1 timer.c部分

1.1 timer.c文件主要功能

在timer.c文件中，主要实现了两大部分功能：

1、在发生时钟中断时，更新当前进程相关的所有信息

2、扫描并执行到期的定时器任务

1.2 相关概念

**1.2.1 中断上半部与下半部**

中断上半部（top half）：接收到一个中断，立即开始执行（有严格时限的工作）

中断下半部（bottom half）：执行与终端处理密切相关但中断处理程序本身不执行的工作。中断下半部的工作是被允许推后完成的。

区分：

1、任务对时间非常敏感，属于中断上半部

2、任务和硬件相关，属于中断上半部

3、任务要保证不被其他中断打断，属于中断上半部

**1.2.2 中断下半部的实现**

下半部可以通过多种机制或者内核定时器来完成。

1、机制：任务队列、软中断、tasklet

2、定时器：定时器不同于机制，内核定时器把操作推迟到具体摸个确定的时间执行。（也是timer.c最主要的实现对象）

**1.2.3 双链表**

双链表是链表的一种，它的每个数据结点中都有两个指针，分别指向直接后继和直接前驱。

**1.2.4 定时器**

定时器（又称内核定时器、动态定时器），是管理内核时间流逝的基础。当内核需要推后执行某些程序时（比如中断下半部），就可以使用定时器。

**1.2.5 时间基准**

使用定时器时，有3个选项可以区分如何计算经过的时间，或定时器所处的基准。

1、ITIMER\_REAL：测量定时器激活以来，实际的流逝时间，在超时时发出信号。不受用户态、核心态、使用该定时器的应用程序是否运行影响。

2、ITIMER\_VIRTUAL：定时器拥有者进程在用户态消耗的时间内运行。

3、ITIMER\_PROF：用户态加核心态消耗的时间，只计算该进程进行时的时间。

时间基准的作用：分析应用程序的性能，查找程序中计算最密集的地方，进行优化

1.3 主要数据结构

**1.3.1 定时器的数据结构**

（定时器的数据结构没有在timer.c的程序中出现，但是因为timer.c的程序中会用到期数据结构，所以还是在报告中写下）

1、数据结构：

expires：表示期望定时器执行的 jiffies 值

function：到达该 jiffies 值时，将调用 function 函数

data：到达该 jiffies 值时，传递 data 作为function函数的参数。

entry：当一个定时器被注册到内核之后，entry 字段用来连接该定时器到一个内核链表中。

base ：是内核内部实现所用的。

2、具体的定时器数据结构程序实现：

struct timer\_list {

struct list\_head entry;

unsigned long expires;

void (\*function)(unsigned long);

unsigned long data;

struct tvec\_base \*base;

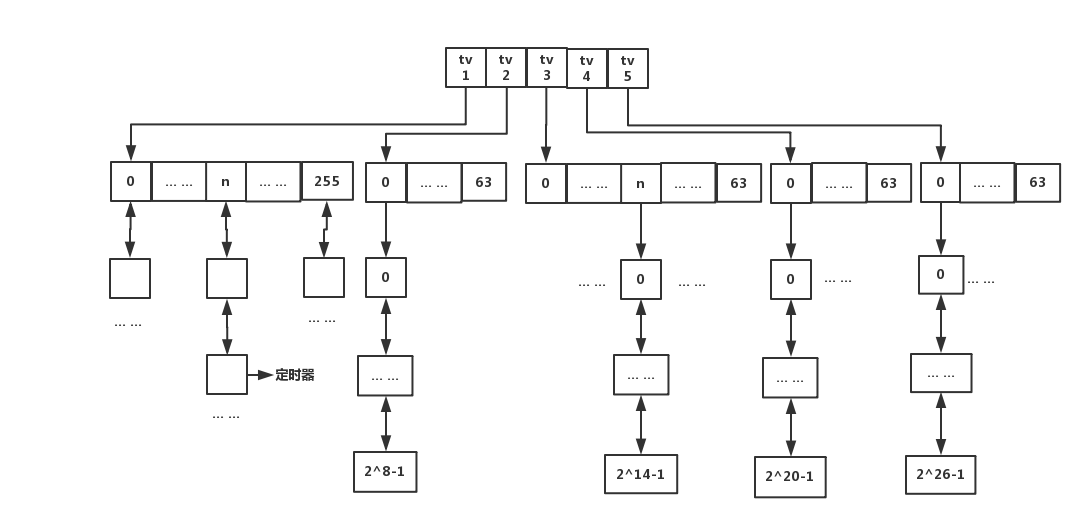
/\* ... \*/

};

**1.3.2 管理定时器的数据结构**

在每个时钟中断时，定时器的数据结构必须允许快速且高效的检查到所有定时器，以免消耗太多CPU时间。

1、用于管理定时器的数据结构如下：



图表 1 管理定时器的数据结构示意图

2、定时器分组：

为了提高效率，只是简单的将定时器（timer\_list）串联起来是不够的。因此内核为定时器创建了不同的组，根据定时器的到期时间（expires）对其进行分类。

如上图第一层，将一个主数组作为分类的基础，该数组共有五个数组项tv1、tv2、tv3、tv4、tv5。主数组根据到期时间对定时器进行了粗略的分类。每一组中包含了到期时间之间的所有定时器。按照到期时间的分类如下表。

每个组本身又由一个数组组成，定时器在其中再次进行较为细致的分类。

第一个数组有256个数组项，每个数组项表示了从0~255个时钟周期之间可能的到期时间。如果系统中存在两个或多个定时器，拥有相同的到期时间，那么他们之间用双链表连接（如图tv1到期时间为n的定时器）。内核主要负责关注第一组的定时器，因为只有第一组的定时器是即将要到期的。这样就能大大提高扫描定时器的时间，提高CPU效率。

其余的组和tv1有稍许不同，其余组也由数组项组成，但是每个组只有64个数组项。数组项中包含了定时器的双链表，每个数组项可允许的时间间隔不再为1，而是扩大为28+6\*(i-2) 。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | tv1 | tv2 | tv3 | tv4 | tv5 |
| 到期时间 | 0~28-1 | 28~214-1 | 214~220-1 | 220~226-1 | 226~232-1 |
| 数组项 | 28 | 26 | 26 | 26 | 26 |
| 数组项时间间隔 | 1 | 28 | 214 | 220 | 226 |

图表 2 定时器数组项信息统计表

3、具体程序实现：

程序为定时器数组设置了索引和定时器向量组。

索引：int index；指向数组元素，数组中保存了即将要到期的定时器。

（1）timer\_vec 实现tv2到tv5:

struct timer\_vec { //tv2-tv5

int index; //索引

struct list\_head vec[TVN\_SIZE]; //定时器向量组，TVN\_SIZE =64

};

（2）timer\_vec\_root实现tv1：

struct timer\_vec\_root { //tv1

int index; //索引

struct list\_head vec[TVR\_SIZE]; //定时器向量组，TVR\_SIZE =256

};

（3）定时器主数组：

static struct timer\_vec \* const tvecs[] = {

(struct timer\_vec \*)&tv1, &tv2, &tv3, &tv4, &tv5

}; //主数组，指向tv1。。。的结构变量

**1.3.3 jiffies相关数据结构**

jiffies提供了内核中一种简单形式的低分辨率时间管理方式。

1、系统计时器(jiffies)数据结构：

unsigned long volatile jiffies; //从开机开始算起的滴答数时间值（10ms一次滴答）

2、从上一个时钟中断开始的计数器(timer\_jiffies)：

static unsigned long timer\_jiffies; //表示上一次运行定时器机制时的jiffies值

**1.3.4 自旋锁的数据结构**

内核动态定时器链表是一种系统全局共享资源，为了实现对它的互斥访问，Linux定义了专门的自旋锁timerlist\_lock来保护，任何想要访问动态定时器链表的代码段都首先必须先持有该自旋锁，并且在访问结束后释放该自旋锁。

数据结构实现：spinlock\_t timerlist\_lock = SPIN\_LOCK\_UNLOCKED;

1.4 程序内主要函数的功能

**1.4.1 init\_timervecs()函数**

功能：实现对定时器数组的初始化

实现方法：分成两部分初始化。

1、对tv2~tv5所在的timer\_vec初始化

2、对tv1所在的timer\_vec\_root。

**1.4.2 internal\_add\_timer()函数**

功能：将一个不处于任何定时器向量中的定时器插入到它应该所处的定时器向量中去

实现方法：

1、将定时器的到期时间（expires）和上一次时钟的中断时间（timer\_jiffies）相减计算出idx

2、根据idx把定时器插入到相应的数组中。

**1.4.3 add\_timer()函数**

功能：将定时器插入到一个合适的定时器链表中

实现方法：

1、加锁

2、判断所指定的定时器是否已经位于在某个定时器向量中等待执行

若已经在：内核警告

若不在：调用internal\_add\_timer()函数，完成插入操作

3、解锁

**1.4.4 detach\_timer()函数**

功能：将定时器从某个链表中删除

实现方法：

判断所指定的定时器是否已经位于在某个定时器向量中等待执行

若不在：不执行任何操作；

若在：将定时器从它原来所处的链表中摘除

**1.4.5 mod\_timer()函数**

功能：当一个定时器已经被插入到内核动态定时器链表中后，修改该定时器的expires值

实现方法：

1、加锁

2、更新定时器的expires

3、调用detach\_timer()函数，将该定时器从它原来所属的链表中删除

4、调用internal\_add\_timer()函数，根据它新的expires值将该定时器重新插入到相应的链表中

5、解锁

**1.4.6 del\_timer()函数**

功能：将一个定时器从相应的内核定时器队列中删除（实际上是对detach\_timer()函数的高层封装）

实现方法：

1、加锁

2、调用detach\_timer()函数，将定时器从链表中删除

3、修改定时器前后的指针

4、解锁

**1.4.7 cascade\_timers()函数**

功能：随着jiffies值增大，定时器的值会越来越小。当index扫描完tv1后，需要进行定时器迁移，将后面tv2的值迁移到tv1。

实现方法：

因为tv1的定时器在最多256个时钟周期后就会耗尽，因此当定时器耗尽时，将后面tv2的第一个数组中的定时器依次前推，重新补足第一组。根据之前分析的不同时间间隔数组中的数组项，我们发现：

tv2数组中64项数组，每项中含有256种不同的到期时间，这256项正好迁移至tv1；

同理，tv3数组中64项数组，每项中含有214种不同的到期时间，这214项正好迁移至tv2；

同理，tv4数组中64项数组，每项中含有220种不同的到期时间，这220项正好迁移至tv3；

同理，tv5数组中64项数组，每项中含有226种不同的到期时间，这226项正好迁移至tv4；

函数调用internal\_add\_timer()函数来判断定时器应该插入到那个定时器向量。

**1.4.8 run\_timer\_list()函数**

功能：扫描并执行当前已经到期的定时器

实现方法：

1、上锁

2、判读否需要迁移定时器，调用cascade\_timers()函数

3、扫描定时器

4、执行到期的定时器

5、解锁

**1.4.9 second\_overflow()函数**

功能：处理微秒数成员溢出的情况

**1.4.10 update\_wall\_time\_one\_tick()函数**

功能：更新一次时钟滴答对系统全局时间的影响

**1.4.11 update\_wall\_time()函数**

功能：更新记录当前的wall\_clock时间的xtime

实现方法：调用second\_overflow()函数和update\_wall\_time\_one\_tick()函数

**1.4.12 do\_process\_times()函数**

功能：更新指定进程的总时间统计信息

实现方法：调用update\_wall\_time()函数

**1.4.13 do\_it\_virt()函数**

功能：为进程的用户态执行时间ITIMER\_VIRTUAL软件定时器更新时间间隔

**1.4.14 do\_it\_prof()函数**

功能：为进程的ITIMER\_PROF软件定时器更新时间间隔

**1.4.15 update\_one\_process()函数**

功能：更新当前进程中的所有与时间相关的统计信息以及成员变量，视需要向当前进程发送相应的信号

实现方法：

1、更新进程在当前CPU上的用户态执行时间统计

2、更新进程在当前CPU上的核心态执行时间统计

3、调用do\_process\_times()函数，更新当前进程的总时间统计

4、调用do\_it\_virt()函数，为当前进程的ITIMER\_VIRTUAL软件定时器更新时间间隔

5、调用do\_it\_prof()函数，为当前进程的ITIMER\_PROF软件定时器更新时间间隔

**1.4.16 update\_process\_times()函数**

功能：在发生时钟中断时更新当前进程以及内核中与时间相关的统计信息，并作出相应的动作

实现方法：调用update\_one\_process()函数

**1.4.17 calc\_load()函数**

功能：更新内核有关系统负载要素的估计值

**1.4.18 update\_times()函数**

功能：更新系统的平均负载、记录当前时间的全局变量

实现方法：调用calc\_load()函数

**1.4.19 timer\_bh()函数**

功能：实现定时器相关操作

实现方法：

1、调用update\_times(函数，更新内核和进程有关时间的统计数字

2、调用run\_timer\_list()函数，处理动态定时器

**1.4.20 do\_timer()函数**

功能：调用定时器

实现方法：

1、调用update\_process\_times()函数，在发生时钟中断时，更新当前进程相关的所有信息

2、标记timer\_bh()函数，使其只要满足条件就能运行

**1.4.21 sys\_相关函数**

sys\_alarm()函数：启动进程的ITIMER\_REAL间隔定时器

sys\_getpid()函数：获取当前进程号

sys\_getppid()函数：获取当前进程父进程的进程号

sys\_ getuid ()函数：获取当前用户号

sys\_geteuid()函数：获取当前有效用户号

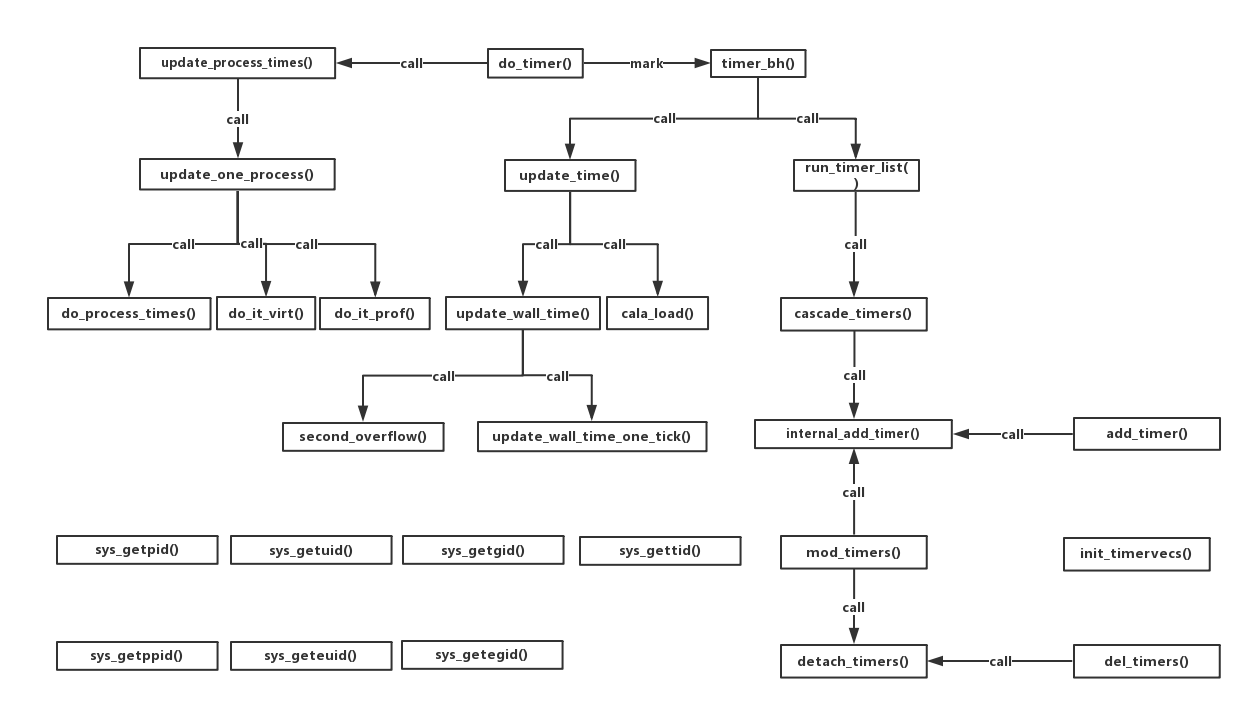
sys\_getgid()函数：获取当前组号

sys\_getegid()函数：获取当前有效组号

sys\_gettid()函数：获取当前线程号

sys\_nanosleep()函数：系统调用，让当前进程睡眠一段时间后继续运行

1.5 程序内整体函数调用框图



图表 3 timer.c函数间调用框图