

# 操作系统

# 实验手册

V2023



中国科学院软件研究所  
Institute of Software, Chinese Academy of Sciences

湖南大学

华为技术有限公司

中科院软件所

# 目录

---

前言 .....	1
<b>1 实验一鲲鹏云 ECS 的构建及内核编译 .....</b>	<b>4</b>
1.1 实验介绍 .....	4
1.1.1 任务描述 .....	4
1.2 实验目的 .....	4
1.3 构建云实验环境 .....	4
1.3.1 创建 VPC .....	4
1.3.2 购买 ECS .....	7
1.3.3 通过 ssh 登录系统 .....	9
1.4 实验任务 .....	10
1.4.1 openEuler 内核编译与安装 .....	10
1.4.2 Hello, world! .....	13
1.5 其他内核编程实验 .....	14
1.6 云环境资源清理 .....	15
1.6.1 ECS 关机 .....	15
1.6.2 删除 ECS .....	15
<b>2 实验二 内核模块编程 .....</b>	<b>18</b>
2.1 内核模块编程 .....	18
2.1.1 四个模块编程 .....	18
2.2 思考题 .....	19
<b>3 实验三 内存管理 .....</b>	<b>20</b>
3.1 内存分配和管理之 kmalloc 和 vmalloc .....	20
3.1.1 用 kmalloc 分配 1KB、8KB 的内存并打印指针地址 .....	20
3.1.2 用 vmalloc 分配 8KB、1MB、64MB 的内存并打印指针地址 .....	21
3.1.3 使用首次适应算法实现一个简单的 malloc 内存分配器，满足内存分配和释放 .....	21
3.2 内存分配和管理之 iomap .....	22
3.2.1 申请、读写、释放 I/O 端口 .....	22
3.2.2 申请、读写、释放 I/O 内存 .....	23
3.3 思考题 .....	23
<b>4 实验四 中断和异常 .....</b>	<b>24</b>

4.1 使用 tasklet 完成一个中断程序 .....	24
4.1.1 使用 tasklet 打印第 31 号中断调用情况 .....	24
4.2 工作队列 .....	24
4.2.1 使用工作队列打印当前日期 .....	24
4.3 思考题 .....	25
<b>5 实验五 内核网络管理 .....</b>	<b>26</b>
5.1 利用 eBPF 机制实现简单的防火墙 .....	26
5.1.1 利用 XDP 编写 eBPF 过滤程序 .....	26
5.2 思考题 .....	28
<b>6 实验六 文件系统 .....</b>	<b>29</b>
6.1 文件系统的简单实现 .....	29
6.1.1 参考 ramfs 文件系统，实现一个简单的内存文件系统 .....	29
6.2 思考题 .....	30
<b>7 实验七 生产者消费者问题 .....</b>	<b>31</b>
7.1 实验目的 .....	31
7.2 预习要求 .....	31
7.3 实验设备与环境 .....	31
7.4 实验原理 .....	31
7.5 实验任务 .....	32
7.6 实验步骤和方法 .....	32
<b>8 实验八 进程调度 .....</b>	<b>34</b>
8.1 实验目的 .....	34
8.2 预习要求 .....	34
8.3 实验设备与环境 .....	34
8.4 实验原理 .....	34
8.5 实验任务 .....	35
8.5.1 动态优先级调度 .....	35
8.5.2 时间片轮转调度 .....	35
8.6 实验步骤和方法 .....	36
8.6.1 动态优先级调度算法 .....	36
8.6.2 时间片轮转调度算法 .....	36

# 前言

鲲鹏云是“华为云鲲鹏云”的简称，本实验手册是基于鲲鹏云弹性云服务器（ECS）的 openEuler 操作系统内核编程实验手册，包括了内核编译、内存管理、进程管理、中断异常管理、内核时间管理、设备管理、文件管理以及网络管理等内核相关实验。

本实验手册不仅包含实验步骤，还在每章简要介绍了实验相关的原理和背景知识，以便读者能更好地理解操作系统内核的原理并进行实验。

## 一、实验网络环境介绍

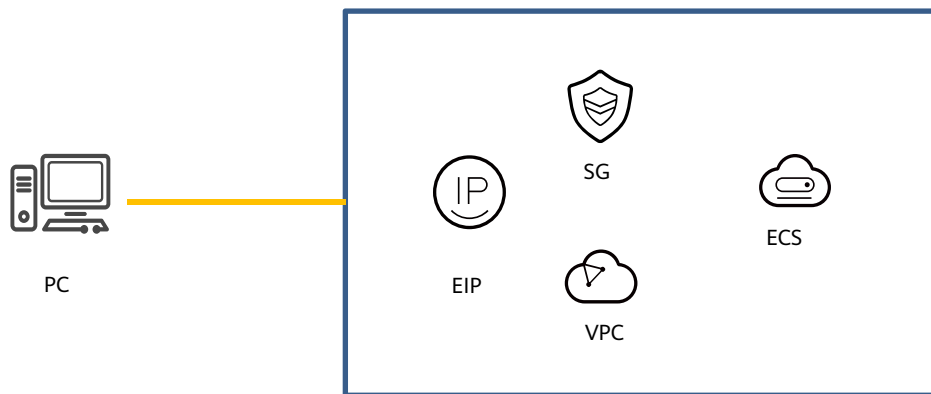


图 1-1 openEuler 操作系统实验的云环境

如上图所示，本实验的云环境主要是一台基于鲲鹏架构的 ECS，附加部件有弹性公网 IP（EIP）、虚拟私有云（VPC）和安全组（SG）。学生端的 PC 机使用 ssh 终端登录 ECS。

## 二、实验设备介绍

关键配置如下表所示：

表 1-1 关键配置

云资源	规格
ECS 鲲鹏计算	4vCPUs   8GB   40GB
EIP 带宽	按流量计费

软件方面，本实验需要一台终端电脑与弹性云服务器(ECS)链接以输入操作命令或/和传输文件。对于 Windows 10 / macOS / Linux，我们可以用命令行工具 ssh 和 scp 完成这个过程。

如果在有些 Windows 系统下不能运行 ssh 工具，也可以使用 Putty 和 WinSCP 工具软件。其中 Putty 工具的推荐下载地址：

<https://hcia.obs.cn-north-4.myhuaweicloud.com/v1.5/putty.exe>

WinSCP 的推荐下载地址:

<https://winscp.net/eng/index.php>

下文若无特殊说明，均以命令行工具 ssh 和 scp 为例进行讲解。

### 特别说明:

内核编程实验需要在完成实验一编译内核的基础上，并且由新的内核引导系统后进行。

本实验手册的实验内容和知识点分布请见下表:

**表 1-2 实验内容和相关知识点**

序号	章节	实验内容	涉及知识点
1	鲲鹏云 ECS 的构建及内核编译	1.构建鲲鹏云 ECS 2.编译安装 openEuler 操作系统新内核 3.简单的内核模块编程实验，在内核模块中打印 “Hello, world!”	掌握鲲鹏云 ECS 的构建，编译安装操作系统新内核并了解简单的内核模块编程
2	内核模块编程	1. 内核模块编程实践	Linux 内核模块编程方法
3	内存管理	1. 以 kmalloc 和 vmalloc 分配和管理内存 2. 以 iomap 申请和管理 I/O 端口和内存	在内核态下对内存的申请、释放和管理
4	中断和异常	1. 使用 tasklet 完成中断程序 2. 工作队列内核编程	了解内核中中断和异常的管理
5	内核网络管理	1. eBPF 机制实现简单防火墙	了解内核如何遵循网络协议对网络数据包的分层处理
6	文件系统	1. 内存文件系统的简单实现	文件系统实现
7	生产者消费者问题	1.展示进程(线程)创建及其同步与互斥的过程	同步互斥算法，理解生产者消费者模型
8	进程调度	1.用高级语言编写和调试一个简单的进程调度程序	进程调度实现

**约定：**

本文档示例中如果有斜体字出现在命令行中则表示用户需要根据自己情况或需要输入相应内容；若出现在显示结果中，则意味着该结果依赖于您自己的运行环境从而导致其与本示例中的可能不一样。如：

```
# make
# insmod kmalloc.ko
# lsmod | grep kmalloc
kmalloc          262144  0
# rmmod kmalloc
# dmesg | tail -n6
[  44.653995] kmalloc: loading out-of-tree module taints kernel.
[  44.654464] kmalloc: module verification failed: signature and/or required key missing - tainting kernel
[  44.655688] Start kmalloc!
[  44.655879] kmallocmem1 addr = ffff800171e53000
[  44.656181] kmallocmem2 addr = ffff8001704fe000
[ 106.904519] Exit kmalloc!
# make clean
```

**感谢：**

本实验手册中引用了华为技术有限公司和上海交通大学陈全老师的实验教学用例，在此表示感谢！

# 1

## 实验一鲲鹏云 ECS 的构建及内核编译

---

### 1.1 实验介绍

本实验通过构建鲲鹏云 ECS、编译安装 openEuler 操作系统新内核以及简单的内核模块编程任务操作带领大家了解操作系统以及内核编程。

#### 1.1.1 任务描述

- 构建鲲鹏云 ECS
- 编译安装 openEuler 操作系统新内核
- 简单的内核模块编程实验，在内核模块中打印 “Hello, world!”

### 1.2 实验目的

- 学习掌握如何安装构建 ECS
- 学习掌握如何编译操作系统内核
- 了解内核模块编程。

### 1.3 构建云实验环境

#### 1.3.1 创建 VPC

步骤 1 在浏览器地址栏输入华为云控制台网址 [console.huaweicloud.com](https://console.huaweicloud.com) 并按回车键，这时页面将跳转至登录页。



账号登录

账号名/邮箱

密码

手机号登录 ☒ 记住登录名

登录

[免费注册](#) | [忘记密码](#) | [IAM用户登录](#) | [HUAWEI ID登录](#)


[使用其他账号登录](#)

步骤 2 按要求输入账号密码，进行登录。

注意：在此之前您需要在华为云主页注册华为云账号。

步骤 3 登录成功后会自动进入控制台页面，这时将区域选在“华北-北京四”。



步骤 4 将鼠标悬停于左侧导航栏图标处展开服务列表，然后在服务列表中点击“虚拟私有云 VPC”项。



步骤 5 点击“虚拟私有云”控制台页面右上角的“创建虚拟私有云”按钮。





步骤 6 在创建虚拟私有云的页面中按照下表内容配置虚拟私有云参数。

参数	配置
区域	华北-北京四
名称	vpc-test
网段	192.168.1.0/24
企业项目	default
默认子网可用区	可用区1
默认子网名称	subnet-test
子网网段	如192.168.1.0/24

步骤 7 配置完成后，点击“立即创建”，创建完成后会自动回到 VPC 控制台。

步骤 8 点击 VPC 控制台左侧导航栏的“访问控制”→“安全组”，进入安全组控制台。



步骤 9 点击右上角的“创建安全组”。



步骤 10 在弹出的对话框中按“通用 Web 服务器”配置安全组参数，然后点击“确定”。



创建安全组

\* 名称: sg-test

\* 模板: 通用Web服务器


描述: 通用Web服务器，默认放通22、3389、80、443端口和ICMP协议。适用于需要远程登录、公网ping及用于网站服务的云服务器场景。

0/255

[查看模板规则](#)

确定 取消

### 1.3.2 购买 ECS

步骤 1 将鼠标悬停于左侧导航栏  图标处展开服务列表。然后在服务列表中点击“弹性云服务器 ECS”项。



步骤 2 点击弹性云服务器 ECS 控制台页面右上角的“购买弹性云服务器 ECS”按钮进入购买页面。



步骤 3 按照下表内容配置弹性云服务器 ECS 的参数。

参数	配置
计费模式	按需计费
区域	华北-北京四
可用区	可用区1
CPU架构	鲲鹏计算
规格	鲲鹏通用计算增强型   kc1.xlarge.2   4vCPUs   8GB
镜像	公共镜像   openEuler   openEuler 20.03 64bit with ARM(40GB)
系统盘	通用型SSD   40GB

注意：这里“区域”的配置是和 VPC 的区域配置保持一致的。

步骤 4 配置完成后点击“下一步：网络配置”，进入网络配置，按下表配置网络参数。

参数	配置
网络	vpc-test   subnet-test   自动分配IP地址
安全组	sg-test
弹性公网IP	现在购买
线路	全动态BGP
公网带宽	按流量计费
带宽大小	5Mbit/s

步骤 5 配置完成后，点击“下一步：高级配置”，按下表配置 ECS 高级配置参数。

参数	配置
云服务器名称	openEuler（输入符合规则名称）
登录凭证	密码
密码	请输入8位以上包含大小写字母、数字和特殊字符的密码，如 openEuler@123
确认密码	请再次输入密码
云备份	暂不购买



云服务器组	不配置
高级选项	不勾选

步骤 6 配置完成后点击右下角“下一步：确认配置”。勾选同意协议，然后点击：立即购买。

步骤 7 在提交任务成功后，点击“返回云服务器列表”，返回 ECS 控制台。

### 1.3.3 通过 ssh 登录系统

步骤 1 在 ECS 控制台查看 ECS 弹性公网 IP 地址。

<input type="checkbox"/>	名称/ID	监控	可用区	状态	规格/镜像	IP地址	计费...
<input type="checkbox"/>	openEuler 5e4b28a...		可用区1	运行中	4vCPUs   8GB   kc1.xlarge.2 openEuler 20.03 64bit with ARM	121.36... 192.168...	按需计费 2020/11/15 ...

步骤 2 在客户端机器操作系统里的 Console 控制台或 Terminal 终端里运行 ssh 命令：

```
$ ssh root@121.36.45.64
```

（注意：此处的 IP 地址 121.36.45.64 即是刚刚购买的弹性公网 IP 地址。）

在客户端（本地 PC）第一次登录时会有安全性验证的提示：

```
The authenticity of host '119.8.238.181 (119.8.238.181)' can't be established.  
ECDSA key fingerprint is SHA256:RVxC1cSuMmqLtWdMw4n6f/VPsfWLkT/zDMT2q4qWxc0.  
Are you sure you want to continue connecting (yes/no)? yes
```

在这里输入 yes 并按回车键继续：

```
Warning: Permanently added '119.8.238.181' (ECDSA) to the list of known hosts.
```

```
Authorized users only. All activities may be monitored and reported.  
root@119.8.238.181's password:
```

输入密码(注意这里不会有任何回显)并回车，登录后的界面如下所示：

```
Welcome to Huawei Cloud Service
```

```
Last login: Mon May 18 15:35:37 2020
```

```
Welcome to Huawei Cloud Service
```

```
Last login: Mon May 18 15:35:37 2020
```

```
Welcome to 4.19.90-2003.4.0.0036.oe1.aarch64
```

```
System information as of time: Sun Nov 15 14:41:58 CST 2020
```

```
System load: 0.15
Processes: 131
Memory used: 5.0%
Swap used: 0.0%
Usage On: 9%
IP address: 192.168.1.5
Users online: 1
```

```
[root@openeuler ~]#
```

### 步骤 3 修改主机名

ECS 创建时被命名为“openEuler”，所以系统默认 hostname 为“openeuler”，为了和本实验手册另外两个版本保持行文上的一致，我们可以将主机名改为“openEuler”或“localhost”（一般在虚拟机中，主机名被默认为 localhost，而 ECS 也是虚拟机。本文的上下文环境中可能同时用到这三种名称，请鉴别）：

```
[root@openeuler ~]# vi /etc/hostname
```

```
[root@openeuler ~]# cat /etc/hostname
```

```
openEuler
```

```
[root@openeuler ~]# reboot
```

修改完成后重启系统并重新登录。

## 1.4 实验任务

### 1.4.1 openEuler 内核编译与安装

#### 步骤 1 安装工具，构建开发环境：

```
[root@openEuler ~]# yum group install -y "Development Tools" --nogpgcheck
```

```
[root@openEuler ~]# yum install -y bc --nogpgcheck
```

```
[root@openEuler ~]# yum install -y openssl-devel --nogpgcheck
```

#### 步骤 2 备份 boot 目录以防后续步骤更新内核失败

```
[root@openEuler ~]# tar czvf boot.origin.tgz /boot/
```

保存当前内核版本信息

```
[root@openEuler ~]# uname -r > uname_r.log
```

注意上述命令中的“-”为英文字符。

### 步骤 3 获取内核源代码并解压

```
[root@openEuler ~]# wget
https://gitee.com/openeuler/kernel/repository/archive/openEuler-20.03-LTS-SP3.zip
[root@openEuler ~]# unzip openEuler-20.03-LTS-SP3.zip
```

注意：您可能需要在 openEuler 代码仓（<https://gitee.com/openeuler/kernel>）获取到正确的内核代码 URL 地址并更新其源代码包的具体文件名称。

### 步骤 4 编译内核

```
[root@openEuler ~]# mv kernel-openEuler-20.03-LTS-SP3/ kernel
[root@openEuler ~]# cd kernel/
```

注意：解压后的 kernel 文件夹名称可能和 “*kernel*” 名称不一样，这里我们仅以 “*kernel*” 这个名称为例。

```
[root@openEuler kernel]# make openeuler_defconfig
```

在这里，我们按源代码文件 kernel/arch/arm64/configs/openeuler\_defconfig 的配置配置内核。

```
[root@openEuler kernel]# make help | grep Image
* Image.gz      - Compressed kernel image (arch/arm64/boot/Image.gz)
  Image         - Uncompressed kernel image (arch/arm64/boot/Image)
```

这一步查看了可编译的 Image。

```
[root@openEuler kernel]# make -j4 Image modules dtbs
```

这一步是编译内核的 Image、modules 和 dtbs。

### 步骤 5 安装内核

```
[root@openEuler kernel]# make modules_install
.....
INSTALL sound/soundcore.ko
DEPMOD  4.19.154
[root@openEuler kernel]# make install
/bin/sh ./arch/arm64/boot/install.sh 4.19.154 \
arch/arm64/boot/Image System.map "/boot"
dracut-install: Failed to find module 'xen-blkfront'
dracut: FAILED:  /usr/lib/dracut/dracut-install -D /var/tmp/dracut.tlIdPu/initramfs --kerneldir
/lib/modules/4.19.154/ -m virtio_gpu xen-blkfront xen-netfront virtio_blk virtio_scsi virtio_net virtio_pci
virtio_ring virtio
```

注意：在最后一步 “make install” 时出现的错误在这里可以忽略。

### 步骤 6 以 VNC 登录 ECS

<input type="checkbox"/>	名称/ID	监控	可用区	状态	规格/镜像
<input type="checkbox"/>	<a href="#">openEuler</a> 5e4b28a...				openEuler 5e4b28a8-e74c-4e62-8df5-f911704b7df1 2Us   8GB   kc1.xlarge.2 openEuler 20.03 64bit with ARM

在控制台 “弹性云服务器 ECS” 的页面中点击刚刚创建的虚拟机 “openEuler” 的名字超链接，在新打开的页面中点击 “远程登录” 按钮：



然后以控制台提供的 VNC 方式登录：



与以 ssh 登录一样，以 root 身份登录：

```
Authorized users only. All activities may be monitored and reported.
openEuler login: [ 429.483128] systemd-rc-local-generator[2892]: /etc/rc.d/rc.local is not marked executable, skipping.

openEuler login: root
Password:
Last login: Sun Nov 15 14:51:43 from 119.3.119.18

    Welcome to Huawei Cloud Service

Welcome to 4.19.90-2003.4.0.0036.oe1.aarch64

System information as of time: Sun Nov 15 15:51:58 CST 2020

System load:  0.00
Processes:    122
Memory used:  5.2%
Swap used:    0.0%
Usage On:     38%
IP address:   192.168.1.5
Users online: 2

[root@openEuler ~]# _
```

大部分的时间，我们仅将此作为一个监视器使用。

## 步骤 7 重启系统

在 ssh 终端重启操作系统：

```
[root@openEuler kernel]# reboot
```

## 步骤 8 登录并验证

在 VNC 窗口中选择以新编译出来的内核启动系统：

```
openEuler (4.19.154) 20.03 (LTS)
openEuler (4.19.90-2003.4.0.0036.oe1.aarch64) 20.03 (LTS)
openEuler (0-rescue-95148c976dae4cd0bfb15a0242465b8a) 20.03 (LTS)
System setup

Use the ▲ and ▼ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
```

在这里新编译出来的内核版本为 4.19.154。您的子版本号可能与此不一样。

#### 步骤 9 登录系统并查看版本

请以 VNC 和 ssh 终端登录系统，并在其中之一查看内核版本：

```
[root@openEuler ~]# uname -r
4.19.154
```

可以看出内核版本已更新。

## 1.4.2 Hello, world!

步骤 1 正确编写满足功能的源文件，包括.c 源文件和 Makefile 文件。在这里请参照示例源文件 hello\_world.c 及其 Makefile。

```
[root@openEuler ~]# cd hello-world
[root@openEuler hello-world]# ls
hello_world.c  Makefile
```

（您可以用 scp 命令将它们上传到 ECS。）

#### 步骤 2 编译源文件

```
[root@openEuler hello-world]# make
```

步骤 3 加载编译完成的内核模块，并查看加载结果。

```
[root@openEuler hello-world]# insmod hello_world.ko guy="Dinu" year=2013
[root@openEuler hello-world]# lsmod | grep hello_world
hello_world          262144  0
```



#### 步骤 4 卸载内核模块，并查看结果。

```
[root@openEuler hello-world]# rmmod hello_world
[root@openEuler hello-world]# dmesg | tail -n5
[ 55.787283] hello_world: loading out-of-tree module taints kernel.
[ 55.787798] hello_world: module verification failed: signature and/or required key missing - tainting kernel
[ 55.788937] Init module.
[ 55.789115] Hello, Dinu, 2013!
[ 55.789283] Exit module.
```

您在 VNC 窗口中也会看到同样的结果。

上文出现在命令行中的斜体字表示您需要根据自己的实际情况输入内容，出现在显示结果中的斜体字表示您的运行结果可能和在这里显示的不一样。

（提示：第一次装在 hello\_world 模块时，请忽略掉最开始安装模块时出现的两行错误提示信息，如上例所示。）

#### 步骤 5 在虚拟机和 VNC 窗口中退出登录

```
[root@openEuler ~]# exit
```

#### 步骤 6 关闭 VNC 客户端页面

## 1.5 其他内核编程实验

在这里可以继续进行其他内核实验。**实验完毕后需要将 ECS 关机**（此时仍有扣费，要完全不扣费就需要删除云资源），下次再用新的内核启动以继续完成实验。

这时，您也可以将 ECS 的规格改成较低的规格以节约费用：

计费模式	标签	操作
按需计费 2021/07/27 17:1...	--	远程登录   更多 ▾
按需计费 2021/05/30 19:3...	--	购买相同配置 开机 关机 重启 重置密码 变更规格 转包周期 删除 镜像/磁盘 ▶ 网络设置 ▶ 迁移云服务器

## 1.6 云环境资源清理

### 1.6.1 ECS 关机

当所实验完成后，应该对 ECS 进行关机以节约经费（ECS 关机后仍有少量扣费）。

步骤 1 回到 ECS 控制台，勾选 openEuler 虚拟机，进行关机。



在弹出的对话框中点击“是”按钮：



点击“是”按钮即可进行关机。

### 1.6.2 删除 ECS

可以等到所有的内核实验完成后再删除 ECS，否则每次都得重新编译内核。这里给出删除 ECS 的方法。

步骤 1 待关机完成后点击“更多”→“删除”



在弹出的对话框中勾选“释放云服务器绑定的弹性公网 IP 地址”和“删除云服务器挂载的数据盘”，然后点击“是”，删除 ECS。



步骤 2 您可以在控制台点击“更多 | 资源 | 我的资源”菜单项，检查资源是否全部删除



注意：(1) 虚拟私有云 VPC 和安全组可以不删除，以留下次使用。(2) 若在除“华北-北京四”之外区域购买了 ECS 和 EIP，请切换到那个区域查看。

# 2

## 实验二 内核模块编程

---

### 2.1 内核模块编程

#### 2.1.1 四个模块编程

##### 2.1.1.1 相关知识点及任务描述

内核模块是一种可以扩展运行时内核功能的目标文件( Object File )。大多数类 Unix 及 Windows 系统均使用模块, 这种机制使得允许内核在运行过程中动态地插入或者移除代码。

本实验要求编写四个模块, 分别实现以下功能:

1. 模块一, 加载和卸载模块时在系统日志输出信息。
2. 模块二, 支持整型、字符串、数组参数, 加载时读入并打印。
3. 模块三, 在 /proc 下创建只读文件。
4. 模块四, 在 /proc 下创建文件夹, 并创建一个可读可写的文件。

需要注意以下问题:

1. 模块可以写在一个 c 文件中, 模块参数传递参考宏定义 `module_param(name, type, perm)`, 需要用到头文件 `linux/moduleparam.h`。
2. 编写 Makefile 文件将 c 源码编译成 .ko 的模块。
3. 模块下 proc 目录和文件的创建参考 `proc_make()` 和 `proc_create` 函数。
4. 写入 proc 文件时, 可以考虑解决写缓冲溢出的问题( 可选 )。

##### 2.1.1.2 参考答案

参考 2-module-programming/exp1/task1 文件夹下源代码( 注意在 Makefile 中指定新编译的 kernel 源码路径 )。

参考 shell 脚本( 非 root 用户部分命令需用 sudo 执行 ) 以下步骤。

#### 步骤 1 编译源代码

```
make clean
make
```

#### 步骤 2 安装模块并测试



```
insmod module1.ko
insmod module2.ko int_var=666 str_var=hello int_array=10,20,30,40
insmod module3.ko
insmod module4.ko
lsmod | grep module
```

```
cat /proc/hello_proc
cat /proc/hello_dir/hello
echo nice > /proc/hello_dir/hello
cat /proc/hello_dir/hello
```

### 步骤 3 卸载模块

```
rmmod module4.ko
rmmod module3.ko
rmmod module2.ko
rmmod module1.ko
```

### 步骤 4 查看系统日志信息

```
clear
dmesg | tail -n80
```

## 2.2 思考题

Linux 内核模块的基本结构是什么？

# 3

## 实验三 内存管理

---

### 3.1 内存分配和管理之 kmalloc 和 vmalloc

#### 3.1.1 用 kmalloc 分配 1KB、8KB 的内存并打印指针地址

##### 3.1.1.1 相关知识点及任务描述

我们可以用 kmalloc() 函数分配内存，其特点：（1）申请的内存位于物理内存映射区域；（2）物理上连续，与真实的物理地址只有一个固定的偏移，存在较简单的转换关系；（3）最大不能超过 128KB。

本次实验完成如下任务：

1. 编写内核模块，调用 kmalloc() 函数，并分配内存，打印指针地址
2. 编写 makefile 文件，执行 make
3. 加载模块，查看加载的模块内容，即打印出来的指针地址
4. 根据机子是 32 位或者是 64 位的情况，分析地址落在的区域，并给出相应的解释

##### 3.1.1.2 参考答案

参考 3-memory/exp1/task1 文件夹下源代码。

要注意的是，在 Makefile 中需指定新编译的 kernel 源码路径，比如，如果您是在 /root/kernel 目录下编译的内核，则是“/root/kernel”。另外，编译、安装完新的内核后，在“/usr/lib/modules/”目录下会出现以您新内核版本号命名的目录，如“4.19.201”，您可以通过“uname -r”这个命令获取这个版本号，那么，在 Makefile 中，您可以以“/usr/lib/modules/\$(shell uname -r)”这种形式指定这个目录路径。若无特别说明，后文所有的实验都遵循这个规则。

参考命令（非 root 用户部分命令需用 sudo 执行，下同）：

```
# make
# insmod kmalloc.ko
# lsmod | grep kmalloc
kmalloc          262144  0
# rmmod kmalloc
# dmesg | tail -n6
[ 44.653995] kmalloc: loading out-of-tree module taints kernel.
[ 44.654464] kmalloc: module verification failed: signature and/or required key missing - tainting kernel
[ 44.655688] Start kmalloc!
[ 44.655879] kmallocmem1 addr = ffff800171e53000
[ 44.656181] kmallocmem2 addr = ffff8001704fe000
```

```
[ 106.904519] Exit kmalloc!  
# make clean
```

### 3.1.2 用 vmalloc 分配 8KB、1MB、64MB 的内存并打印指针地址

#### 3.1.2.1 相关知识点及任务描述

我们也可以用 vmalloc() 函数分配内存，其特点：（1）在虚拟内存空间给出一块连续的内存区（这片连续的虚拟内存存在物理内存中并不一定连续）；（2）由于其没有保证申请到的是连续的物理内存，因此对申请的内存大小没有限制。

本次实验完成如下任务：

1. 编写内核模块，调用 vmalloc() 函数，并分配内存，打印指针地址
2. 编写 makefile 文件，执行 make
3. 加载模块，查看加载的模块内容，即打印出来的指针地址
4. 根据机子是 32 位或者是 64 位的情况，分析地址落在的区域，并给出相应的解释

#### 3.1.2.2 参考答案

参考 01-memory/exp1/task2 文件夹下源代码（注意在 Makefile 中指定新编译的 kernel 源码路径）。

参考命令（非 root 用户部分命令需用 sudo 执行）：

```
# make  
# insmod vmalloc.ko  
# lsmod | grep vmalloc  
vmalloc          262144  0  
# rmmod vmalloc  
# dmesg | tail -n5  
[ 487.351051] Start vmalloc!  
[ 487.351292] vmallocmem1 addr = ffff00000b960000  
[ 487.351702] vmallocmem2 addr = ffff000020760000  
[ 487.352191] vmallocmem3 addr = ffff000023f60000  
[ 496.556492] Exit vmalloc!  
# make clean
```

### 3.1.3 使用首次适应算法实现一个简单的 malloc 内存分配器，满足内存分配和释放

#### 3.1.3.1 相关知识点及任务描述

首次适应算法在进行内存分配时，从空闲分区链首开始查找，直至找到一个能满足其大小需求的空闲分区，然后再按照作业的大小从该分区中划出一块内存分配给请求者，余下的空闲分区仍留在空闲分区链中。

本次实验完成如下任务：



模拟 Linux 伙伴系统的算法和机制，编写自己的内存管理小程序，要求设计实现一个简单的内存管理程序，能够在 256MB 的内存块内，支持 4KB 内存、1MB 内存、4MB 内存三类固定长度内存管理的频繁分配和回收。

### 3.1.3.2 参考答案

参考 01-memory/exp1/task3 文件夹下源代码。在此目录中实现了 emalloc 函数，在 test.c 文件中尝试使用该 emalloc 用于分配内存。

参考答案演示流程：

```
# make
# ./test
arrays [0][0] is OK.
.....
arrays [49][49] is OK.
# make clean
```

（使用./test 命令执行，查看运行结果。）

## 3.2 内存分配和管理之 iomap

### 3.2.1 申请、读写、释放 I/O 端口

#### 3.2.1.1 相关知识点及任务描述

I/O 接口电路需要设置专用寄存器缓冲输入输出数据、设定控制方式、保存输入输出状态信息等，这些寄存器可被 CPU 直接访问，称为端口。其可分为数据端口、状态端口和控制端口用以传输数据信息、状态信息和控制信息。我们可以用 request\_region()函数来申请一块输入输出区域。

本次实验完成如下任务：

1. 编写内核模块，调用内核相关接口，实现申请 I/O 端口、读写 I/O 端口、释放 I/O 端口；并打印输出相关信息。
2. 编写 makefile 文件，执行 make。
3. 加载模块，查看加载的模块内容。

#### 3.2.1.2 参考答案

参考 01-memory/exp2/task1 文件夹下源代码（注意在 Makefile 中指定新编译的 kernel 源码路径）。

```
# make
# insmod request_region.ko
# lsmod |grep request_region
request_region      262144  0
# rmmod request_region
```

```
# dmesg | tail -n4
[ 496.556492] Exit vmalloc!
[ 1578.353837] Start request region!
[ 1578.354111] it's ok for 22222.
[ 1646.519287] Exit request_region!
# make clean
```

## 3.2.2 申请、读写、释放 I/O 内存

### 3.2.2.1 相关知识点和任务描述

RISC 指令系统的 CPU 通常只实现一个物理地址空间（RAM），外设 I/O 端口的物理地址被映射到 CPU 的单一物理地址空间中而成为内存的一部分，从而形成统一编址。另外一些体系结构的 CPU（典型地如 x86）则为外设专门实现了一个单独地地址空间，称为“I/O 地址空间”或者“I/O 端口空间”。我们可以通过 request\_mem\_region()函数申请 I/O 内存。

本次实验完成如下任务：

1. 编写内核模块，调用内核相关接口，实现申请 I/O 内存、读写 I/O 内存、释放 I/O 内存；并打印输出相关信息。
2. 编写 makefile 文件，执行 make；
3. 加载模块，查看加载的模块内容。

### 3.2.2.2 参考答案

参考 01-memory/exp2/task2 文件夹下源代码（注意在 Makefile 中指定新编译的 kernel 源码路径）。

```
# make
# insmod request_mem_region.ko
# lsmod |grep request_mem_region
request_mem_region    262144  0
# rmmod request_mem_region
# dmesg | tail -n3
[ 1893.484863] Start request mem region!
[ 1893.485170] it's ok for 994115584.
[ 1914.441594] Exit request_region!
# make clean
```

## 3.3 思考题

kmalloc()和 vmalloc()有何不同？

# 4

## 实验四 中断和异常

---

### 4.1 使用 tasklet 完成一个中断程序

#### 4.1.1 使用 tasklet 打印第 31 号中断调用情况

##### 4.1.1.1 相关知识点和任务描述

在 ARM Linux 内核中，小任务分为高优先级的小任务和低优先级的小任务，它们是基于软中断实现的。本次实验需要完成如下任务：

1. 编写内核模块，对 31 号中断注册一个中断处理函数，打印出其调用的次数。
2. 把加载、卸载内核模块以 install/uninstall 写入 Makefile 文件中。

##### 4.1.1.2 参考答案

请参考 tasklet\_interrupt.c 及其 Makefile（在 4-interrupt/exp1/task1 中）。

```
# make
# insmod tasklet_interrupt.ko irq=31 devname="tasklet_dev"
# lsmod | grep tasklet_interrupt
tasklet_interrupt      262144  0
# rmmod tasklet_interrupt
# dmesg | tail -n5
[ 9709.588043] === Module starts...
[ 9709.588333] === req_ret is 0
[ 9709.588533] === tasklet_dev request IRQ:31 success...
[ 9782.983798] === Module exits...
[ 9782.984069] === tasklet_dev request IRQ:31 leaving success...
# make clean
```

### 4.2 工作队列

#### 4.2.1 使用工作队列打印当前日期

##### 4.2.1.1 相关知识点及任务描述

工作队列是实现延迟的新机制，从 2.5 版本 Linux 内核开始提供该功能。利用工作队列，完成如下任务：

1. 编写内核模块，分别发送一个实时任务（立即执行）和一个延迟任务(延后十秒)，观察它们的执行顺序。
2. 编写对应 Makefile 文件，并使用 make 编译上述内核模块。
3. 手动加载内核模块，查看加载内容。
4. 手动卸载上述内核模块。

#### 4.2.1.2 参考答案

请参考 workqueue\_test.c 及其 Makefile（在 4-interrupt/exp2/task1 中）。

```
# ls
Makefile  workqueue_test.c
# make
# insmod workqueue_test.ko times=5
# lsmod |grep workqueue_test
workqueue_test      262144  0
# rmmod workqueue_test
# dmesg | tail -n12
[12503.319760] 0:
[12503.319761] 2021-8-3  17:40:50
[12508.388953] 1:
[12508.388955] 2021-8-3  17:40:55
[12513.508898] this is a delay work : 2021-8-3  17:41:0
[12513.509737] 2:
[12513.509738] 2021-8-3  17:41:0
[12518.628862] 3:
[12518.628864] 2021-8-3  17:41:5
[12523.748841] 4:
[12523.748843] 2021-8-3  17:41:10
[12556.120925] unloading OK
# make clean
```

以上日期会有一个延迟时间打印出来。

### 4.3 思考题

响应中断的时机在哪里？

# 5

## 实验五 内核网络管理

---

### 5.1 利用 eBPF 机制实现简单的防火墙

#### 5.1.1 利用 XDP 编写 eBPF 过滤程序

##### 5.1.1.1 相关知识点及任务描述

Linux kernel 3.18 版本开始包含 eBPF，而 XDP 全称为 eXpress Data Path，是 Linux 内核网络栈的最底层。本次实验需要学习了解 XDP 与 eBPF，编写 eBPF 过滤程序、编译 eBPF 过滤程序，并将过滤程序应用到网卡中。

##### 5.1.1.2 参考答案

编写 eBPF 过滤程序

举例：禁止 eth0（这里的 eth0 根据自己的网口更改）上所有的数据包

```
// test.c
#include <linux/bpf.h>

#define __section(NAME) \
    __attribute__((section(NAME), used))

__section("prog")
int drop_all(struct xdp_md *ctx)
{
    return XDP_DROP;
}

char __license[] __section("license") = "GPL";
```

查看网口命名：lshw -C network



```
__section("prog")
```

```
int drop_tcp(struct xdp_md *ctx)
{
    int ipsize = 0;

    void *data = (void *) (long) ctx->data;
    struct iphdr *ip;

    ip = data + sizeof(struct ethhdr);

    if (ip->protocol == IPPROTO_TCP) {
        return XDP_DROP;
    }

    return XDP_PASS;
}

char _license[] __section("license") = "GPL";
```

注意：虚拟机上的设备可能无法支持 native 模式。比如在鲲鹏云弹性云服务器 ECS 上运行时会出现如下错误：

```
# ip link set dev eth0 xdp obj eth0.o
Error: virtio_net: Too few free TX rings available.
建议使用物理机。
```

## 5.2 思考题

Linux 高级网络栈架构通常包括哪些层？

# 6

## 实验六 文件系统

---

### 6.1 文件系统的简单实现

#### 6.1.1 参考 ramfs 文件系统，实现一个简单的内存文件系统

##### 6.1.1.1 相关知识点及任务描述

在这里，我们需要在 Linux 下实现一个内存文件系统，请完成以下任务：

1. 使用文件系统注册/注销函数，注册一个文件系统类型，名称为"mrfs"或其他自定义名称；
2. 此文件系统至少需要拥有以下功能：
  - (1) ls:查看当前目录下的文件和文件夹信息命令。
  - (2) cd:进入下级目录命令。
  - (3) mv:移动文件命令
  - (4) touch: 新建文件命令
  - (5) mkdir: 新建文件夹命令
  - (6) rm:删除文件命令
  - (7) rmdir:删除文件夹命令
  - (8) read: 从某文件内读取信息命令
  - (9) write: 向某文件内写入信息命令
  - (10) exit: 退出文件系统命令

##### 6.1.1.2 参考答案

参见 Disk.cpp、Disk.h、File.cpp、File.h、Makefile、my\_shell.cpp 文件。

```
# make
# ./my_shell
/ls
total:0
name    type    size    FCB      dataStartBlock
.....
/exit
# make clean
```

使用 make 进行编译，然后运行./my\_shell 进入文件系统，可以运行 ls、mkdir、cd、mv、touch、rmdir、my\_read、my\_write 等命令，最后 exit 退出。



## 6.2 思考题

虚拟文件系统（VFS）是什么、不是什么以及为什么是这样？

# 7

## 实验七 生产者消费者问题

---

### 7.1 实验目的

掌握基本的同步互斥算法，理解生产者消费者模型；了解 windows 或 openEuler 多线程的并发执行机制，线程间的同步与互斥；学习使用 windows 或 openEuler 中基本的同步对象，掌握相应的 API。

### 7.2 预习要求

已学习进程同步问题，对生产者-消费者问题有个基本的了解。

### 7.3 实验设备与环境

P4 以上电脑一台，已经安装 VC++、GCC 或其他 C 语言编译环境。

### 7.4 实验原理

在同一个进程地址空间内执行的两个线程。生产者线程生产物品，然后将物品放置在一个空缓冲区中供消费者线程消费。消费者线程从缓冲区中获得物品，然后释放缓冲区。当生产者线程生产物品时，如果没有空缓冲区可用，那么生产者线程必须等待消费者线程释放出一个空缓冲区。当消费者线程消费物品时，如果没有满的缓冲区，那么消费者线程将被阻塞，直到新的物品被生产出来。

## 7.5 实验任务

生产者/消费者模型为依据，在 Windows 环境下创建一个控制台进程，在该进程中创建  $n$  个线程模拟生产者和消费者，实现进程(线程)的同步与互斥；或进行 GUI 图形用户界面编程，展示上述进程(线程)创建及其同步与互斥的过程。

## 7.6 实验步骤和方法

供参考的部分核心代码：

1. 先初始化缓冲区长度为 6：

```
/*buffer.h*/  
  
typedef int buffer_item;  
  
#define BUFFER_SIZE 6
```

2. 创建三个信号量：mutex 信号量，作为互斥信号量，用于互斥的访问缓冲区；

full 信号量，判断缓冲区是否有值，初值为 0；

empty 信号量，判断缓冲区是否有空缓冲区，初值为缓冲区数。

3. 缓冲将会被用于两个函数：insert\_item()和 remove\_item()。
4. 编写两个函数：DWORD WINAPI producer(void \*param)和 DWORD WINAPI consumer(void \*param)，随机函数 rand()产生随机数。
5. 编写 main () 函数，主要功能是：

```
int main(int argc, char *argv[])  
  
{  
  
/*1. Get command line arguments argv[1], argv[2], argv[3]*/  
  
/*2. Initialize buffer*/  
  
/*3. Create producer threads(s)*/
```

```
/*4. Create consumer threads(s)*/
```

```
/*5. Sleep*/
```

```
/*6.Exit*/
```

```
}
```

6. 打印出相应结果。

# 8

## 实验八 进程调度

---

### 8.1 实验目的

进程调度是处理器管理的核心内容。本实验要求用高级语言编写和调试一个简单的进程调度程序，通过本实验加深对进程控制块、进程队列等概念的了解，掌握优先数调度算法和时间片调度算法的具体实施方法。

### 8.2 预习要求

完成进程管理理论课程的学习，掌握进程、进程调度的基本概念以及典型进程调度算法的基本思想。

### 8.3 实验设备与环境

PII 以上电脑一台，已经安装 VC++、GCC 或其他 C 语言编译环境

### 8.4 实验原理

操作系统是计算机系统中必不可少的系统软件。它是计算机系统中各种资源的管理者和各种活动的组织者、指挥者。进程调度解决了竞争处理器的问题。进程调度程序按照某种调度算法从就绪队列中选择一个进程，让它占用处理器。或者说，进程调度程序把处理器分配给了一个被选中的进程。所以，有时也把进程调度程序称为“处理器调度”程序。

在**优先数调度算法**方面：不同的系统确定优先数的方法可以不同，但一般都从任务的紧迫性和系统效率等方面考虑。例如，让系统进程的优先数大于用户进程的优先数，重要计算

问题的进程优先数大于一般计算问题的进程优先数，交互式作业进程的优先数大于批处理作业进程的优先数等。

在**时间片轮转调度算法**方面：时间片取值的大小关系到计算机系统的效率和用户的满意度，所以，时间片的值应根据进程要求系统给出应答时间和进入系统的进程数来决定。如果要求系统快速应答则时间片小一些，这样使轮转一遍的总时间减少而可对进程尽快回答。如果进程数少，则时间片可以大一些，这样可减少进程调度的次数，提高系统效率。对每个进程可规定相同的时间片，也可对不同的进程规定不同的时间片。

## 8.5 实验任务

设计实现一个控制台或 GUI 程序，包括两个调度算法模块，根据不同的调度算法模拟操作系统对进程的调度。

### 8.5.1 动态优先级调度

- 1、设计进程控制块 PBC 表结构，分别适用优先数调度算法
- 2、PBC 结构通常包括以下信息：进程名、进程优先数、轮转时间片、进程的 CPU 时间，进程状态等。根据调度算法不同，PCB 结构可作适当的调整。
- 3、建立进程队列。对不同的算法编制不同的入链程序。

程序要求达到的运行效果：在设置好进程数量、调度算法后，系统能按设定的参数运行，并在屏幕上交替显示就绪队列和完成队列的进程名等信息。

### 8.5.2 时间片轮转调度

- 1、设计进程控制块 PBC 表结构，适用循环时间片轮转算法。
- 2、PBC 结构通常包括以下信息：进程名、进程优先数、轮转时间片、进程的 CPU 时间，进程状态等。根据调度算法不同，PCB 结构可作适当的调整。
- 3、建立进程队列。对不同的算法编制不同的入链程序。

程序要求达到的运行效果：在设置好进程数量、调度算法后，系统能按设定的参数运行，并在屏幕上交替显示就绪队列和完成队列的进程名等信息。

## 8.6 实验步骤和方法

### 8.6.1 动态优先级调度算法

#### 1. 数据结构设计

PCB 结构:

name	进程名
pri /round	进程优先数/进程轮转时间片
cputime	进程占用的 CPU 时间
needtime	进程到完成还要的时间
state	进程状态 (假设状态为 Ready、Run、Finish)
next	链指针

#### 2. 算法设计

时间以时间片为计量单位。

- 1) 系统初始化时给每一个进程赋一个 NEEDTIME 和初始 PRI。并按优先数入队。
- 2) 系统每次选定一个优先级最高的进程投入运行, 进程每执行一次, 并将它的进程占用的 CPU 时间加 10, 进程到完成还要的 CPU 时间减 10。
- 3) 每当一个进程运行一个时间片后, 系统根据它的 CPUTIME 来判断它是否已经结束, 若 CPUTIME>0, 那么将它重新排入就绪队列。
- 4) 如果系统中尚有进程没有运行完毕, 那么转入 2) 。

### 8.6.2 时间片轮转调度算法

#### 1. 数据结构设计

PCB 结构:

name	进程名
round	进程轮转时间片
cputime	进程占用的 CPU 时间
needtime	进程到完成还要的时间

state

进程状态（假设状态为 Ready、Run、Finish）

next

链指针

## 2. 算法设计

时间以时间片为计量单位。

- 1) 系统初始化时给每一个进程赋以一个 needtime，并将所有进程按进入的次序排成一个队列。
- 2) 取队头进程，并投入运行。
- 3) 采用相对固定时间片（round），进程每执行一次，进程占用的 CPU 时间加 ROUND，进程到完成还要的 CPU 时间减 round。并排到就绪队列的尾部。
- 4) 如果当前进程的 needtime>0，那么将它排到队尾。
- 5) 如果尚有进程在队列中，那么转入 2)。