openEuler内核编程技术

实训指导书

第四章 第3讲

《cgroup机制》

软件所制

**术语表**

|  |  |  |  |
| --- | --- | --- | --- |
| 英文 | 中文 | 含义 | 别名 |
|  |  |  |  |
|  |  |  |  |

第四章 第3讲 cgroup机制

# 任务1：使用 cgroup 实现限制 CPU 核数（45min）

## 相关知识

**一、cgroup介绍**

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。

1、相关术语说明：

（1）任务：在cgroups中，任务就是系统的一个进程。

（2）控制族群（control group）：控制族群就是一组按照某种标准划分的进程。cgroups中的资源控制都是以控制族群为单位实现。一个进程可以加入到某个控制族群，也从一个进程组迁移到另一个控制族群。一个进程组的进程可以使用cgroups以控制族群为单位分配的资源，同时受到cgroups以控制族群为单位设定的限制。

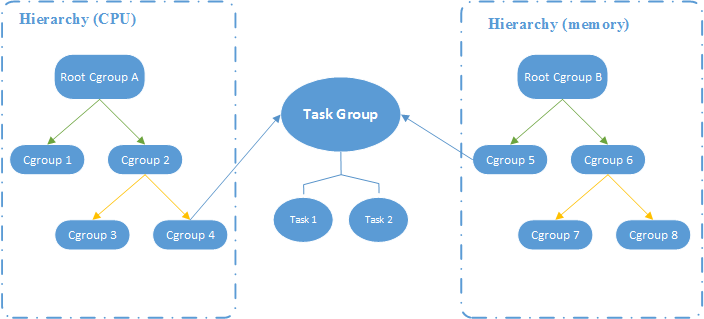
（3）层次等级/层级（hierarchy）：它是cgroup的集合，可以把它理解成cgroup的树，cgroup是根进程组的结点。

。控制族群可以组织成hierarchical的形式，即一颗控制族群树。控制族群树上的子节点控制族群是父节点控制族群的孩子，继承父控制族群的特定的属性。

（5）子系统（subsytem）：一个子系统就是一个资源控制器。

比如cpu子系统就是控制cpu时间分配的一个控制器。子系统必须附加（attach）到一个层级上才能起作用，一个子系统附加到某个层级以后，这个层级上的所有控制族群都受到这个子系统的控制。比如上面的CPU占用率就是一个子系统。简而言之，子系统就是cgroup中可添加删除的模块。在cgroup架构的封装下为cgroup提供多种行为控制。

CGroup 层级图



上图显示：CPU 和 Memory 两个子系统有自己独立的层级系统，而又通过 Task Group 取得关联关系。

2、cgroup为每种可以控制的资源定义了一个子系统。典型的子系统介绍如下：

1）cpu子系统：该子系统为每个进程组设置一个使用CPU的权重值，以此来管理进程对cpu的访问，限制进程的 cpu 使用率。

2）cpuset子系统：对于多核cpu，该子系统可以设置进程组只能在指定的核上运行，并且还可以设置进程组在指定的内存节点上申请内存。如果要使用 cpuset 控制器，需要同时配置 cpuset.cpus 和 cpuset.mems 两个文件（参数）。cpuset.mems用来设置内存节点的；cpuset.cpus用来限制进程可以使用的 cpu 核心；这两个参数中 cpu 核心、内存节点都用 id 表示，之间用 “,” 分隔，比如 0,1,2 ；也可以用 “-” 表示范围，如 0-3 ；两者可以结合起来用。如“0-2,6,7”。在添加进程前，cpuset.cpus、cpuset.mems 必须同时设置，而且必须是兼容的，否则会出错。

3）cpuacct子系统：该子系统只用于生成当前进程组内的进程对cpu的使用报告。

4）memory子系统：该子系统提供了以页面为单位对内存的访问，比如对进程组设置内存使用上限等，同时可以生成内存资源报告。

5）blkio子系统：该子系统用于限制每个块设备的输入输出（比如物理设备（磁盘，固态硬盘，USB 等等）。首先，与CPU子系统类似，该系统通过为每个进程组设置权重来控制块设备对其的I/O时间；其次，该子系统也可以限制进程组的I/O带宽以及IOPS。

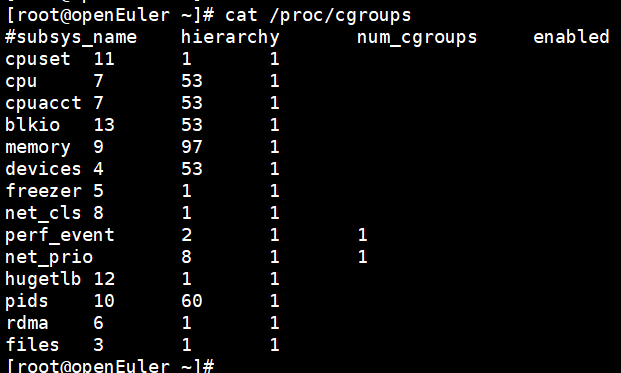
6）devices子系统：通过该子系统可以限制进程组对设备的访问，即该允许或禁止进程组对某设备的访问。

7）freezer子系统：该子系统可以使得进程组中的所有进程挂起或恢复。

8）net-cls子系统：该子系统使用等级识别符标记网络数据包，可允许 Linux 流量控制程序识别从具体 cgroup 中生成的数据包，提供对网络带宽的访问限制，比如对发送带宽和接收带宽进程限制。

9）ns子系统：名称空间子系统，可以使不同 cgroups 下面的进程使用不同的 namespace。

针对运行中的内核而言，可以使用的 cgroup 子系统由 /proc/cgroup 来确认。



3、cgroups可以做什么？

cgroups最初的目标是为资源管理提供的一个统一的框架，既整合现有的cpuset等子系统，也为未来开发新的子系统提供接口。现在的cgroups适用于多种应用场景，从单个进程的资源控制，到实现操作系统层次的虚拟化。

cgroups提供了以下功能：

* 限制进程组可以使用的资源数量。  
  比如：memory子系统可以为进程组设定一个memory使用上限，一旦进程组使用的内存达到限额再申请内存，就会出发OOM。
* 进程组的优先级控制。  
  比如：可以使用cpu子系统为某个进程组分配特定cpu share。
* 记录进程组使用的资源数量。  
  比如：可以使用cpuacct子系统记录某个进程组使用的cpu时间。
* 进程组隔离。  
  比如：使用ns子系统可以使不同的进程组使用不同的namespace，以达到隔离的目的，不同的进程组有各自的进程、网络、文件系统挂载空间。
* 进程组控制。  
  比如：使用freezer子系统可以将进程组挂起和恢复。

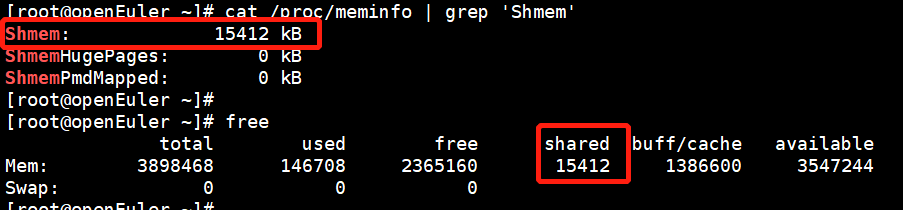
二、tmpfs文件系统

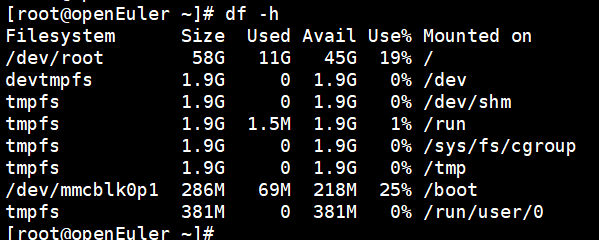
tmpfs即临时文件系统，是一种基于内存的文件系统，也称之为虚拟内存文档系统。它不同于传统的用块设备形式来实现的ramdisk，也不同于针对物理内存的ramfs。tmpfs能够使用物理内存，也能够使用交换分区。

在Linux内核中，虚拟内存资源由物理内存（RAM）和交换分区swap组成，这些资源是由内核中的虚拟内存子系统来负责分配和管理。tmpfs就是和虚拟内存子系统来"打交道"的，他向虚拟内存子系统请求页来存储文档，他同Linux的其他请求页的部分相同，不知道分配给自己的页是在内存中还是在交换分区中。tmpfs同ramfs相同，其大小也不是固定的，而是随着所需要的空间而动态的增减。

所有在 tmpfs 上储存的资料在理论上都是临时存放的，也就是说，档案不会建立在硬盘上面。一旦重新开机，所有在 tmpfs 里面的资料都会消失不见。理论上，内存使用量会随着 tmpfs 的使用而时有增长或消减。tmpfs将所有内容放入内核内部高速缓存中，并进行扩展和收缩以容纳其中包含的文件，并且能够将不需要的页面交换出来以交换空间。它具有最大大小限制，可以通过“ mount -o remount ...”即时调整。

由于tmpfs完全存在于页面缓存和交换中，因此所有tmpfs页面将在 /proc/meminfo中显示为 “Shmem”，在free命令后中显示为“Shared”。请注意，这些计数器还包括共享内存（shmem，请参阅ipcs（1））。获取计数的最可靠方法是使用df和du。





使用tmpfs，首先您编译内核时得选择"虚拟内存文档系统支持（Virtual memory filesystem support）" 或设置CONFIG\_TMPFS=y，然后就能够加载tmpfs文档系统了。

详细的介绍，可参见内核源码中的官方文档：Documentation/filesystems/tmpfs.txt。

可挂载 tmpfs 格式的 cgroup 文件夹进行 cgroup 的相关操作。

三、配置环境（树莓派）

1、安装libcgroup：dnf install libcgroup -y

libcgroup包含cgroup用户空间工具套件（如lscgroup，lssubsys等）以及静态或者动态库，以供其他程序调用，并且包含debug套件。

三、mount命令

cgroup在Linux内核中是以文件系统的形式存在的，不过cgroup对应的这种文件系统与proc文件系统类似，都是只存在于内存中的“虚拟”文件系统。既然如此，就可以通过mount命令创建一个cgroup实例。

命令语法：mount [-afFhnrvVw] [-L<标签>] [-o<选项>] [-t<文件系统类型>] [设备名] [加载点]

命令功能：加载指定的文件系统。

参数：

-o options 主要用来描述设备或档案的挂接方式。这里则是子系统，如memory、cpuset等。

-t type 用于指定或限制文件系统类型，如：-t cgroup表明是cgroup类型的挂载。

例如：

sudo mount -t cgroup -o memory memory\_cgroup /dev/cgroup/

说明：上述命令在/dev/cgroup/下创建了一个memory子系统，-t cgroup表明是cgroup类型的挂载，-o memory表明是memory子系统，memory\_cgroup就是设备名，/dev/cgroup/是加载路径。

四、echo命令

命令语法：echo [-ne][字符串] 或 echo [--help][--version]

命令功能：显示文字。echo会将输入的字符串送往标准输出。输出的字符串间以空白字符隔开, 并在最后加上换行号。

参数：

-n 不要在最后自动换行

-e 激活转义字符。若字符串中出现特殊字符，则特别加以处理，而不会将它当成一般文字输出（如：\n 换行且光标移至行首；\t 插入tab）。

五、taskset 命令（参考：<https://blog.csdn.net/test1280/article/details/87991302> ）

命令语法：taskset [options] -p [mask] pid

命令功能：依据线程PID（TID）查询或设置线程的CPU亲和性（即与哪个CPU核心绑定）。

参数：

-p, --pid 查询/指定PID的进程/线程可用的CPU核心，默认以掩码形式表示

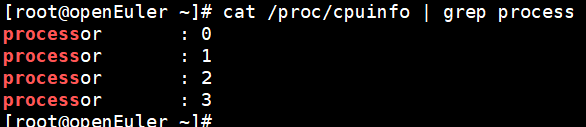
-a, --all-tasks 操作所有的任务线程

-c, --cpu-list 通过列表显示方式设置CPU（逗号相隔）

说明：

设置绑定CPU核心有两种方法：掩码形式、列表形式。

mask指的是掩码，将掩码转换为二进制形式，从最低位到最高位代表物理CPU的#0、#1、……、#n号核。某位的值为0表示不绑该核，1表示绑。当前系统有几个CPU核心以及对应的核心编号，可通过以下方式查看：



使用实例：

（1）查询 PID= 11698 的线程可用的CPU核心，掩码形式表示

|  |
| --- |
| # taskset -p 11698  pid 11698's current affinity mask: f |

说明：f => 0x0F（十六进制） => 1111（二进制），从最右起第一个1表示CPU核心#0，第二个1表示CPU核心#1，也就是说，线程11698可使用0-3号CPU核心。

（2）查询 PID（TID）= 11698 的线程可用的CPU核心：

|  |
| --- |
| # taskset -cp 11698  pid 11698's current affinity list: 0-3 |

说明：程11698可使用0-3号CPU核心。

（3）设置 PID（TID）= 11698 的线程可用 #0、#2号CPU核心：

|  |
| --- |
| # taskset -pc 0,2 11698  pid 11698's current affinity list: 0-3  pid 11698's new affinity list: 0,2 |

说明：原来是可用0-3号CPU核心；设置后是0,2。

（4）设置 PID（TID）= 11698 的线程可用 #0、#2、#3号CPU核心，掩码形式表示：

|  |
| --- |
| # taskset -p 7 11698  pid 11698's current affinity mask: 5  pid 11698's new affinity mask: 7 |

说明：原来的是0,2，即0101（掩码十进制5）；现在的是0111，即掩码十进制7。

6、cgexec 命令

命令功能：在指定的cgroup中运行任务。

安装命令：dnf install -y libcgroup

命令语法：cgexec [-h] [-g <子系统>:<路径>] [--sticky] 命令名 [命令参数]

参数：

-g 决定了将会运行哪一个cgroup系统。

-h help 帮助信息

## 任务描述

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。

## 审核要求

1. 正确、成功地限制CPU核数。
2. 提交每一步操作以及结果显示的截图。

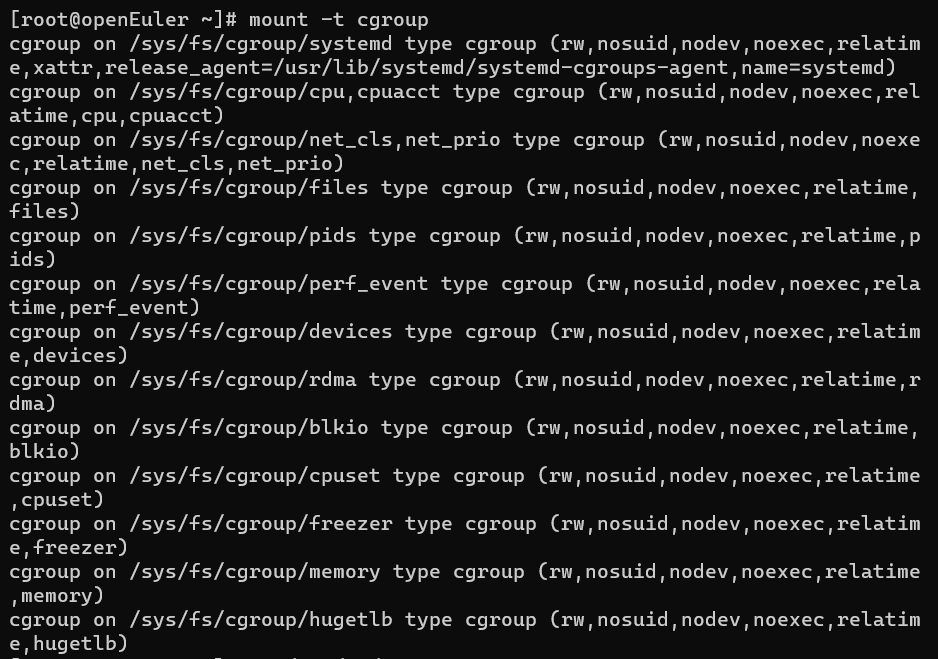
## 参考答案

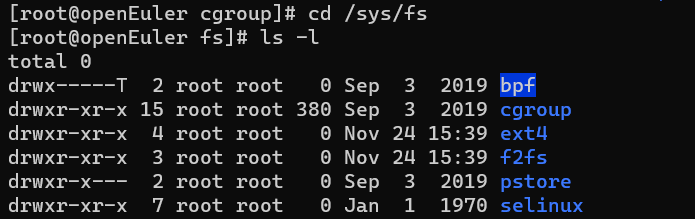
**一、操作步骤**

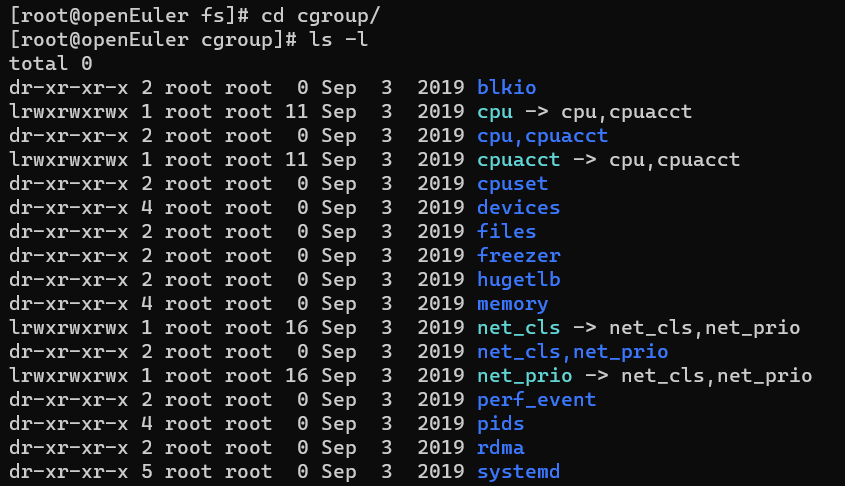
1、Linux把cgroup实现成了一个file system，在root权限在执行以下命令：

|  |
| --- |
| # mount -t cgroup  # cd /sys/fs/  # ls -l  # cd cgroup  # ls -l |

在/sys/fs下有一个cgroup的目录，这个目录下还有很多子目录，比如： cpu、cpuset、memory、blkio……，这些都是cgroup的子系统。



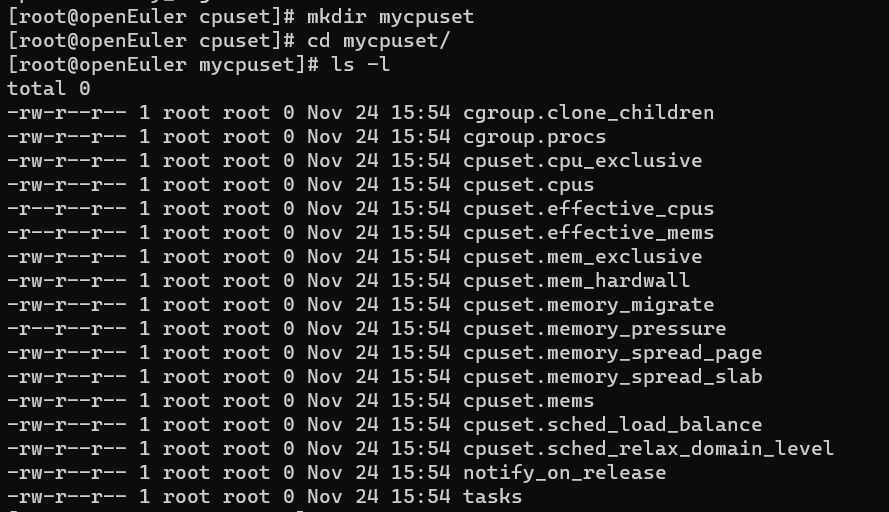




2、挂载cpuset管理子系统

cgroup挂载成功后，就可以通过在cgroup的各个子目录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。你会发现，一旦你创建了一个子目录，这个子目录里又有很多文件了。

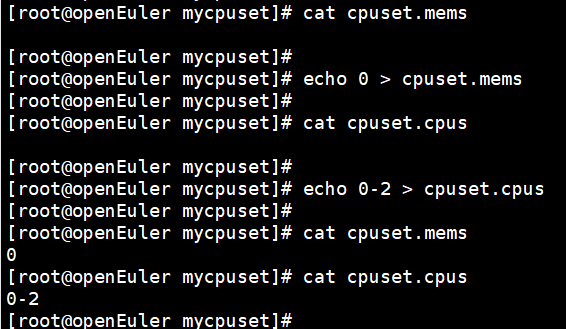
|  |
| --- |
| # cd cpuset  # mkdir mycpuset #创建一个控制组，删除用 rmdir 命令  # cd mycpuset  # ls -l |



3、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。



4、简单的死循环C源文件while\_long.c

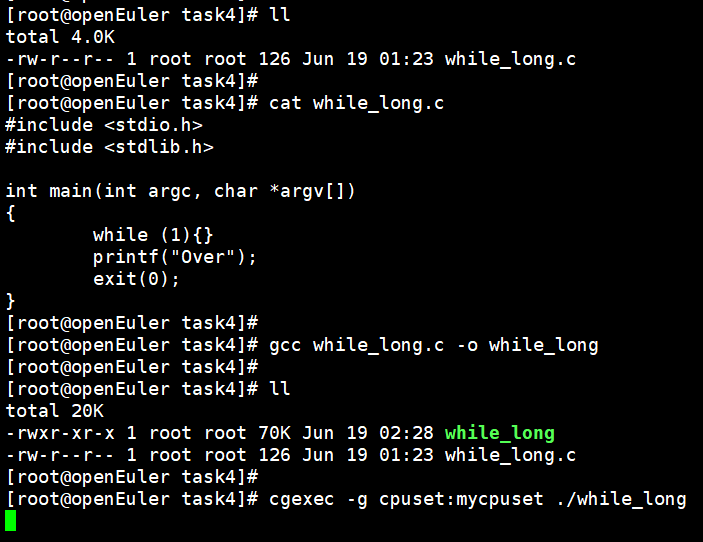
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main(int argc, char \*argv[])  {  while (1){}  printf("Over");  exit(0);  } |

5、测试验证

（1）打开一个终端，执行以下命令：

|  |
| --- |
| ## 编译上述C源文件  # gcc while\_long.c -o while\_long  ## 指定在cpuset子系统的mycpuset控制组中运行while\_long，使用 ctrl+c 退出运行  # cgexec -g cpuset:mycpuset ./while\_long |

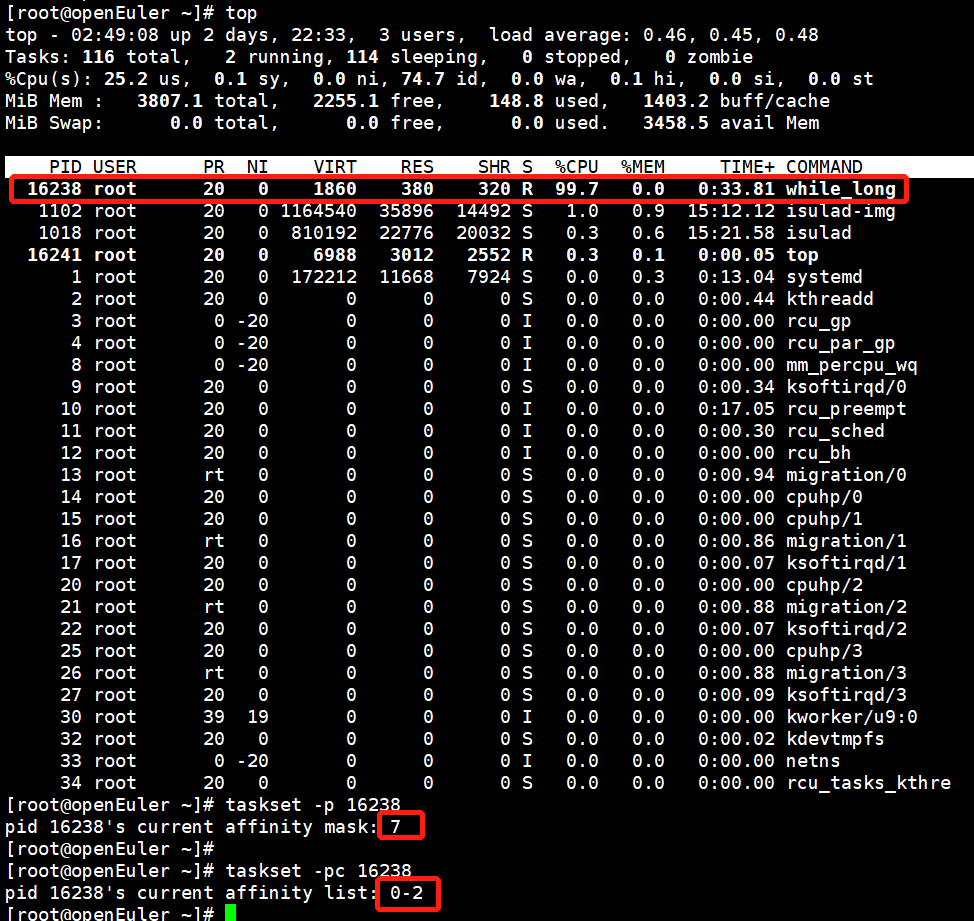
执行如下：



（2）不要关闭上述终端，另打开一个终端，执行以下命令：

|  |
| --- |
| # top //查看程序while\_long的PID, 假设为16238。输入q 退出当前查看状态  # taskset -p 16238 //显示的如果是7（111），则测试限制cpu核数成功。 |

执行如下：



# 任务2：使用 cgroup 实现不允许访问U盘（45min）

## 相关知识

**同1.1。**

1、fdisk 命令

命令语法：fdisk [必要参数][选择参数]

命令功能：创建和维护分区表的程序。

参数：

（必要参数）：

-l 列出素所有分区表

-u 与"-l"搭配使用，显示分区数目

（选择参数）：

m ：显示菜单和帮助信息

a ：活动分区标记/引导分区

d ：删除分区

l ：显示分区类型

n ：新建分区

p ：显示分区信息

q ：退出不保存

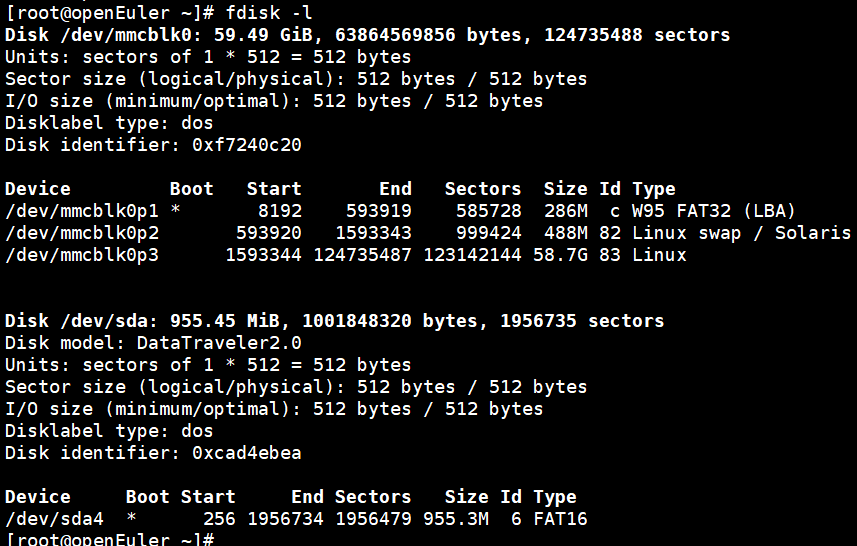
t ：设置分区号

v ：进行分区检查

w ：保存修改

x ：扩展应用，高级功能

例如：常用fdisk -l 显示当前分区情况



2、ls -l ：显示文件及其详细信息。

例：

|  |
| --- |
| # ls –l  total 5  -rw-r--r-- 1 root root 1668 May 3 2020 anaconda-ks.cfg  drwxr-xr-x 2 root root 4096 May 6 00:04 aa |

显示的文件详细信息分别代表什么呢？以上面蓝色部分为例。

1. total 5 代表当前目录下文件大小的总和为5K（每个目录的大小都按4K算）
2. drwxr-xr-x  
   第一个字符有4种情况：“-”表示普通文件，“d”代表目录，“l”代表连接文件，“b”代表设备文件。后面的9个字符每3个为一组，分别代表：文件所有者、文件所有者所在用户组、其它用户对文件拥有的权限。每组中3个字符分别代表读、写、执行的权限，若没有其中的任何一个权限则用“-”表示。执行的权限有两个字符可选“x”代表可执行，“s”代表套接口文件。
3. 紧接着的数字2代表 “aa”这个目录下的目录文件数目（这个数目=隐藏目录数目+普通目录数目）。我们进入“aa”目录用命令 ls -al （为了看到隐藏文件我们加上-a这个参数）

|  |
| --- |
| # ls -al  total 8  drwxr-xr-x 2 root root 4096 May 6 00:04 .  drwxr-x--- 14 root root 4096 May 6 00:04 .. |

（上面的第3行中的2代表当前目录中有子目录2个，即.和..

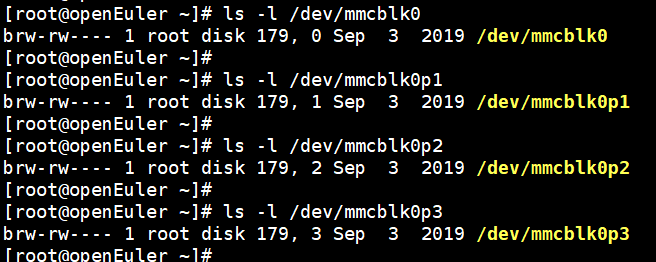
上面的第4行中的14代表这个目录的上一层目录中有14个子目录。）

1. 再接下来的root代表这个文件（目录）的属主为 用户root。
2. 再接下来的root代表这个文件（目录）所属的用户组为 组root
3. 4096 代表文件的大小（字节数），目录的大小总是为4096字节。
4. May 6 00:04 代表文件（目录）的修改时间。
5. aa代表文件（目录）在名字。

文件名颜色的含义：

* 默认色代表普通文件。 例：install.log
* 绿色代表可执行文件。 例：rc.news
* 红色代表tar包文件。 例：vim-7.1.tar.bz2
* 蓝色代表目录文件。 例：aa
* 水红代表图象文件。 例：Sunset.jpg
* 青色代表链接文件。 例：rc4.d （此类文件相当于快捷方式）
* 黄色代表设备文件。 例：fd0

注：设备信息的查看



其中可以知道设备号分别为：179,0 179,1 179,2 179,3 。

3、dd命令

命令语法：dd [option]

命令功能：转换和复制文件。

参数：

if =输入文件（或设备名称）。

of =输出文件（或设备名称）。

ibs = bytes 一次读取bytes字节，即读入缓冲区的字节数。

skip = blocks 跳过读入缓冲区开头的ibs\*blocks块。

obs = bytes 一次写入bytes字节，即写入缓冲区的字节数。

bs = bytes 同时设置读/写缓冲区的字节数（等于设置ibs和obs）。

cbs = byte 一次转换bytes字节。

count=blocks 只拷贝输入的blocks块。

conv = ASCII 把EBCDIC码转换为ASCIl码。

conv = ebcdic 把ASCIl码转换为EBCDIC码。

conv = ibm 把ASCIl码转换为alternate EBCDIC码。

conv = block 把变动位转换成固定字符。

conv = ublock 把固定位转换成变动位。

conv = ucase 把字母由小写转换为大写。

conv = lcase 把字母由大写转换为小写。

conv = notrunc 不截短输出文件。

conv = swab 交换每一对输入字节。

conv = noerror 出错时不停止处理。

conv = sync 把每个输入记录的大小都调到ibs的大小（用NUL填充）。

4、devices子系统

使用devices 子系统可以允许或者拒绝cgroup中的进程访问设备。

devices子系统有三个控制文件：devices.allow，devices.deny，devices.list。

* devices.allow用于指定cgroup中的进程可以访问的设备；
* devices.deny用于指定cgroup中的进程不能访问的设备；
* devices.list用于报告cgroup中的进程访问的设备。

devices.allow文件中包含若干条目，每个条目有四个字段：type、major、minor 和 access。type、major 和 minor 字段中使用的值对应 Linux 分配的设备。

（1）type指定设备类型：

a -------- 应用所有设备，可以是字符设备，也可以是块设备

b -------- 指定块设备

c -------- 指定字符设备

（2）major和minor指定设备的主次设备号。如下图：8为主设备号，4为次设备号。



（3）access 则指定相应的权限：

r -------- 允许任务从指定设备中读取

w ------- 允许任务写入指定设备

m ------- 允许任务生成还不存在的设备文件

写入示例：

|  |
| --- |
| # echo 'a,8,4,rwm' > /cgroup/devices/mydevices/devices.deny  # echo 'a 8 4 rwm' > /cgroup/devices/mydevices/devices.deny  # echo 'a 8:4 rwm' > /cgroup/devices/mydevices/devices.deny |

## 任务描述

1. 使用cgroup实现不允许访问U盘。
2. 使用 cgexec 与 dd 命令验证上述限制操作是否成功。

## 审核要求

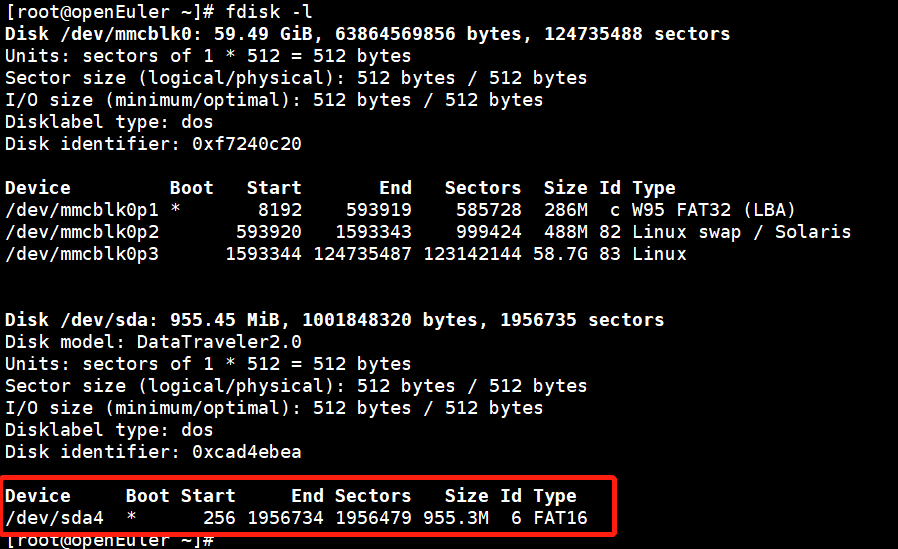
1. 正确、成功地限制U盘访问。
2. 提交每一步操作以及结果显示的截图。

## 参考答案

**一、操作步骤**

1、将U盘插入树莓派，使用 fdisk -l 获取该U盘的盘符，例如此处是：/dev/sda4

（不好判断哪个是插入U盘的盘符，可在插入U盘前后执行 fdisk -l 进行确认）



2、调用shell命令 “ls -l”获取设备号，如ls -l /dev/sda4，截图如下：



那么设备号为 8,4 。

3、将U盘挂载到当前系统中

|  |
| --- |
| # mkdir /usb  # mount /dev/sda4 /usb |

4、挂载设备管理devices子系统

|  |
| --- |
| # cd /cgroup/  # mkdir devices  # mount -t cgroup -o devices devices /cgroup/devices #挂载devices子系统  # cd /cgroup/devices  # mkdir mydevices # 创建mydevices控制组  # cd mydevices |

5、设置拒绝U盘访问

|  |
| --- |
| # echo 'a 8:4 rwm' > /cgroup/devices/mydevices/devices.deny |

这里的8:4就是我们上文中得到的设备号，将相关设置写入devices.deny文件实现拒绝u盘访问。

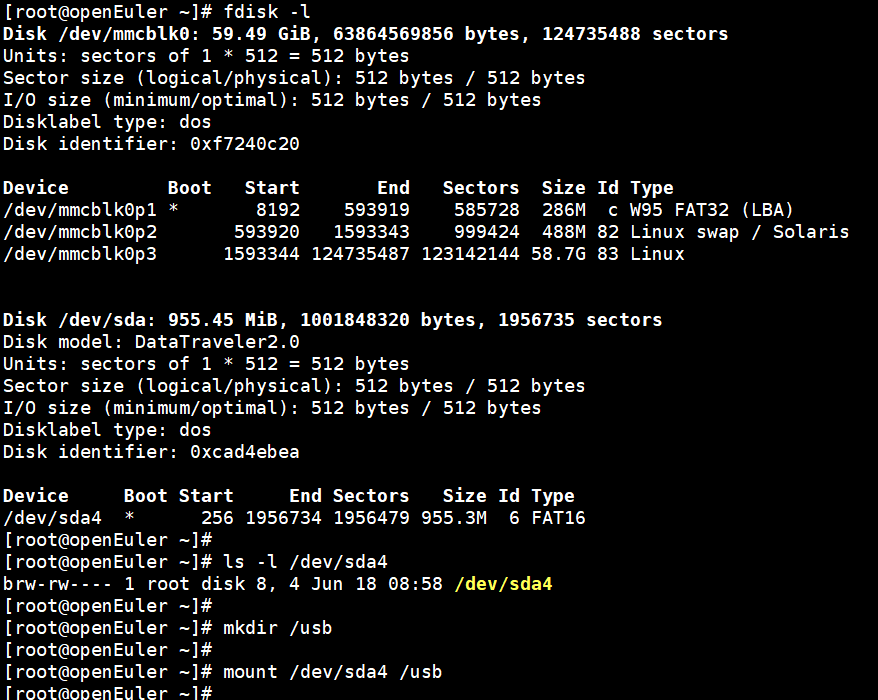
**二、测试验证**

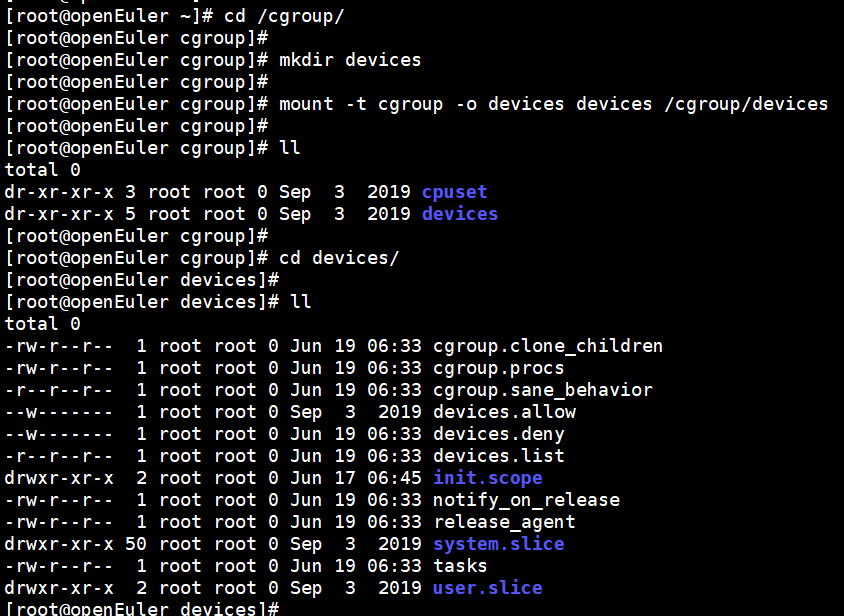
在给的测试文件夹中打开终端，切换至root用户，执行：

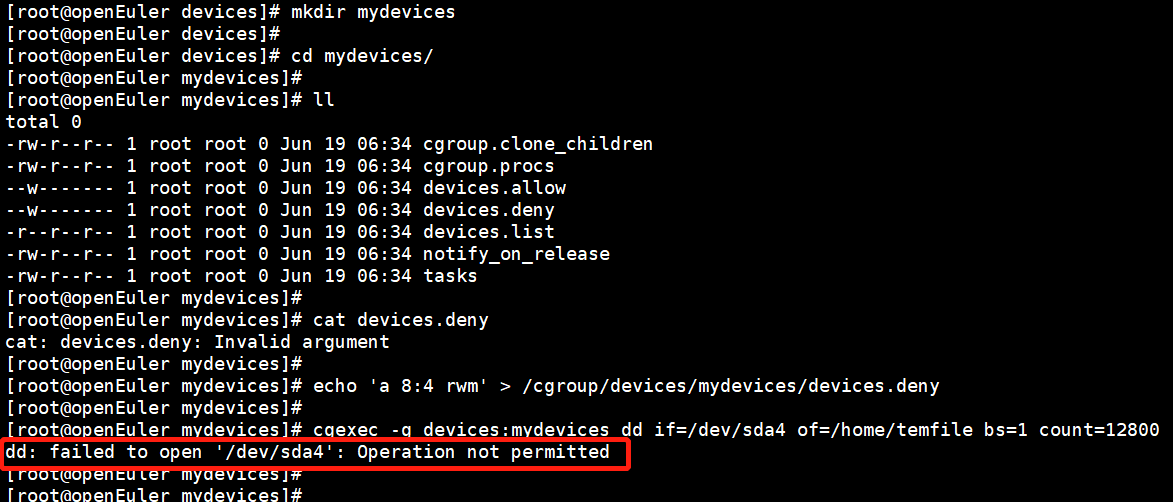
|  |
| --- |
| # cgexec -g devices:mydevices dd if=/dev/sda4 of=/home/temfile bs=1 count=12800 |

其中/dev/sda4就是前文读到的u盘的设备编号，如果终端显示Operation not permitted。就说明不允许访问u盘设置成功。

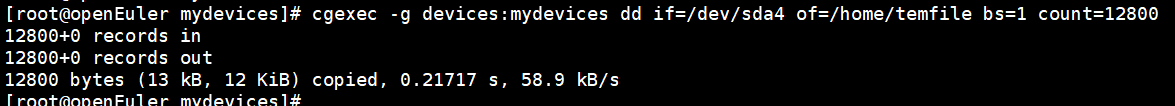
实际操作截图：







当允许访问U盘时，上述运行结果是：



# 任务3：使用cgroup限制内存利用率 (45min)

## 相关知识

同4.1

**memory.limit\_in\_bytes**

设定用户内存（包括文件缓存）的最大用量。如果没有指定单位，则该数值将被解读为字节。但是可以使用后缀代表更大的单位 —— k 或者 K 代表千字节，m 或者 M 代表兆字节 ，g 或者 G 代表千兆字节。

**memory.oom\_control**

包含标签（0 或者 1），它可以为 cgroup 启用或者禁用“内存不足”（Out of Memory，OOM） 终止程序。如果启用（0），尝试消耗超过其允许内存的任务会被 OOM 终止程序立即终止。默认情况下，所有使用 memory 子系统的 cgroup 都会启用 OOM 终止程序。要禁用它，请在 memory.oom\_control 文件中写入 1：

|  |
| --- |
| # echo 1 > /cgroup/memory/lab1/memory.oom\_control |

禁用 OOM 杀手程序后，尝试使用超过其允许内存的任务会被暂停，直到有额外内存可用。

memory.oom\_control 文件也在 under\_oom 条目下报告当前 cgroup 的 OOM 状态。如果该 cgroup 缺少内存，则会暂停它里面的任务。under\_oom 条目报告值为 1。

## 任务描述

1. 使用cgroup限制memory利用率。
2. 编写测试程序，使得进程占用的内存超出设置的阈值；验证上述限制操作是否成功。
3. 分别设置memory.oom\_control的值为0和1，运行测试程序，观察运行结果区别并分析。

## 审核要求

1. 正确、成功地限制内存利用率。
2. 提交每一步操作以及结果显示的截图。

## 参考答案

**一、操作步骤**

1、挂载cpuset管理子系统

在/sys/fs/cgroup/ memory下创建了一个mymemory的控制组。

|  |
| --- |
| # cd /sys/fs/cgroup/memory  # mkdir mymemory  # cd mymemory |

2、简单的死循环程序C源文件cgroup\_mem.c

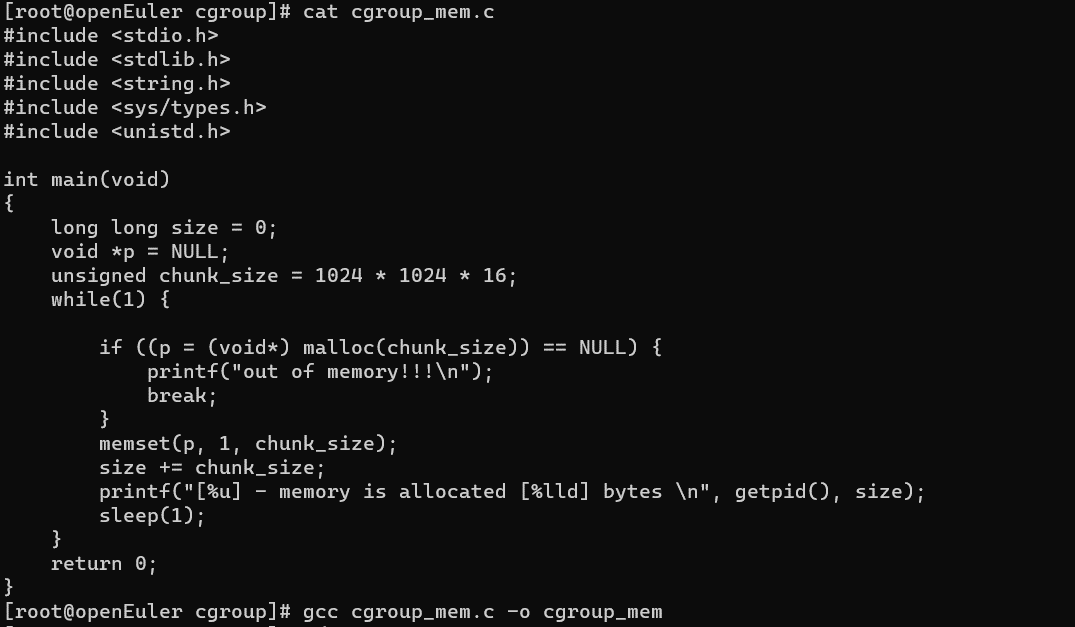
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/types.h>  #include <unistd.h>  int main(void)  {  long long size = 0;  void \*p = NULL;  unsigned chunk\_size = 1024 \* 1024 \* 16;  while(1) {  if ((p = (void\*) malloc(chunk\_size)) == NULL) {  printf("out of memory!!!\n");  break;  }  memset(p, 1, chunk\_size);  size += chunk\_size;  printf("[%u] - memory is allocated [%lld] bytes \n", getpid(), size);  sleep(1);  }  return 0;  } |

3、测试验证

（1）打开一个终端，执行以下命令：

|  |
| --- |
| ## 编译并运行上述C源文件  # cat cgroup\_mem.c  # gcc cgroup\_mem.c -o cgroup\_mem |

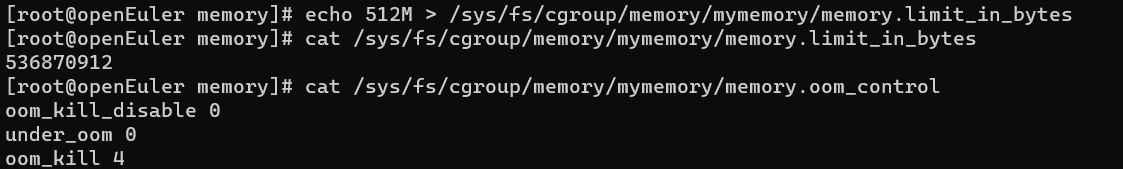
执行如下：

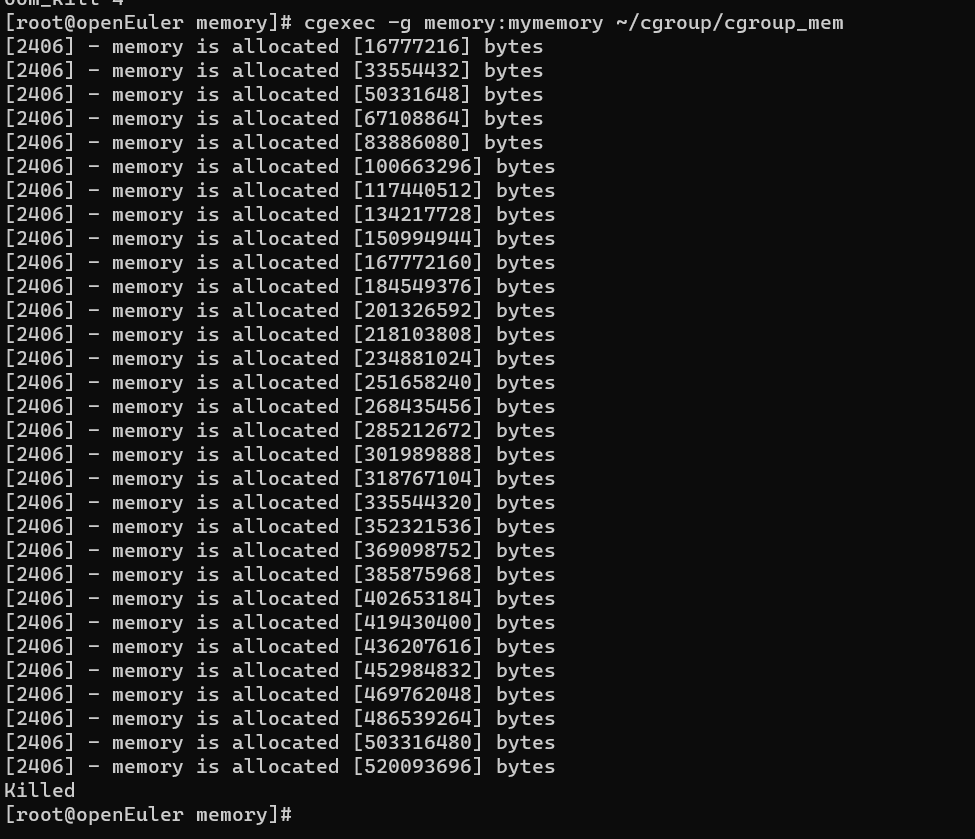


（2）设置内存利用限制

运行以下命令

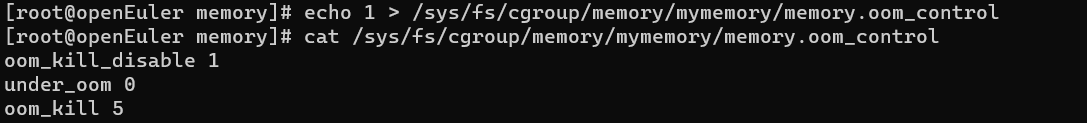
|  |
| --- |
| # echo 1G > /sys/fs/cgroup/memory/mymemory/memory.limit\_in\_bytes  # cat /sys/fs/cgroup/memory/mymemory/memory.limit\_in\_bytes  # cat /sys/fs/cgroup/memory/mymemory/ memory.oom\_control  # cgexec -g memory:mymemory ~/cgroup/cgroup\_mem |

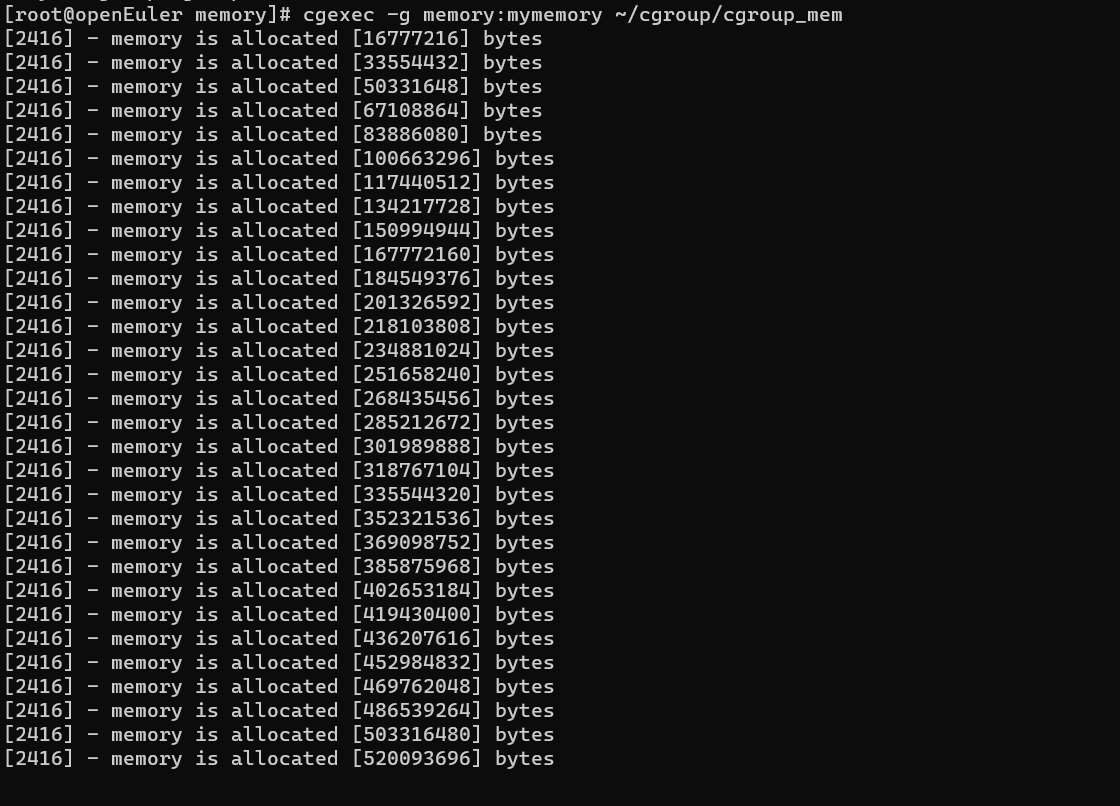




（3）修改memory.oom\_control并运行

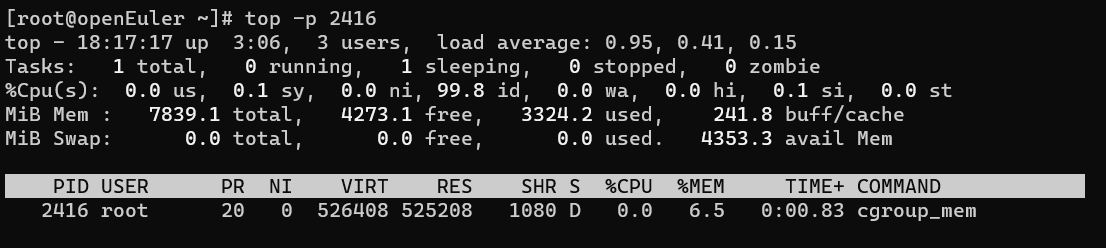
|  |
| --- |
| # echo 1 > /sys/fs/cgroup/memory/mymemory/memory.oom\_control  # cat /sys/fs/cgroup/memory/mymemory/memory.oom\_control  # cgexec -g memory:mymemory ~/cgroup/cgroup\_mem |





查看进程状态：

|  |
| --- |
| # top -p 2416 |



表示进程进入了不可中断的睡眠状态。

# 任务4：使用cgroup限制cpu利用率 (45min)

## 相关知识

同4.1

**cpu.rt\_runtime\_us**

此参数可以指定在某个时间段中， cgroup 中的任务对 CPU 资源的最长连续访问时间，单位为微秒（µs，这里以“*us*”表示），只可用于实时调度任务。建立这个限制是为了防止一个 cgroup 中的任务独占 CPU 时间。如果 cgroup 中的任务，在每秒内有 0.2 秒可存取 CPU 资源，请将 cpu.rt\_runtime\_us 设定为 200000，并将 cpu.rt\_period\_us 设定为 1000000。请注意：运行时间和阶段参数会根据 CPU 操作。例如：如要允许一个实时任务完全利用两个 CPU，请将 cpu.cfs\_quota\_us 设定为 200000 并将 cpu.cfs\_period\_us 设定为 100000。

**cpu.cfs\_period\_us**

此参数可以设定重新分配 cgroup 可用 CPU 资源的时间间隔，单位为微秒（µs，这里以 “us” 表示）。如果一个 cgroup 中的任务在每 1 秒钟内有 0.2 秒的时间可存取一个单独的 CPU，则请将 cpu.rt\_runtime\_us 设定为 2000000，并将 cpu.rt\_period\_us 设定为 1000000。cpu.cfs\_quota\_us 参数的上限为 1 秒，下限为 1000 微秒，默认为 100000，即百毫秒

**cpu.cfs\_quota\_us**

此参数可以设定在某一阶段（由 cpu.cfs\_period\_us 规定）某个 cgroup 中所有任务可运行的时间总量，单位为微秒（µs，这里以 "us" 代表）。一旦 cgroup 中任务用完按配额分得的时间，它们就会被在此阶段的时间提醒限制流量，并在进入下阶段前禁止运行。如果 cgroup 中任务在每 1 秒内有 0.2 秒，可对单独 CPU 进行存取，请将 cpu.cfs\_quota\_us 设定为 200000，cpu.cfs\_period\_us 设定为 1000000。请注意，配额和时间段参数都根据 CPU 来操作。例如，如要让一个进程完全利用两个 CPU，请将 cpu.cfs\_quota\_us 设定为 200000，cpu.cfs\_period\_us 设定为 100000。

如将 cpu.cfs\_quota\_us 的值设定为 -1，这表示 cgroup 不需要遵循任何 CPU 时间限制。这也是每个 cgroup 的默认值（root cgroup 除外）。

如要让一个 cgroup 完全使用一个 CPU，请使用以下指令：

|  |
| --- |
| # echo 10000 > /cgroup/cpu/red/cpu.cfs\_quota\_us  # echo 10000 > /cgroup/cpu/red/cpu.cfs\_period\_us |

如要让一个 cgroup 使用 CPU 的 10%，请使用以下指令：

|  |
| --- |
| # echo 10000 > /cgroup/cpu/red/cpu.cfs\_quota\_us  # echo 100000 > /cgroup/cpu/red/cpu.cfs\_period\_us |

在多核系统中，如要让一个 cgroup 完全使用两个 CPU 核，请使用以下指令：

|  |
| --- |
| # echo 200000 > /cgroup/cpu/red/cpu.cfs\_quota\_us  # echo 100000 > /cgroup/cpu/red/cpu.cfs\_period\_us |

## 任务描述

1. 使用cgroup限制cpu利用率。
2. 编写相关测试程序，使得进程的cpu利用率超出设置的阈值，验证上述限制操作是否成功。

## 审核要求

1. 正确、成功地限制CPU利用率。
2. 提交每一步操作以及结果显示的截图。

## 参考答案

**一、操作步骤**

1、挂载cpu管理子系统

在/sys/fs/cgroup/cpu下创建了一个mycpu的控制组。

|  |
| --- |
| # cd /sys/fs/cgroup/cpu  # mkdir mycpu  # cd mycpu |

2、简单的死循环程序C源文件cgroup\_cpu.c

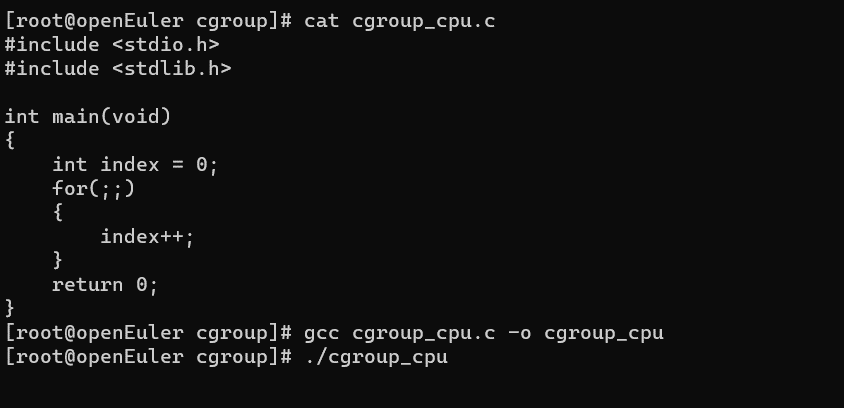
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main(void)  {  int index = 0;  for(;;)  {  index++;  }  return 0;  } |

4、测试验证

（1）打开一个终端，执行以下命令：

|  |
| --- |
| ## 编译并运行上述C源文件  # cat cgroup\_cpu.c  # gcc cgroup\_cpu.c -o cgroup\_cpu  # ./cgroup\_cpu |

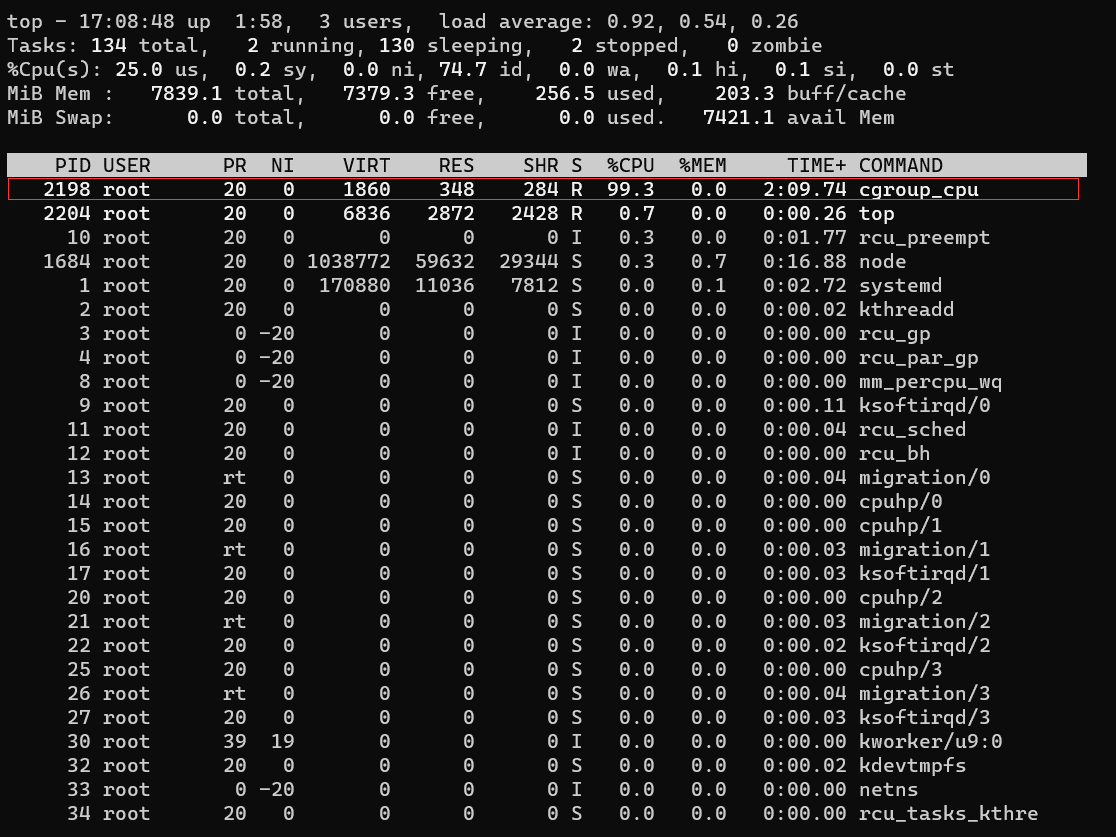
执行如下：



（2）不要关闭上述终端，另打开一个终端，执行以下命令：

|  |
| --- |
| # top //查看程序while\_long的PID, 假设为16238。输入q 退出当前查看状态 |

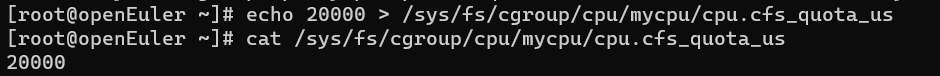
执行如下：



（3）设置cpu利用限制

退出top，运行以下命令

|  |
| --- |
| # echo 20000 > /sys/fs/cgroup/cpu/mycpu/cpu.cfs\_quota\_us  # cat /sys/fs/cgroup/cpu/mycpu/cpu.cfs\_quota\_us  #echo 2198 >> /sys/fs/cgroup/cpu/mycpu/tasks |





再次运行top命令

