4.6 实验 4-6：per-cpu 变量

1．实验目的

学会 Linux 内核中 per-cpu 变量的用法。

2．实验要求

1）写一个简单的内核模块，创建一个 per-cpu 变量，并且初始化该 per-cpu 变量，修改 per-cpu 变量的值，然后输出这些值。

3．实验步骤

下面是本实验的实验步骤。

进入本实验的参考代码目录进行交叉编译。

```
cd
/home/lab466/runninglinuxkernel_4.0/rlk_lab/rlk_basic/chapter_8/lab6_percpu
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabi-
    make BASEINCLUDE=/home/lab466/runninglinuxkernel_4.0
```

然后把 ko 内核模块拷贝到 runninglinuxkernel_4.0/kmodules 目录下面。

```
    cp mypercpu.ko /home/lab466/runninglinuxkernel_4.0/kmodules
```

启动 QEMU+runninglinuxkernel。最好另外开一个窗口，运行：

```
    sudo su
cd /home/lab466/runninglinuxkernel_4.0
    sh run.sh arm32
```

进入本实验的参考代码。

```
    # cd /mnt
```

安装本实验的内核模块。

```
benshushu:lab6_percpu# insmod mypercpu.ko
[14275.095433] module loaded at 0x000000004ef9b3f0
[14275.101083] init: cpuvar on cpu0  = 10
[14275.101371] init: cpuvar on cpu1  = 15
[14275.101608] init: cpuvar on cpu2  = 15
[14275.101833] init: cpuvar on cpu3  = 15
[14275.102501] init: cpu:0 cpualloc = 100
[14275.103728] init: cpu:1 cpualloc = 100
[14275.104307] init: cpu:2 cpualloc = 100
```

```
[14275.104787] init: cpu:3 cpualloc = 100
```

```
benshushu:lab6_percpu# insmod mypercpu.ko
[14275.095433] module loaded at 0x000000004ef9b3f0
[14275.101083] init: cpuvar on cpu0  = 10
[14275.101371] init: cpuvar on cpu1  = 15
[14275.101608] init: cpuvar on cpu2  = 15
[14275.101833] init: cpuvar on cpu3  = 15
[14275.102501] init: cpu:0 cpualloc = 100
[14275.103728] init: cpu:1 cpualloc = 100
[14275.104307] init: cpu:2 cpualloc = 100
[14275.104787] init: cpu:3 cpualloc = 100
```

卸载内核模块 rmmod mypercpu

```
benshushu:lab6_percpu# rmmod mypercpu
[14344.450078] exit module...
[14344.450489] cpuvar cpu0 = 15
[14344.450956] exit: cpualloc0 = 100
[14344.452227] cpuvar cpu1 = 20
[14344.452963] exit: cpualloc1 = 100
[14344.453146] cpuvar cpu2 = 15
[14344.453272] exit: cpualloc2 = 100
[14344.453442] cpuvar cpu3 = 15
[14344.453581] exit: cpualloc3 = 100
[14344.454607] Bye: module unloaded from 0x00000000fdb69cab
```

4. 实验代码

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/percpu.h>
#include <linux/cpumask.h>

static DEFINE_PER_CPU(long, cpuvar) = 10;
static long __percpu *cpualloc;

static int __init my_init(void)
{
      int cpu;

      pr_info("module loaded at 0x%p\n", my_init);

      /* modify the cpuvar value */
      for_each_possible_cpu(cpu){
          per_cpu(cpuvar, cpu) = 15;
          pr_info("init: cpuvar on cpu%d  = %ld\n",
               cpu, get_cpu_var(cpuvar));
          put_cpu_var(cpuvar);

      }

      __this_cpu_write(cpuvar, 20);

      /* alloc a percpu value */
      cpualloc = alloc_percpu(long);

      /* set all cpu for this value */
      for_each_possible_cpu(cpu){
          *per_cpu_ptr(cpualloc, cpu) = 100;
          pr_info("init: cpu:%d cpualloc = %ld\n",
                    cpu, *per_cpu_ptr(cpualloc, cpu));
      }

      return 0;
```

```
}

static void __exit my_exit(void)
{
      int cpu;
      pr_info("exit module...\n");

      for_each_possible_cpu(cpu) {
          pr_info("cpuvar cpu%d = %ld\n", cpu, per_cpu(cpuvar, cpu));
          pr_info("exit: cpualloc%d = %ld\n", cpu, *per_cpu_ptr(cpualloc,
cpu));
      }

      free_percpu(cpualloc);

      pr_info("Bye: module unloaded from 0x%p\n", my_exit);
}

module_init(my_init);
module_exit(my_exit);

MODULE_AUTHOR("Ben ShuShu");
MODULE_LICENSE("GPL v2");
```

per-cpu 变量是 Linux 内核中同步机制的一种。当系统中所有的 CPU 都访问共享的一个变量 *v* 时，CPU0 修改了变量 *v* 的值时，CPU1 也在同时修改变量 *v* 的值，那么就会导致变量 *v* 值不正确。一个可行的办法就是 CPU0 访问变量 *v* 时使用原子加锁指令，CPU1 访问变量 *v* 时只能等待了，可是这会有两个比较明显的缺点。

- 原子操作是比较耗时的。
- 现代处理器中，每个 CPU 都有 L1 缓存，那么多 CPU 同时访问同一个变量时会导致缓存一致性问题。当某个 CPU 对共享数据变量 *v* 修改后，其他 CPU 上对应的缓存行需要做无效操作，这对性能是有所损耗的。

per-cpu 变量为了解决上述问题出现一种有趣的特性，它为系统中每个处理器都分配该变量的副本。这样在多处理器系统中，当处理器只能访问属于它自己的那个变量副本，不需要考虑与其他处理器的竞争问题，还能充分利用处理器本地的硬件缓存来提升性能。

3）声明 per-cpu 变量。per-cpu 变量的定义和声明有两种方式：一个是静态声明，另一个是动态分配。

静态 per-cpu 变量通过 DEFINE_PER_CPU 和 DECLARE_PER_CPU 宏定义和声明一个 per-cpu 变量。这些变量与普通变量的主要区别是放在一个特殊的段中。

```
#define DECLARE_PER_CPU(type, name)                 \
   DECLARE_PER_CPU_SECTION(type, name, "")

#define DEFINE_PER_CPU(type, name)                 \
    DEFINE_PER_CPU_SECTION(type, name, "")
```

动态分配和释放 per-cpu 变量的 API 函数如下。

```
#define alloc_percpu(type)                    \
   (typeof(type) __percpu *)__alloc_percpu(sizeof(type),       \
                    __alignof__(type))
```

```
void free_percpu(void __percpu *ptr)
```

4）使用 per-cpu 变量。对于静态定义的 per-cpu 变量，可以通过 get_cpu_var()和 put_cpu_var()函数来访问和修改 per-cpu 变量，这两个函数内置了关闭和打开内核抢占的功能。另外需要注意的是，这两个函数需要配对使用。

```
#define get_cpu_var(var)                    \
(*({                                \
   preempt_disable();                   \
   this_cpu_ptr(&var);                   \
}))

#define put_cpu_var(var)                    \
do {                                \
    (void)&(var);                       \
   preempt_enable();                    \
} while (0)
```

访问动态分配的 per-cpu 变量需要通过下面的接口函数来访问。

```
#define put_cpu_ptr(var)                    \
do {                                \
    (void)(var);                        \
   preempt_enable();                    \
} while (0)

#define get_cpu_ptr(var)                    \
({                                  \
   preempt_disable();                   \
   this_cpu_ptr(var);                   \
})
```