Linux 下相对定时器池的实现及 TD-LTE 基站应用

黄 骞[1],胡群超[2]
(1.中国科学技术大学 软件学院,安徽 合肥 230051;
2.中科院南京宽带无线移动通信研发中心,江苏 南京 211111)

摘 要:针对 TD-LTE 系统基站应用,提出一种 Linux 用户空间下的相对定时器池的实现方法。结合哈希表、相对定时算法等技术,实现大数量定时器的高效管理,以 Linux 系统定时器单位为定时器粒度,定时器池满足基站高层协议软件大数量并发任务的应用需求。

关键词:TD-LTE;协议定时器;哈希表;定时器池;相对定时器

中图分类号:TN929.52 **文献标识码**:A **文章编号**:1674-7720(2014)01-0008-03

Implementation of relative-timer pool based on Linux and application of TD-LTE eNodeB

Huang Qian[1], Hu Qunchao[2]
(1.School of Software Engineering, University of Science and Technology of China, Hefei 230051, China;
2.Nanjing R&D Centre for Broadband Wireless Communications CAS, Nanjing 211111, China)

Abstract: For the requirement of high layer protocol software in TD-LTE system eNodeB, this paper proposed a new implementation of a relative-timer pool in the user space. Combined with hash list and relative timing appointment, implement a relative-timer pool to manage large number of timers. The relative-timergranularity is 100 milliseconds and timer pool meets thelow latency of the application requirements for high-layer protocol software, meanwhile the efficient management of a number of timers.

Key words: TD-LTE; protocol timer; hash list; timerpool; relative-timer

Linux 系统提供这样一种机制,预先设置一定时间长度并在设定的时间到期后执行预先设定的操作,这种机制即为定时器(timer)。Linux 系统面向用户提供多种用户级的定时器接口,而基于这些用户级定时器接口实现的应用于特定场合定时任务的定时器称为相对定时器(relative timer)。TD-LTE(时分长期演进)系统基站控制面高层协议需要使用较大数量的百毫秒级、秒级甚至是分钟级协议定时器,例如 S1 切换准备定时器的建议取值为 3 000 ms,S1 切换保护定时器的建议取值为 5 000 ms,S1、X2 再次切换间隔时间的建议取值为 1 min。为满足 TD-LTE 基站系统定时精度相对较低但定时器数量庞大的应用需求,本文实现了一种基于 Linux 系统定时器的相对定时器池,以守护进程(Daemon)的形式向基站控制面高层协议软件内部各个任务提供定时服务并负责较大数量的定时器管理功能,定时粒度为 100 ms,所实现的相对定时器池可稳定运行于 TD-LTE 系统基站设备。

1 工作流程及算法实现

1.1 定时器池架构

相对定时器池模块作为一个守护进程(Daemon)为高层协议软件的其他任务提供服务。相对定时器池架构如图 1 所示。定时器池后台任务拥有两个线程,主线程负责定时器池中超时定时器的检测,另一个线程处理用户任务对定时器池操作的请求消息。相对定时器池后台程序每隔 100 ms 检测一次定时器池链表,如果池内有

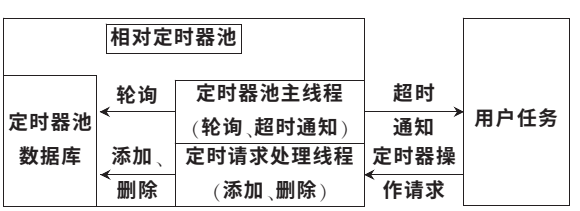


图 1 相对定时器池架构

超时定时器，将发送一个超时事件消息给注册该定时器的拥有者，通常是高层协议软件的一个任务（对于定时器池而言，即为用户任务）；定时器操作请求处理线程通过 socket 接收定时器请求消息[1]，这些请求转变为协议栈内部定义的消息，而且使用用户任务的特定参数发送消息给定时器池。

主要数据结构定义如下：

```
structlist_head {structlist_head*next, *prev;};
                //选用 linux 内核 list.h 中的双向链表结构
typedefstructtmr_q{structlist_headtime_vect_list;
                /* 链表元素用来组织定时器向量 Hash 表 */
structlist_headevent_list;
                /* 链表元素用来组织定时事件 Hash 表 */
TMR_PARA tmr_para;                    /* 定时器参数 */
}TMR_Q;                    /* 定时器池数据库单元结构 */
```

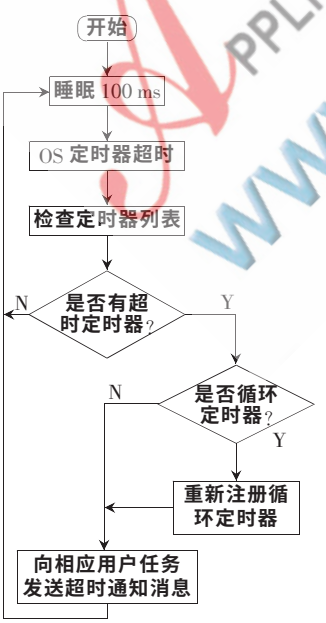
1.2 工作流程

1.2.1 主线程处理流程

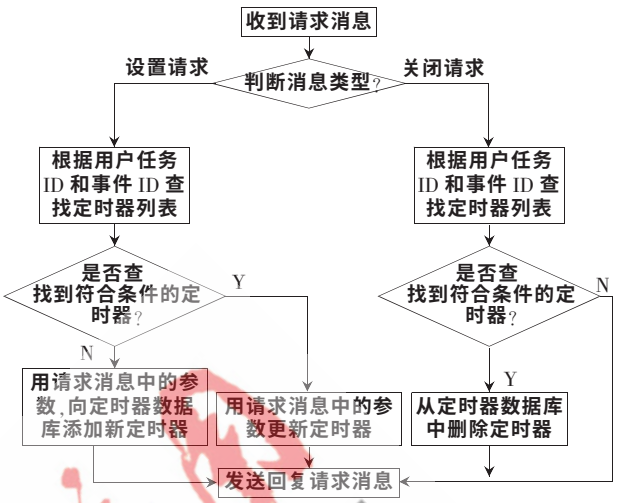
相对定时器池的主线程每隔 100 ms 被 Linux 系统定时器唤醒。此后，主线程检查定时器列表以便找出超时定时器。当有超时定时器被找出，线程将发送一个带有已注册事件 eventID 的超时通知消息给相应用户任务。当用户任务接收到这个消息，触发相应的处理方法来处理这个超时事件。主线程处理流程示意图如图 2(a)所示。

1.2.2 定时器操作请求消息处理流程

操作请求消息的处理线程是一个无限循环，它一直在等待接收用户任务通过 socket 发送的定时器操作请求消息，当接收到操作请求消息后进行相应的请求处理，请求消息处理流程图如图 2(b)所示。请求处理模块中有 4 个操作函数，函数定义如下：



(a)主线程定时轮询程序流程



(b)请求消息处理线程程序流程

图 2 相对定时器池程序流程图

```
TMR_Q*FindTimer(TMR_PARA*target);  /* 查找定时器 */
u32 AddTimer(TMR_PARA*Timer_new);  /* 添加定时器*/
void ModifyTimer(TMR_Q*timer,TMR_PARA*Timer_new);
                                   /* 修改更新定时器 */
void DeleteTimer(TMR_PARA*timer);  /* 删除定时器 */
```

1.2.3 用户接口的实现

为了用户任务使用定时器，定时器模块提供了两种操作接口，在用户任务中启动定时器和在任务中关闭定时器。如果一个任务要启动多个定时器，每一个定时器将使用不同的事件 eventID 来进行标识区分，在定时器发送超时事件消息给用户任务收到时，用户任务通过事件 eventID 分开处理这些事件。超时时间的单位为 100 ms，因此如果一个任务需要在 5.3 s 后触发一个定时操作，超时时间应该被设置为 53。因为资源限制等问题，定时器设置操作有可能失败，本定时器池向用户任务提供最多 1 024 个定时器。为简化消息接口处理，关闭定时器的用户请求消息只需要携带定时事件 eventID 参数即可完成定时器的删除。

1.3 相对定时算法实现

本定时器池中使用 Linux 内核 list.h 对定时器链表进行操作，相对定时算法围绕两个无符号长整型全局变量 current_ticks 和 stored_ticks，作为链表的哈希函数的计算以及相对定时的计算参数，定时器池守护进程开启后初始化这两个变量为 0，单位为 10 ms。定义用于存储定时器及定时事件的哈希表静态数组变量如式(1)，其中 TIMER_VECTOR_SZ=8，TOTAL_EVENT_NUM=24。

```
static structlist_head timer_vector
[TIMER_VECTOR_SZ],event_vector[TOTAL_EVENT_NUM],   (1)
free_timer_list;                   /* 空闲定时器标记 */
```

(1)添加定时器：整个定时器池数据库中的定时器分为两部分，已经被使用的定时器和空闲定时器。请求处理线程添加定时器 newTimer 时，首先计算相对定时时

间(relativeTime=Timeout+stored_ticks),随后计算定时器池哈希表的向量索引 timerIndex=relativeTime&(TIMER_VECTOR_SZ-1),如果新添定时器的 relativeTime 小于 stored_ticks,将 relativeTime 的值设置为 stored_ticks,将其插入到索引位置对应的向量链表 timer_vector[timerIndex]的头部,保证该定时器在主线程轮询第一轮循环中触发;否则,从链表后面向前比较直到第一个 relativeTime 提前于新添定时器的位置,将定时器插入,同时绑定定时器到事件向量链表,采用除留余数法计算定时事件哈希表的向量索引 eventIndex=eventID% TOTAL_EVENT_NUM。

(2)轮询定时器池:主线程经 100 ms 系统定时器唤醒后执行 current_ticks 加 10 操作,以 stored_ticks++为步长,current_ticks-stored_ticks>=0 为循环条件,对定时器向量哈希表进行轮询查找超时定时器,遍历 timer_vector[TIMER_VECTOR_SZ]向量索引下的链表,比较定时器池中已使用的定时器的相对定时时间 relativetime 与 current_ticks 的大小,小于或等于 current_ticks 则已超时,立即发送超时响应消息,随后在链表中删除该定时器并在此位置重新放入空闲定时器标记。

(3)删除定时器:在添加定时器时返回定时器的定时事件 eventID,删除定时器时通过 eventID 在事件链表 event_vector[TOTAL_EVENT_NUM]中查找。

2 性能测试及应用

(1)测试环境:PowerPC MPC8548,1.33 GHz,Linux 2.6.35。

(2)测试设计如下:为满足 TD-LTE 系统基站设备的实际应用需求,根据基站协议栈实际运用的定时时间片长度分为三类精度级别,分别为 500 ms,5 000 ms,60 s。对应每个定时时间片测试定时器池的定时器批量分别为 10 个、100 个、1 024 个定时器。测试开始前将定时器加入到定时器池中,完成初始化;然后开始工作,首先使用 gettimeofday 函数获取当前时间 t_Begin,主线程检查超时定时向用户任务发送超时通知消息前,再次获取当前时间 t_Now,记录并保存 t_Begin 和 t_Now 两个时间参数用于测试结果的统计,测试结果为定时器的绝对误差比和相对误差。定时器绝对误差比代表定时器的定时性能,测试得到的定时绝对误差与定时时间片的比值;相对误差代表定时器池的稳定性[2],即相同定时时间片大小的定时器在 100 次测试中的相对误差。

表 1 数据显示,定时器个数相同情况下,定时时间片越长,绝对误差比例越小。同时,在定时器满负荷(1 024 个定时器同时使用)也能保证良好的稳定性和定时性能。测试结果中的绝对误差比和定时误差均处于可忽略范围内,定时器池的定时性能稳定。表 1 每种情况测试 100 次。

表 1 相对定时器池性能测试结果

定时时间片/ms	10 个定时器		100 个定时器		1 024 个定时器	
	误差比/%	相对误差/ms	误差比/%	相对误差/ms	误差比/%	相对误差/ms
500	1.76	<2	2.12	<2	2.88	<3
5 000	0.63	<7	0.67	<12	0.70	<20
60 000	0.23	<10	0.30	<50	0.41	<80

本文设计并实现了一种基于哈希表定时器池算法,统一管理大量定时器,在满足用户任务定时需求、不影响定时器性能的前提下,提高了定时器池的容量及稳定性。通过测试验证,本文实现的相对定时器池的稳定性、精确性能均满足设计目标,稳定运行于基站设备。

参考文献

[1] STEVENS W R.UNIX 网路编程(第 2 卷)[M].北京:人民邮电出版社,2010.

[2] 许健,于鸿洋.Linux 下一种高性能定时器池的实现[J].电子技术应用,2012,38(12):114-119.

[3] 赵红武,之金瑜,刘云生.一种改进的定时器实现算法及其性能分析[J].微计算机应用,2006,27(3):343-345.

(收稿日期:2013-09-05)

作者简介:

黄骞,男,1990 年生,硕士研究生,主要研究方向:嵌入式系统设计。

胡群超,男,1982 年生,工程师,主要研究方向:无线通信协议栈。