/142054146

https://www.cnblogs.com/Jelline/articles/2063565.html

https://blog.csdn.net/dyrlovewc/article/details/80817026