Linux 中信号量机制研究

郑尚志[1], 陈祖爵[2], 韩 云[1,2], 陆 军[1]
(1. 巢湖学院 计算机系, 安徽 巢湖 238000;
2. 江苏大学 计算机科学与通信工程学院, 江苏 镇江 212013)

摘 要: 在 Linux 中, 信号量机制是实现并发进程同步、解决互斥的有效方法。文中以 Linux2.4 版为例, 系统地研究了信号和信号量机制, 从信号量分类入手详细论述了信号量的数据结构及相关调用, 不仅为全面、清晰地研究信号与信号量机制提供了有益的参考, 还为进一步应用信号量机制提供了支持。

关键词: Linux; 信号量; 内核信号量; 用户信号量
中图分类号: TP316.81 文献标识码: A 文章编号: 1673-629X(2007)12-0092-04

Research on Mechanism of Signal Quantity in Linux

ZHENG Shang-zhi[1], CHEN Zu-jue[2], HAN Yun[1,2], LU Jun[1]
(1. Computer Department, Chaohu College, Chaohu 238000, China;
2. Computer Science & Communication Engineering School, Jiangsu University, Zhenjiang 212013, China)

Abstract: In Linux, the mechanism of signal quantity is an effective method to realize the synchronism of concomitant process and solve the mutual exclusion. Takes Linux 2.4 edition as an example, systematically researches signal and signal mechanism. Starting with the signal quantity, discusses in detail data structure of signal quantity and its related transferring. It not only offers significant reference to the overall, clear signal research and the mechanism of signal quantity, but also offers support to the further use of the mechanism of signal quantity.

Key words: Linux; core signal quantity; user signal quantity

0 引 言

Linux 是多用户、多任务的操作系统, 相同时刻有多个进程或彼此竞争资源, 或协同完成一项任务。进程间存在同步和互斥, Linux 支持多种不同形式机制的 CIPCC, 主要有管道、信号、UNIX 系统 V 的 IPC 及套接字[1]等, 其中, 信号量机制是应用较广泛的一种方法。但由于涉及多个进程, 且随着在 Linux 的发展, 不断优化, 所以非常难于理解。虽然有不少文章对此进行研究, 但大都论述简略。文中将以 Linux2.4 版为例, 对 Linux 中的信号量机制作深入、细致和全面的剖析研究。

1 信号量

在 Linux 中, 信号和信号量是两种常见的通讯方式, 其中信号是最古老的进程通讯方式。在本质上, 信号是在软件层次上模拟中断机制, 而且信号是进程间唯一的异步通信机制[2]。信号量也称信号灯, 它和信号是不同的东西。信号是由若干个定值组成的, 而信号量是一个变量记录着某些特定信息。信号量(灯)与其它进程通信方式有所不同, 它主要用于进程间同步。信号量的分类有多种类型, 其中最重要的、也是最易混淆的为内核级信号量与用户级信号量。

1.1 内核信号量

内核信号量主要用于内核级进程之间的通信和内核级线程之间的通信。内核信号量是由系统定义并专有使用的, 用户进程不能对其操作。在它的类型定义中, 每个信号量至少需记录两个信息: 信号量的值和等待该信号量的进程队列[3]。数据结构如下:

```
struct semaphore {
    atomic_t count;
    int sleepers;
    wait_queue_head_t wait;
#if_IF WAITQUEUE_DEBUG
    long_magic;
```

收稿日期: 2007-03-19
基金项目: 国家自然科学基金资助项目(40674060)
作者简介: 郑尚志(1963-), 男, 安徽巢湖人, 高级实验师, 研究方向为操作系统理论、密码学; 陈祖爵, 副教授, 研究方向为嵌入式系统、网络技术。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
#endif
};
```

其中: count 相当于资源计数, 为正或 0 时表示可用资源数, －1 则表示没有空闲资源且有等待进程, 而其他负数仅仅是一个中间过程, 当所有进程都稳定下来后, 最终将变为－1; sleepers 原来的意思为等待的进程数, 但在应用中它相当于一个状态标志, 它仅有 0 和 1 两个值。当 sleepers 为 0 时, 表示没有进程在等待, 或等待进程在被唤醒中[4]。当 sleepers 为 1 时, 表示至少有一个进程在等待; wait 是等待队列, 但等待进程的数量并不被关心。

在内核级信号量的具体实现中, 有两个 inline 函数 down()和 up(), 其中 down()函数相当于 p 操作, up()函数相当于 v 操作。都在 include/asm－i386/semaphore.h 中, 在 down()函数中, count 做原子减 1 操作, 如果结果不小于 0, 则表示申请成功, 从 down()返回, 如果结果为负(大多数情况是－1, 注意判断"结果不小于 0"的原因, 是因为结果有可能是其他负数), 表示需要等待, 则调用_down_fail()。

当进程运行在内核态时, 是不会发生强制性进程切换的, 即代码中没有明确调用 schedule()函数, 就不会发生进程切换, 如果进程在内核态发现需要进行进程切换, 而自己又不想进入睡眠状态, 一般是设置进程数据结构中的标志 need_resched 为 1, 表示本进程在返回用户态前, 需要进行一次进程切换。

中断发生在用户态运行中, 可能发生进程切换, 但在内核态运行中却不会发生进程切换。因为在用户态有时钟中断, 当该进程的时间片到了时, 它就必须让; 在内核态则不同, 从中断返回时, 系统会首先判断是在什么运行态下发生中断的, 只有在用户态发生的中断, 才有可能在这里进行进程切换。

1.2　用户信号量

为了实现与其他系统的兼容, Linux 也支持 3 种 UNIX 系统 V 的进程间通信机制 IPC: 消息、信号量和共享内存[5]。每一对象都具有同样类型。但是 UNIX 系统 V 信号量实际上是一个信号量的集合, 且是由用户自己定义和使用的信号量集合。那么为什么已有内核信号量, 还需要用户自己定义信号量集合呢? 答案是: 因为内核信号量只有一个值, 而某些进程需要共享多种资源, 同时它们之间还需要同步。

每个用户信号量集合中包含很多信号量, 每个信号量都有一个值, 可以用来表示当前该信号量代表的共享资源可用(available)数量, Linux 对信号量有各种各样的限制, 信号量相关的重要数据结构放在 sem.h 文件中。

1.2.1　与用户信号量相关的数据结构及关系

在系统中的每个用户信号量集对应一个 sem_array 结构:

```
struct sem_array {
    struct kern_ipc_perm sem_perm; /* IPC 权限 */
    time_t sem_otime; /* 最后一次对信号量操作的时间 */
    time_t sem_ctime; /* 最后一次修改的时间 */
    struct sem *sem_base; /*在信号量集合中指向第一个信号量的指针 */
    struct sem_queue *sem_pending; /* 指向请求挂起队列头的指针 */
    struct sem_queue **sem_pending_last; /* 指向挂起队列尾的指针 */
    struct sem_undo *undo; /*指向该信号量数组上的 undo 请求链表(欠债)的指针 */
    unsigned long sem_nsems; /*信号量数组上的信号量序号 */
};
```

其中:

① sem 结构如下:

```
struct sem{
    int semval; /*信号量的当前值 */
    int sempid /* 在信号量上最后一次操作的进程识别号 */
}
```

②系统中每个因为信号量而睡眠的进程, 都对应一个 sem_queue 结构, sem_queue 结构如下:

```
struct sem_queue {
    struct sem_queue * next; /*队列中下一个节点 */
    struct sem_queue ** prev; /* 队列中上一个节点 */
    struct task_struct * sleeper; /* 正在睡眠的进程 */
    struct sem_undo * undo; /* 指向 undo 结构的指针 */
    int pid; /* 请求进程的进程识别号 */
    int status; /* 操作的完成状态 */
    struct sem_array * sma; /* 有操作的信号量集合数组 */
    int id; /* 内部信号量的识别号 */
    struct sembuf * sops; /* 挂起操作的数组 */
    int nsops; /*操作的个数 */
    int alter; /*操作要改变的信号量 */
};
```

③当进程退出时, 执行一系列的调整恢复请求(还债), 其数据结构为 sem_undo 如下:

```
struct sem_undo {
    struct sem_undo proc_next; /*在这个进程上的下一个 sem_undo 节点 */
    struct sem_undo; id_next /* 在这个信号量集上的下一个 sem_undo 节点 */
    int semid; /*信号量集的标识号 */
    short *semadj; /*信号量集的调整, 每个进程一个 */
};
```

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

④各数据结构间关系如图 1 所示。

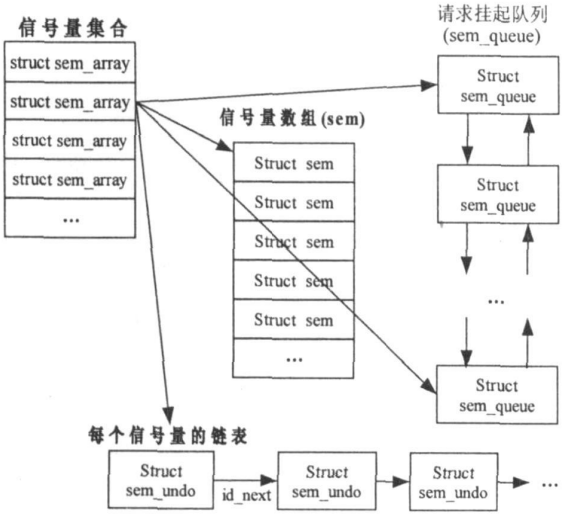

图 1 IPC 信号量数据结构图

1.2.2 用户信号量的创建或打开已存在的信号量

用户进程为共享某一信号量集合，必须首先创建相关的信号量集合，如果此信号量集合已存在，则只需打开即可。有时创建或打开的信号量也需要进行初始化。创建一个新的信号量组或获取一个已经存在的信号量组，使用 semget()函数，具体格式为：

int semget(key_t key, int nsems, int semflg);

其中参数含义：Key 是键值；nsems 是新创建的集合中信号量的个数，其最大值为 250；semflg 是一些标志位，主要为 IPC_CREAT 和 IPC_EXCL。使用时，如 IPC_CREAT 单独使用，semget()值为一个新创建的信号量集合的标识号，如已存在此信号量集合，则直接返回其值。如 IPC_CREAT 与 IPC_EXCL 一起使用，则或者创建一个信号量集合，或者当已存在的集合返回 -1，只有两者联用，才可保证新创建集合的打开和存取。如创建成功，则系统给此信号量集合分配空间。

当系统调用 semget()创建信号量集时，调用进程的同时也初始化了该信号量集。

1.2.3 用户信号量的操作

在 Linux 中，通过 semop()函数来实现 P、V 操作。如果一个进程要申请共享资源，那么就从信号量值中减去要申请的数目，如果当前没有足够的可用资源，进程可以睡眠等待，也可以立即返回。需要强调的是 semop()同时操作多个信号量，在实际应用中，对应多种资源的申请或释放。Semop()保证操作的原子性，尤其对于多种资源的申请来说，要么一次性获得所有资源，要么放弃申请，要么在不占有任何资源情况下继续等待，这样，一方面避免了资源的浪费；另一方面，避免了进程之间由于申请共享资源造成死锁[6]。

Semop()的数据结构如下：

int semop (int semid, struct sembuf *sop, int nsops);

其中：semid 是信号量集合的 ID；sops 指向一个类型为 sembuf 的数组，其数组的每一个分量都表示在一个信号量上的操作，如果这个数组有多个分量，则就表示一次 semop()可对多个信号量进行操作，同时 sop 参数中的 sem_op 字段还指明了对信号量进行的是 P 操作还是 V 操作；nsops 为 sops 指向数组的大小。

在对信号量操作 semop 时会使用 sembuf 结构，其数据结构如下：

```
struct sembuf{
    unsignaed short sem_num; /*数组中信号量的序号*/
    short sem_op; /* 信号量操作*/
    short sem_flg; /*操作标记*/
}
```

其中：sem_num 对应信号量集中的一个信号量，0 对应第一个信号量；sem_flg 可取 IPC_NOWAIT、SEM_UNDO 两个标志，如果设置了 SEM_UNDO 标志，那么在进程结束时，相应的操作将被取消，这是比较重要的一个标志位。如果设置了该标志位，那么在进程没有释放共享资源就退出时，内核将代为释放。内核分配一个 sem_undo 结构来记录它，为的是确保以后资源能够安全释放。事实上，如果进程退出了，那么它所占用的资源就释放了，但信号量值却没有改变，此时，信号量值反映的已经不是资源占有的实际情况，在这种情况下，问题的解决就靠内核来完成。这有点像僵尸进程，进程虽然退出了，资源也都释放了，但内核进程表中仍然有它的记录[7]，此时就需要父进程调用 waitpid 来解决问题了；sem_op 的值为正数，等于 0 以及负数确定了对 sem_num 指定的信号灯进行的三种操作：信号灯的当前值记录相应资源目前可用数目；sem_op＞0 表示进程要释放 sem_op 数目的共享资源(V 操作)；sem_op＝0 表示相应进程用于对共享资源是否已用完的测试；sem_op＜0 表示相应进程需申请 sem_op 数目的共享资源(P 操作)。

1.2.4 用户信号量的控制与管理

用户信号量集合在创建和使用过程中，用户还可以对它进行控制，主要有获取一些信号量的使用信息或者对信号量赋初值。使用的函数为 semctl()，其数据结构为：

int semctl(int semid, int semnum, int cmd, union semun arg)

其中：semid 为信号量集合的 ID；参数 semnum 指定对哪个信号灯操作，只对几个特殊的 cmd 操作有意义；cmd 指定具体的操作类型；arg 用于设置或返回信

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

号量的值。

参数 cmd 所能指定的操作: IPC_STAT 获取信号量信息, 信息由 arg.buf 返回; IPC_SET 设置信号量信息, 待设置信息保存在 arg.buf 中(在 manpage 中给出了可以设置哪些信息); GETALL 返回所有信号量的值, 结果保存在 arg.array 中, 参数 sennum 被忽略; GETNCNT 返回等待 semnum 所代表信号量的值增加的进程数, 相当于目前有多少进程在等待 semnum 代表的信号量所代表的共享资源; GETPID 返回最后一个对 semnum 所代表信号量执行 semop 操作的进程 ID; GETVAL 返回 semnum 所代表信号量的值; GETZCNT 返回等待 semnum 所代表信号量的值变成 0 的进程数; SETALL 通过 arg.array 更新所有信号量的值; 同时更新与本信号集相关的 semid_ds 结构的 sem_ctime 成员; SETVAL 设置 semnum 所代表信号量的值为 arg.val; 调用返回: 调用失败返回 -1, 成功返回与 cmd 相关。

2 结　论

在 Linux 信号量机制中, 内核信号量与用户信号量之间的区别在于内核信号量由系统定义, 且只有一个值, 只能实现内核进程简单的同步, 使用方法简便。而用户信号量则可以由用户定义一组信号量, 用于进程间共享多个资源的同步, 并使用 3 个系统调用进行操作, 要求操作必须具有整体性, 而且各数据结构之间的关系较复杂, 通过对内核信号量与用户信号量的深入研究, 为进一步应用信号量机制打下了坚实的基础。

参考文献:

[1] Molloy S, Man P H. Scalable Linux Scheduling[R]. CITI Technical Report. [s.l.]: [s.n.], 2001: 285-295.

[2] LI Chun-guang, WEN Tao, XU Qiang. Analysis of Linux Kernel Service Mechanism[J]. Fushun Shiyou Xuebao, 1998, 9(18): 65-68.

[3] 王文义, 武华北. Linux 中进程间信号通信机制的分析及其应用[J]. 计算机工程与应用, 2005(3): 108-115.

[4] 王社国. Linux 信号量通信机制分析与实践[J]. 微机发展, 2002, 12(6): 63-66.

[5] 李　晋, 葛敬国. Linux 下互斥机制及其分析[J]. 计算机应用研究, 2005(8): 72-77.

[6] 胡希明, 毛德操. Linux 内核源代码情景分析[M]. 杭州: 浙江大学出版社, 2001.

[7] 倪继利. Linux 内核分析及编程[M]. 北京: 电子工业出版社, 2005.

(上接第 88 页)

Machine Vision[M]. 2nd ed. New York: PWS Publishing, 1999.

[8] Reddy B S, Chatterji B N. An FFT-Based Technique for Translation, Rotation, and Scale Invariant Image Registration [J]. IEEE Trans Image Procession, 1996, 5(7): 1215-1220.

[9] Belongie S, Malik J, Puzicha J. Shape matching and object recognition using shape contexts[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(4): 509-522.

[10] 章夏芬, 庄越挺, 鲁伟明, 等. 根据形状相似性的书法内容检索[J]. 计算机辅助设计与图形学学报, 2005, 17(11): 2565-2569.

[11] Zhang T Y, Suen C Y. A Fast Parallel Algorithm for Thinning Digital Patterns[J]. Comm ACM, 1984, 27(3): 236-239.

[12] Hu M K. Visual Pattern Recognition by Moment Invariants [J]. IEEE Trans Information Theory, 1962, 8: 179-187.

[13] 李宗民, 刘玉杰, 李振波, 等. bezier 矩及其在人体姿态识别中的应用[J]. 计算机工程, 2005, 24: 38-40.

[14] Zheng Huicheng, Daoudi M, Jedynak B. Blocking Adult Images Based on Statistical Skin Detection[J]. Electronic Letters on Computer Vision and Image Analysis, 2004, 4(2): 1-14.

(上接第 91 页)

的可行性, 及其对简单利用等价类的概念进行聚类分析方法的改进。这种聚类方法解决了传统聚类方法中对定性数据处理的不足, 它无需把定性数据变换为定量数据, 并且该方法又不是简单地利用等价关系进行聚类, 而是通过分析属性的重要性, 并对数据进行约简, 得到了意义较为明确的聚类结果, 避免了简单利用等价关系聚类而产生结果的不明确性。

参考文献:

[1] Ziarko W. Variable Precision Rough Set Model[J]. Journal of Computer and System Science, 1993(40): 39-59.

[2] Barbar′A D, Chen P. Using Self-Similarity to Cluster Large Data Sets[J]. Data Mining and Knowledge Discovery, 2003, 7: 123-152.

[3] 李树军, 纪宏军. 对应聚类分析与变量选择[J]. 地球物理学进展, 2005, 20(3): 694-697.

[4] 来升强, 朱建平. 数据挖掘中高维定性数据的粗糙集聚类[J]. 统计研究, 2005(8): 56-60.

[5] 张文修. 粗糙集理论与方法[M]. 北京: 科学出版社, 2001.

[6] 朱建平. 数据挖掘的统计方法及实践[M]. 北京: 中国统计出版社, 2005.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net