**《操作系统》实验报告**

徽标, 公司名称

描述已自动生成

**题 目 进程管理**

**学 院 计算机学院**

**班 级 2021211304**

**学 号 2021212484**

**姓 名 张梓良**

**2024年 5 月 5 日**

目录

[1. 编写内核模块，打印输出当前CPU负载 1](#_Toc166330336)

[1.1 实验环境 1](#_Toc166330337)

[1.2 实验内容及预备知识 1](#_Toc166330338)

[1.1.1 实验内容 1](#_Toc166330339)

[1.1.2 预备知识 1](#_Toc166330340)

[1.3 实验代码 1](#_Toc166330341)

[1.4 实验结果 3](#_Toc166330342)

[2. 实验5-3-4 使用cgroup实现限制CPU核数 3](#_Toc166330343)

[2.1 实验环境 3](#_Toc166330344)

[2.2 实验内容及预备知识 3](#_Toc166330345)

[2.2.1 实验内容 3](#_Toc166330346)

[2.2.2 预备知识 3](#_Toc166330347)

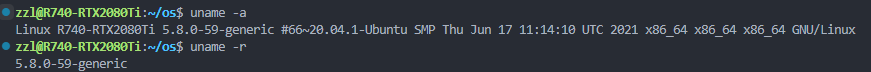
[2.3 实验步骤 4](#_Toc166330348)

[3. 总结 5](#_Toc166330349)

# 编写内核模块，打印输出当前CPU负载

# 实验环境

* Linux R740-RTX2080Ti 5.8.0-59-generic #66~20.04.1-Ubuntu SMP Thu Jun 17 11:14:10 UTC 2021 x86\_64 x86\_64 x86\_64 GNU/Linux
* 5.8.0-59-generic



# 实验内容及预备知识

## 实验内容

1. 编写一个内核模块，实现读取系统一分钟内的CPU负载。
2. 加载、卸载模块并查看模块打印信息。

## 预备知识

proc文件系统能够保存系统当前工作的特殊数据，但并不存在于任何物理设备中，对其进行读写时，才根据系统中的相关信息即时生成。所有proc文件挂载在/proc目录下。

而我们的目的是查看系统CPU的负载，需要使用的文件为 /proc/loadavg 。

内核中读写文件数据的方法：

•filp\_open() 函数原型：struct file\* filp\_open(const char\* filename, int open\_mode, int mode);

•filp\_close() 函数原型：int filp\_close(struct file\*filp, fl\_owner\_t id);

•kernel\_read() 函数原型：ssize\_t kernel\_read(struct file file, void buf, size\_t count, loff\_t \*pos);

•kernel\_write() 函数原型：ssize\_t kernel\_write(struct file file, const void buf, size\_t count, loff\_t \*pos);

我们需要打开文件/proc/loadavg 所以使用函数 filp\_open() 打开文件，使用kernel\_read() 读取打开的文件，在结束时调用 filp\_close() 关闭文件。

# 实验代码

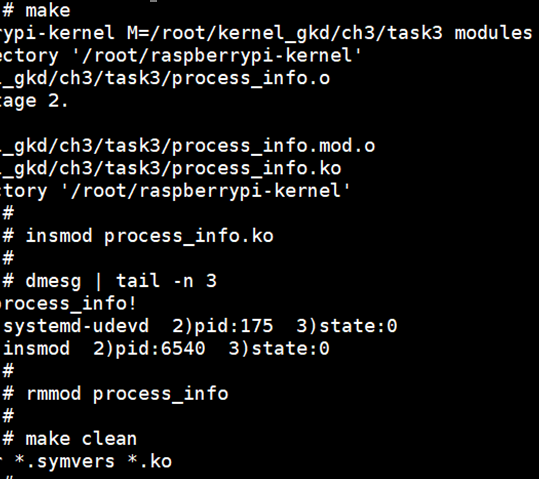
编写cpu\_loadavg.c:

1. #include <linux/module.h>
2. #include <linux/fs.h>
3. MODULE\_LICENSE("GPL");
4. char tmp\_cpu\_load[5] = {'\0'};
5. static int get\_loadavg(void)
6. {
7. struct file \*fp\_cpu;
8. loff\_t pos = 0;
9. char buf\_cpu[10];
10. fp\_cpu = filp\_open("/proc/loadavg", O\_RDONLY, 0);
11. if (IS\_ERR(fp\_cpu)){
12. printk("Failed to open loadavg file!\n");
13. return -1;
14. }
15. kernel\_read(fp\_cpu, buf\_cpu, sizeof(buf\_cpu), &pos);
16. strncpy(tmp\_cpu\_load, buf\_cpu, 4);
17. filp\_close(fp\_cpu, NULL);
18. return 0;
19. }
20. static int \_\_init cpu\_loadavg\_init(void)
21. {
22. printk("Start cpu\_loadavg!\n");
23. if(0 != get\_loadavg()){
24. printk("Failed to read loadarvg file!\n");
25. return -1;
26. }
27. printk("The cpu loadavg in one minute is: %s\n", tmp\_cpu\_load);
28. return 0;
29. }
30. static void \_\_exit cpu\_loadavg\_exit(void)
31. {
32. printk("Exit cpu\_loadavg!\n");
33. }
34. module\_init(cpu\_loadavg\_init);
35. module\_exit(cpu\_loadavg\_exit);

编写makefile:

1. ifneq ($(KERNELRELEASE),)
2. obj-m := cpu\_loadavg.o
3. else
4. KERNELDIR ?= /root/kernel
5. PWD := $(shell pwd)
6. default:
7. $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
8. endif
9. .PHONY:clean
10. clean:
11. -rm \*.mod.c \*.o \*.order \*.symvers \*.ko

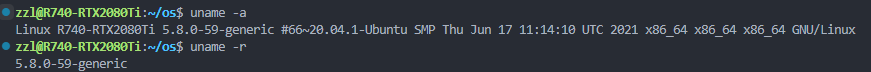
# 实验结果



# 实验5-3-4 使用cgroup实现限制CPU核数

# 实验环境

* Linux R740-RTX2080Ti 5.8.0-59-generic #66~20.04.1-Ubuntu SMP Thu Jun 17 11:14:10 UTC 2021 x86\_64 x86\_64 x86\_64 GNU/Linux
* 5.8.0-59-generic



# 实验内容及预备知识

## 实验内容

1. 使用 cgroup 实现限制 CPU 核数；
2. 编写一个简单的c源程序，实现无线循环/死循环，使其占用某一进程（默认情况下会使得 cpu 资源消耗在 100%）；
3. 使用 cgexec 与 taskset 测试上述限制操作是否成功。

## 预备知识

cgroup（Control Groups）是将任意进程进行分组化管理的 Linux 内核功能，提供将进程进行分组化管理的功能和接口的基础结构。I/O 或内存的分配控制等具体的资源管理功能是通过这个功能来实现的，这些具体的资源管理功能称为 cgroup子系统或控制器。

cgroup的机制是：它以分组的形式对进程使用系统资源的行为进行管理和控制。也就是说，用户通过cgroup对所有进程进行分组，再对该分组整体进行资源的分配和控制。

cgroup中的每个分组称为进程组，它包含多个进程。最初情况下，系统内的所有进程形成一个进程组（根进程组），根据系统对资源的需求，这个根进程组将被进一步细分为子进程组，子进程组内的进程是根进程组内进程的子集。而这些子进程组很有可能继续被进一步细分，最终，系统内所有的进程组形成一颗具有层次等级（hierarchy）关系的进程组树。

如果某个进程组内的进程创建了子进程，那么该子进程默认与父进程处于同一进程组中。也就是说，cgroup对该进程组的资源控制同样作用于子进程。

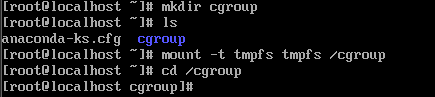
比如，我们限制进程的CPU使用为20%，我们就可以建一个cpu占用为20% 的cgroup，然后将进程添加到这个cgroup中。当然，一个cgroup可以有多个进程。

cgroup提供了一个 cgroup虚拟文件系统，作为进行分组管理和各子系统设置的用户接口。要使用 cgroup，必须挂载 cgroup文件系统。这时通过挂载选项指定使用哪个子系统。

# 实验步骤

**挂载tmpfs格式的cgroup文件夹**

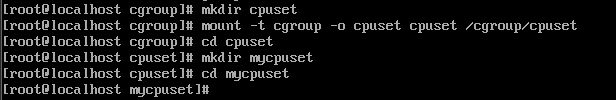
在root权限在执行以下命令：



挂载tmpfs文件类型：tmpfs是直接建立在VM之上的，用一个简单的mount命令就可以创建tmpfs文件系统了。速度快，可以动态分配文件系统大小。

**挂载cpuset管理子系统**

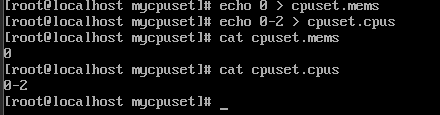
挂载某一个 cgroups 子系统到挂载点之后，就可以通过在挂载点下面建立文件夹或者使用cgcreate命令的方法创建 cgroups 层级结构中的节点/控制组；对应的删除则使用 rmdir删除文件夹，或使用cgdelete命令删除。

****

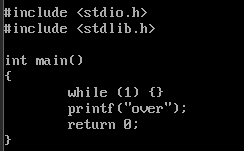
**设置cpu核数**

echo 0 > cpuset.mems 设置0号内存结点。mems默认为空，因此需要填入值。

echo 0-2 > cpuset.cpus 这里的0-2指的是使用cpu的0、1、2三个核。实现了只是用这三个核。

****

**简单的死循环C源文件while\_long.c（需要退出当前目录编写）**

****

**测试验证**

****

显示为7（111），证明测试限制cpu核数成功。

# 总结

在这个项目中，我首先编写了一个Linux内核模块来读取系统一分钟内的CPU负载，并通过加载和卸载模块来检查其输出。接着，我使用cgroup限制了CPU核数，并通过编写一个简单的C程序来模拟高CPU负载情况，验证了这些限制。通过使用cgexec和taskset命令，我确保了设置的CPU限制得到了执行。通过这些步骤，我加深了对Linux系统资源管理和内核行为的理解。