Linux 进程结构及组织方式研究

殷联甫 沈士根 郭 步
(嘉兴学院信息工程学院 浙江 嘉兴 314001)

摘 要 本文首先对 Linux 中进程 PCB 的结构作了详尽的分析,然后对 Linux 中进程的组织方式作了深入的探讨。深刻理解和掌握 Linux 的进程结构及组织方式,对于我们正确分析 Linux 源代码和掌握 Linux 操作系统的实现方法具有非常重要的意义。

关键词 Linux 进程 操作系统

RESEARCH ON LINUX PROCESS STRUCTURE AND ORGANIZATION

Yin Lianfu Shen Shigen Guo Bu
(*School of Information Engineering, Jiaxing College, Jiaxing Zhejiang 314001, China*)

Abstract In this paper we firstly explain the structure of Linux PCB in detail, then we deeply explain the organization of Linux process. Linux process structure and organization is very important for us to understand operating system principles and the implementation of Linux.

Keywords Linux Process Operating system

1 引 言

Linux 是一个源代码公开的操作系统,通过分析 Linux 源代码,我们可以更好地理解操作系统的工作原理和实现方法,对我们开发具有自主知识版权的操作系统具有一定的指导意义和借鉴作用。

进程是操作系统的核心,是操作系统的调度单位,进程的概念贯穿于整个操作系统。深刻理解和掌握 Linux 的进程结构及组织方式,对于我们正确分析 Linux 源代码和 Linux 操作系统的实现方法具有非常重要的意义。

2 Linux 进程概述

进程由程序、数据和进程控制块 PCB (Process Control Block)组成。进程控制块 PCB 是进程存在惟一标识,系统通过 PCB 的存在而感知进程的存在。当系统创建一个进程时,实际上是建立一个 PCB。当进程消失时,实际上是撤消 PCB。在进程活动的整个生命周期内,系统通过 PCB 对进程进行管理和调度。

在 Linux 中,进程是操作系统调度单位,进程中的 PCB 用一个名为 task _ struct 的结构体来表示,定义在 include/linux/sched. h 中。在 2. 4. 0 版本中,每个 task _ struct 结构占 1680 字节,系统中的最大进程数由系统的物理内存大小决定。

每当创建一个新进程时,便在内存中申请一个空的 task _ struct 结构,填入所需信息。同时,指向该结构的指针也被加入到 task 数组中,所有进程控制块都存储在 task[] 数组中。

Linux 中的进程分普通进程和实时进程两种,实时进程具有一定的紧迫性,对外部事件要作出快速响应,实时进程的优先级高于普通进程。

3 Linux 进程 PCB 的结构

Linux 中进程的 PCB 用 task _ struct 结构体来表示,task _ struct 结构体比较复杂,总共有 80 多个数据成员。下面我们将其中的主要数据成员按功能划分成几个部门,同时对每个部分中的重要数据成员作详细的说明和解释。

3.1 调度数据成员

1) volatile long state

表示进程的当前状态。进程运行时,它会根据具体情况改变状态。进程状态总共有 TASK _ RUNNING(可运行状态)、TASK _ INTERRUPTIBLE(可中断的等待状态)、TASK _ UNINTERRUPTIBLE(不可中断的等待状态)、TASK _ ZOMBIE(僵死状态)、TASK _ STOPPED(暂停状态)等 5 种。

2) long priority

进程优先级,priority 的值给出了进程每次获取 CPU 后,可使用的时间片长度(单位是 jiffies)。

3) unsigned long rt _ priority

rt _ priority 的值给出了实时进程的优先级,rt _ priority + 1000 给出进程每次获取 CPU 后,可使用的时间片长度(单位是 jiffies)。

4) long counter

在轮转法调度时 counter 表示当前进程还可运行多久。在进程开始时被赋为 priority 的值,以后每隔一个时钟中断递减 1,减到 0 时引起新一轮调度。

5) unsigned long policy

表示该进程的进程调度策略。调度策略有:

收稿日期 2004 - 03 - 08。殷联甫,讲师,主研领域 操作系统,计算机安全。

- SCHED _ OTHER 0,非实时进程,用基于优先权的轮转法。
- SCHED _ FIFO 1,实时进程,用先进先出算法。
- SCHED _ RR 2,实时进程,用基于优先权的轮转法。

3.2 进程队列指针

1) struct task _ struct *next _ task, *prev _ task

在 Linux 中所有进程(以 PCB 的形式)组成一个双向链表,next _ task 和 prev _ task 是链表的前后向指针。

2) struct task _ struct *p _ opptr, *p _ pptr

struct task _ struct *p _ cptr, *p _ ysptr, * p _ osptr

以上分别是指向该进程的原始父进程、父进程、子进程和新老兄弟进程的指针。

3) struct task _ struct *pidhash _ next

struct task _ struct **pidhash _ pprev

用于链入进程 hash 表的前后指针。系统进程除了链入双向链表外,还被加到 hash 表中。

3.3 进程标识

1) uid _ t uid gid _ t gid uid 和 gid 分别是运行进程的用户标识和用户组标识。

2) pid _ t pid pid _ t pgrp pid 和 pgrp 分别是运行进程的进程标识号和进程组标识号。

3.4 时间数据成员

1) long per _ cpu _ utime[NR _ CPUS] per _ cpu _ stime[NR _ CPUS]

per _ cpu _ utime 是用户态进程运行的时间,per _ cpu _ stime 是内核态进程运行的时间。

2) unsigned long start _ time

进程创建时间。

3.5 文件系统数据成员

1) struct fs _ struct *fs

fs 保存了进程本身与 VFS(虚拟文件系统)的关系信息。

```
struct fs _ struct
{
  atomic _ t count;
  rwlock _ t lock;
  int umask;
  struct dentry *root, *pwd, *altroot;
  struct vfsmount *rootmnt, *pwdmnt, *altrootmnt;
}
```

其中 root、rootmnt 是根目录的 dentry 和其 mount 点的 vfsmount。pwd、pwdmnt 是当前工作目录的 dentry 和其 mount 点的 vfsmount。altroot、altrootmnt 是保存根节点被替换之后原来根目标的 dentry 和其 mount 点的 vfsmount。

2) struct files _ struct *files

files 包含了进程当前所打开的文件。

3) int link _ count

文件链的数目。

3.6 内存数据成员

1) struct mm _ struct *mm

在 Linux 中,采用按需分页的策略解决进程的内存需求。task _ struct 的数据成员 mm 指向关于存储管理的 mm _ struct 结构。

2) struct mm _ struct *active _ mm

active _ mm 指向活动地址空间。

3) mm _ segment _ t addr _ limit

表示线程空间地址。

用户线程空间地址 0-0xBFFFFFFF。

内核线程空间地址 0-0xFFFFFFFF。

4) spinlock _ t alloc _ lock

用于申请空间时用的自旋锁。自旋锁的主要功能是临界区保护。

3.7 页面管理

1) int swappable:1

进程占用的页面是否可换出。swappable 为 1 表示可换出。

2) unsigned long min _ flt, maj _ flt

该进程累计 minor 缺页次数和 major 缺页次数。

3) unsigned long nswap

该进程累计换出页面数。

4) unsigned long swap _ cnt

下一次循环最多可换出的页数。

3.8 支持对称多处理机方式的数据成员

1) int has _ cpu, processor

processor 指向进程正在使用的 CPU,has _ cpu 表示进程是否占有 CPU。

2) struct thread _ struct thread

记录该进程的 CPU 状态信息。

3.9 其他数据成员

1) unsigned short used _ math

是否使用 FPU。

2) char comm[6]

进程正在运行的可执行文件的文件名。

3) volatile long need _ resched

重新调度标志。当需要 Linux 调度时该变量置位。

4 Linux 中进程的组织方式

在 Linux 中,每个进程都有自己的 task _ struct 结构。在 2.4.0 版中,系统拥有的进程数取决于物理内存的大小,因此进程数可能达到成千上万个。为了对系统中的很多进程及处于不同状态的进程进行管理。Linux 采用了以下几种组织方式来管理进程。

4.1 哈希表

哈希表是进行快速查找的一种有效的组织方式。Linux 在进程中引入的哈希表叫做 pidhash,在 include/linux/sched.h 中定义如下:

struct task _ struct *pidhash[PIDHASH _ SZ];

PIDHASH _ SZ 在 include/linux/sched.h 中定义,其值为 1024。系统根据进程的进程号求得 hash 值,加到 hash 表中:

#define pid _ hashfn(x)((((x) > > 8) ^ (x)) & (PIDHASH _ SZ - 1))

其中,PIDHASH _ SZ 是表中元素的个数,表中的元素是指向 task _ struct 结构的指针。pid _ hashfn 为哈希函数,将进程的 pid 转换为表的索引,通过该函数,可以将进程的 pid 均匀地散列在它们的域中。

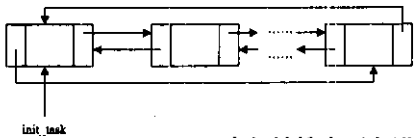
如果知道进程号,可以通过 hash 表很快地找到该进程,查找函数如下:

static inline struct task _ struct *find _ task _ by _ pid(int pid)

```
{
  struct task_struct *p, **htable = &pidhash[pid_hashfn(pid)];
  for(p = *htable; p&&p->pid != pid; p = p->pidhash_next)
    ;
  return p;
}
```

4.2　双向循环链表

哈希表的主要作用是根据进程的 pid 可以快速地找到对应的进程,但它没有反映进程创建的顺序,也无法反映进程之间的亲属关系,因此引入双向循环链表。每个进程 task_struct 结构中的 prev_task 和 next_task 成员用来实现这种链表。链表的头和尾都是 init_task(即 0 号进程)。



通过宏 for_each_task 可以方便地搜索所有进程;

```
#define for_each_task(p)
    for(p = &init_task; (p = p->next_task) != &init_task;)
```

4.3　运行队列

当内核要寻找一个新的进程在 CPU 上运行时,一般只考虑那些处于可运行状态的进程,因为查找整个进程链表效率是很低的,所以引入了可运行状态进程的双向循环链表,也叫运行队列。

运行队列容纳了系统中所有可以运行的进程,它是一个双向循环队列,该队列通过 task_struct 结构中的两个指针 run_list 链表来维护。队列的标志有两个:一个是"空进程"idle_task,一个是队列的长度。

空进程是一个比较特殊的进程,只有系统中没有进程可运行时它才会被执行,Linux 将它看作运行队列的头,当调度程序遍历运行队列时,是从 idle_task 开始、到 idle_task 结束的。

队列长度表示系统中处于可运行状态的进程数目,用全局变量 nr_running 表示,在/kernel/fork.c 中定义如下:

```
int nr_running = 1;
```

若 nr_running 为 0,表示队列中只有空进程。

(下转第 141 页)

(上接第 29 页)

(1):节点编号,表示该分析表记录的信息主要是属于哪个节点的;

(2)该节点在同类树型结构中出现的次数;

(3)子女节点的集合,包含了该节点的所有子女节点和在该类树型结构中出现过的节点;

(4)记录每个子女节点出现的次数;

(5)统计相应节点出现的概率,形成规则的基本库。

仍以图 3 为例,编制各个节点的分析表,以表 1 ~ 5 所示。

表 1　A 节点分析表

节点编号:A(A)					
出现次数	2				
子女节点编号	B	C	D	E	F
出现次数	2	2	2	2	1
概率	1	1	1	1	0.5

表 2　B 节点分析表

中间节点编号:B(B)			
出现次数	2		
子女节点编号	G	H	I
出现次数	2	1	1
概率	1	0.5	0.5

表 3　C 节点分析表

中间节点编号:C(C)		
出现次数	2	
子女节点编号	H	I
出现次数	1	1
概率	0.5	0.5

表 4　E 节点分析表

中间节点编号:E(E)			
出现次数	2		
子女节点编　号	J	K	M
出现次数	1	2	2
概率	0.5	1	1

表 5　F 节点分析表

中间节点编号:F	
出现次数	1
子女节点编号	J
出现次数	1
概率	1

4　规则产生

最后通过对节点分析表的汇总可形成规则判定树,供今后使用,规则判定树如图 4 所示。

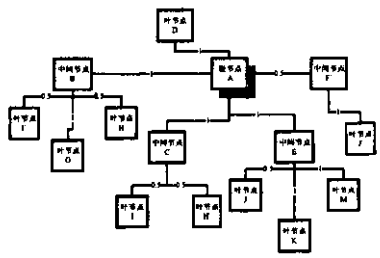


图 4　规则判定树

5　总结与展望

通过本方法的实施,配合相当的工程实例,本人在专家系统的前期规则整理中,已经发现一些具有相当指导和参考意义的规则。甚至,某些规则的深度已经到达或在一些方面超过部分与行业专家沟通获得的规则。因此,该方法具有相当的实用性。

树型结构是一个常用的数据结构,在数据库的实现中,大量的应用实例采用树型结构对现实世界建模。在积累了相当的数据后,如要对其进行分析和提炼规则,则必定会遇到类似的问题。希望本文所描述的方法会对基于树型结构的规则获取有一定的帮助,使数据挖掘技术能够在更广泛的领域内被应用,提炼出更多有效的规则,建立起更为完善的专家系统,来提高我们的工作效率。

参　考　文　献

[1] Joseph Giarratano 等,Expert Systems Principles and Programming(Third Edition),PWS publishing Company,2002 年 8 月.

[2] 蔡自兴,人工智能及其应用,清华大学出版社,1996 年 5 月.

[3] JiaWei Han 等,数据挖掘概念与技术,Morgan Kaufmann,机械工业出版社,2001 年 8 月.

[4] Abraham Silberschatz 等,数据库系统概念,McGraw-Hill,机械工业出版社,2000 年 2 月.

[5] David M. Kroenke,数据库处理基础、设计与实现,Prentice Hall 出版公司,电子工业出版社,1998 年 5 月.

库的辅助。在知识库的帮助下,VDBMS 调用相似检索匹配模块将查询特征与特征库中的人脸特征按一定的匹配算法进行相似匹配,满足一定相似的一组人脸图像从数据库中提取出来,按相似度大小排列返回给用户,对系统返回一组满足初始特征的查询结果,用户可以通过遍历(浏览)来选到满意的结果,或者从候选结果中选择一个示例,经特征调整,然后形成一个新的查询。逐步缩小查询范围,直到用户满意查询的结果为止。

当用户对所要查找的目标并不十分明确时,可对视频数据进行快速浏览以便寻找感兴趣的内容。浏览的目标是要跳过一些次要内容,以比较少的图像尽可能多地表达视频数据的主要内容,智能监控视频快速浏览可采用浏览关键帧的方法,并提供方便直观的视频浏览界面。

计算机人脸识别技术也就是利用计算机分析人脸图像,进而从中提取出有效的识别信息,用来"辨认"身份的一门技术。计算机人脸识别技术是近 20 年才逐步发展起来的,90 年代更成为科研热点,许多人致力于人脸自动识别技术的研究,例如,Poggio 和 Brunelli 用改进的积分投影法提取出用欧氏距离表征的 35 维人脸特征矢量用于模式分类,该系统获得了 90% 以上的识别率;1991 年 Turk 和 Pentland 提出了一种人脸识别方法,也就是众所周知的特征脸法,该方法在 16 个被测试对象共 2500 幅图像的人脸图像数据库中的正确识别率达到了 96%,该数据库中的人脸图像包括了不同光照、尺度、头部方向等;1991 年 Nakamura 提出一种等灰度线的人脸识别方法,该方法利用等灰度线图来表示人脸图像,并使用 Sobel 算子和其他后处理方法来获得人脸图像的边缘线,该方法在对不同人脸图像进行测试时,据称达到了 100% 的识别精度[2]。

智能监控视频数据库系统中可进行以下查询:

(1) 查找某日某时间范围内的监控视频记录;

(2) 查找有人进入监视范围的视频记录片断;

(3) 查找有巨大噪声的视频片断;

(4) 追踪搜索到数据库中某个特定人的全部时间和日期信息以及累计看到的次数;

(5) 寻找某一特定对象出现的所有帧;

(6) 查找某时间段出现的所有人像。

4　基于内容的智能监控视频数据库数据模型

视频数据模型是设计基于内容查询视频数据库的关键环节,视频数据由于其本身的综合性、结构复杂性,要用一个恰当的数据模型把现实世界的视频反映到信息世界及机器世界,是一个十分复杂的问题。传统的数据库系统的数据模型(如关系模型、层次模型、网状模型)无法直接应用于视频世界,即使对其作某些改进、补充,也难以适应视频数据这类复杂的数据类型。目前还没有表示视频数据模型的经典完善的理论及技术。在建立视频模型时,虽然可能借助其他方面的研究成果,但表示视频数据的统一理论及方法还没有形成。

本文基于文[3]中提出的分段模型,根据智能监控视频数据的特点,提出了一种基于分段的层次视频数据模型。

设 V 是视频数据模型,结构如下:

V 视频区间[oid,t_b,t_e]

n 特征个数

(F_1,F_2,…,F_n) 特征

其中,oid 表示区间标识符,t_b,t_e 分别表示起始帧与终止帧。

5　结束语

文章研究了基于内容的智能监控视频数据库的系统结构,并提出了相应的视频数据模型,提出了将视频分成背景图像和运动图像的分别存储入库的思想,可使智能监控数据库的数据压缩比得到质的提高。

在研究中笔者感到,视频数据库系统无论在技术上还是在理论上都远未成熟。它所涉及的理论和技术众多,如数据理论、图像处理、计算机视觉和人工智能。其中有许多技术还处于实验阶段,许多问题有待于解决,但作为数据库技术发展的最新技术,虽然只是起步,然而其前景是辉煌的,随着问题的逐步解决,视频数据库系统将在安全监控以及信息社会中发挥重要的作用。

参　考　文　献

[1] 周洞汝、胡宏斌,视频数据库管理系统导论[M],北京:科学出版社,2000 年,pp. 13 ~ 14.

[2] 凌旭峰,"彩色图像监控系统的人脸检测和识别[D]",《上海交通大学博士论文》,2001.

[3] Arun Hampapur. Design Video Data Management System. Ph. D thesis [D],1995.

(上接第 61 页)

4.4　等待队列

运行队列链表将所有状态为 TASK _ RUNNING 的进程组织在一起。将所有状态为 TASK _ INTERRUPTIBLE 和 TASK _ UNINTERRUPTIBLE 的进程组织在一起而形成的进程链表称为等待队列。

进程必须经常等待某些事件的发生,等待队列实现在事件上的条件等待,希望等待特定事件的进程将自己放进合适的等待队列,并放弃控制权。等待队列表示一组睡眠的进程,当条件满足时,由内核将它们唤醒。

等待队列由循环链表实现。在 2.4.0 版本中,等待队列的定义如下:

```
struct _ wait _ queue
{  unsigned int flags;
     struct task _ struct *task;
     struct list _ head task _ list;
}
```

5　结束语

进程是操作系统中一个非常重要的概念,了解进程的结构和组织方式对于我们了解操作系统的工作原理和实现机制具有非常重要的意义。Linux 是一个开放源代码的操作系统,Linux 的出现为我们深入了解操作系统的工作原理和实现机制提供了一个重要的机遇,我们一定要抓住这个机遇,为开发具有自主知识版权的操作系统而努力。

参　考　文　献

[1] 李善平、刘文峰等,Linux 内核 2.4 版源代码分析大全,北京:机械工业出版社,2002 年.

[2] 陈莉君,深入分析 Linux 内核源代码,北京:人民邮电出版社,2002 年.

[3] 马季兰、彭新光,Linux 操作系统,北京:电子工业出版社,2002 年.

Linux进程结构及组织方式研究

作者： 殷联甫， 沈士根， 郭步， Yin Lianfu， Shen Shigen， Guo Bu
作者单位： 嘉兴学院信息工程学院,浙江,嘉兴,314001
刊名： 计算机应用与软件 ISTIC PKU
英文刊名： COMPUTER APPLICATIONS AND SOFTWARE
年，卷(期)： 2005，22(11)
被引用次数： 0次

参考文献(3条)

1. 李善平. 刘文峰. 李程远 Linux内核2.4版源代码分析大全 2002

2. 陈莉君 深入分析Linux内核源代码 2002

3. 马季兰. 彭新光 Linux操作系统 2002

相似文献(10条)

1. 学位论文 王京辉 基于Linux操作系统的进程研究及其应用 2001

该文首先简单介绍了Linux和Linux的内核,然后从Linux内核在操作系统中的位置,Linux内核的抽象结构和内核中各个子系统之间的依赖关系,详细分析了Linux的内核结构;该文的重点在于对于进程的管理分析,首先介绍了进程的相关概念,然后分析了在Linux下的进程具体实现结构,包括进程的数据结构,多处理器系统中的调度和时钟和定时器,并且在实现中涉及了和进程有关的存储管理部分和进程之间的通信部分;线程是小的进程,该文介绍了线程的概念,实行和创建,LinuxThreads线程库和线程通信等等.最后通过对Linux内核相关部分的修改,完成了在特殊用途下的进程实现实际应用.该文的主要工作如下:该文的创新在于经过详细的分析Linux的进程机制,并且从进程的高度讨论了多线程的应用之后,在现有的硬件和软件无法更改的情况下,但是要求有更多的任务同时执行的时候,也就是要求有更多的进程同时工作,通过修改内核关于进程的管理的部分来实现进程数量的增大,完成相应进程数量的需要.突破默认最大进程数的限制.在这一问题中必须解决没有足够的gdt表项的问题.Gdt的大小是硬件限制的,该文的通过动态地设置进程的描述符,取消为进程预先分配空间的做法,内核中可以动态地寻址到每个进程的tss和ldt段,因此在任务切换时不再由于Linux进程数的限制而拒绝服务,使用给该方法可以突破对于512个进程的限制.其应用已经在上海环境检测系统的局域网中应用,采用这样的做法可以节约开支,经济有效.

2. 期刊论文 胡桃成. 曹俊彬. 左国存. 莫林. Hu Taocheng. Cao Junbin. Zuo Guocun. Mou Lin 从进程切换看Linux进程管理 -中国水运（理论版）2006,""(3)

事件驱动模型是优秀的系统分析和开发方法,其本质在于从外部世界与系统的动态交互中把握系统.本文试图用这种方法通过研究进程切换的过程来分析Linux对进程的管理.

3. 期刊论文 屈志强. 乔静. 纪爽. Qu Zhiqiang. Qiao Jing. Ji Shuang Linux操作系统进程实验项目设计与分析 -中国教育技术装备2010,""(6)

进程是操作系统的重点和难点,抽象、复杂、难以理解.实验教学是操作系统教学的重要组成部分,可以帮助学生更好地学习进程.但由于操作系统本身的复杂性,只有进行合理的设计才能使实验达到预想的效果.根据操作系统教学的要求、进程的特点,结合教学实践,设计并汇总与进程相关的实验项目,并对实验结果进行分析,对学生的学习有着积极的作用.

4. 期刊论文 高斌. 宗光华 基于Linux的预先创建子进程池的服务器程序设计 -计算机应用研究2002,19(2)

分析了服务器程序的通用设计方法的缺点,并介绍了一种在Linux操作系统下预先创建子进程池的服务器程序设计方法,在原理上进行了详细的讨论;最后给出了一个笔者实现的原型系统的主程序结构框架.

5. 学位论文 武华北 Linux进程间的通信机理及其在PC集群中的应用 2004

众所周知,操作系统是连接计算机硬件与上层软件及用户的桥梁,它的安全性是至关重要的.虽然我们不能说Linux一定比Windows更安全,但与封闭源代码的Windows相比,开放源代码的Linux系统可以让我们方便地分析其源代码,以找出其中的不足并加以修改,而不是像Windows那样,用户只能被动地接收微软公司的安全补丁.另外,从国家安全的角度来讲,中国显然不能过多的依赖某些西方国家,能拥有自己的操作系统一直是国人努力的方向,而开放源代码的Linux给了我们这个机会.有迹象表明,开放源代码的Linux操作系统将会在中国大有作为,所以我们需要向人们推介它,使更多的人了解Linux.对于计算机专业人员来说,更要了解和掌握Linux的内核源代码,这样才能使得我们的系统开发以及提高操作系统的效率和安全性等工作更具针对性和更加有效.对于多任务、多用户的操作系统来说,进程间通信是非常重要的,它是系统稳定运行的基础,而信号又是作为Linux最基本的进程通信机制.该文从数据结构入手,结合主要的功能函数,重点分析了信号通信机制在Linux中的具体实现过程.同时,Linux提供的强大的网络功能,也使它成为了PC集群系统的主流操作系统.MPI作为当前主流的并行编程工具之一,日益受到广大用户的青睐,它主要是通过借助操作系统提供的socket机制来实现具体的通信功能.该文着重分析了作为分布式进程间的通信手段的socket通信机制在Linux中的实现,以及作为通用的分布式通信手段socket在PC集群系统中的不足之处.

6. 学位论文 赵丰 基于Linux内核的多进程应用程序的开发 2007

本文讲述了Linux内核有关多进程应用程序设计所需进程、信号、共享内存、信号量、消息队列及网络套接字等基本概念。结合在Linux平台上开发的工作经验，总结出了Linux应用层开发的进程管理、进程通信和网络套接字的模型，实现了多进程对共享区一写多读的同步与互斥、环形缓冲和网络分发的设计。提出了一个基于Linux内核的多进程应用程序的解决方案。

本文在设计上比较实用，先讲述相关的概念，然后根据设计需求及工作经验总结出一般的设计模型，并在具体的实现上有机的将这些模型结合起来，实现最终的设计。同时为相关开发人员提供一个很好的参考。

7. 期刊论文 杨宪泽. 腾建旭. 撒晓英. 宋建成 进程及Linux进程探讨 -西南民族大学学报(自然科学版)2004,30(6)

第一部分讨论了为什么需要进程;第二部分探讨了进程的特征(动态性、并发性、独立性、异步性和结构性),以及特征的状态、实体和调度;第三部分论述了Linux进程.

8. 学位论文 巫晓明 Linux下Capabilities安全机制的分析与改进 2004

Capabilities(本文译为“权能”)是Linux操作系统对特权操作进行访问控制的一种安全机制，POSIX.1e标准草案中有对它的具体描述。

在老版本的Linux操作系统中，进程的特权操作是根据进程的有效用户身份来进行访问控制的。即以root为有效用户身份的进程可以进行任何特权操作，而其它用户的进程则不能进行任何特权操作。因此，在Linux系统中以root为有效用户身份的每个进程，不管需不需要，都拥有系统所有的特权。而在大多数情况下，进程完成其服务功能时并不需要什么特权，或仅仅需要很少的几种特权。拥有多余特权对系统安全总是不利的。

Capabilities就是针对上面的问题而提出来的，目的在于实现一种更加细粒度的特权操作的访问控制，使用户能够根据实际的安全需要来控制一个进程拥有的特权范围，从而能够取消进程的多余特权及多余特权带来的安全隐患。Capabilities的主要思想是：将root用户拥有的所有特权分割成多种权能，每种权能代表进行某种特权操作的权限；在系统所有实体中，只有进程和可执行文件具有权能的信息；系统根据一定的策略赋予进程权能，并根据进程拥有的权能来进行特权操作的访问控制。

Linux从内核版本2.1起就开始支持Capabilities。但是，由于文件系统的制约，Linux没有实现可执行文件的权能。因此，用户还不能根据实际的安全需要

来具体设置可执行文件的权能，从而控制相应进程拥有的权能范围。

本文主要阐述了我们对Linux下的Capabilities所作的改进与完善，使用户能够方便地利用Capabilities机制来提高Linux系统的安全。我们主要从两方面对Capabilities进行改进与完善。一是实现了可执行文件的权能，使用户可通过配置可执行文件的权能来灵活控制相应进程拥有的权能范围；二是对系统赋予进程权能的策略(即进程权能计算公式)进行了改进，从而实现了在只有一种权能配置的情况下，不同用户的进程在执行同一程序时可以拥有不同的权能。另外，我们还在Linux下实现了基于程序(进程)鉴别的ACL(AccessControlList)文件访问控制，使用户能够对可执行文件的权能配置信息以及其它系统资源提供更好的保护。同时，为了方便用户以“最小权限”的原则来配置可执行文件权能，我们提供了获取进程所需最小权能集的方法与途径。本文最后讲述了如何使用改进后的Capabilities来提高Linux系统的安全性。

9.期刊论文 张海.ZHANG Hai 一种LINUX操作系统守护进程的编程实现方法 -广东水利电力职业技术学院学报2006,4(2)

针对LINUX环境下的守护进程daemon,通过对一般性守护进程编写的分析,提出若干见解,从而使之简化且更为实用,并给出具体的实现程序.

10.学位论文 邹治锋 基于Linux的进程调度算法的改进与实现 2006

随着Linux系统的逐渐推广，它被越来越多的计算机用户所了解和应用，各国政府都在鼓励和支持Linux在本国的发展。O(1)调度算法推出后，使调度器的时间复杂度从O(n)降到了O(1)，又激起了人们对基于Linux的进程调度算法的研究。

本文首先对Linux系统进行了简要的介绍，介绍了国内外在调度算法方面的研究现状，分析了课题研究的背景和意义。然后介绍了进程管理的相关理论知识。

其次，详细研究了O(1)调度算法及其在Linux2.6内核中的具体实现。O(1)调度器中增加了数据结构runqueue，就绪队列被分成active和expired，结合bitmap[]不必遍历整个就绪队列，查找next进程的时间复杂度降为O(1)。进程运行时间片的重新分配更及时；动态优先级的计算过程更简单，计算时机更分散；调度时机更宽松，提高了调度器的实时性能；调度流程更简单。也指出了，在O(1)调度器下，不同用户执行相同的程序，他们创建的进程将获得相同的运行时间片和周转时间。这样对高级别用户是不公平的。

最后，引入了用户级别的概念，证明了O(1)调度器的以上不足。提出了一种基于用户级别的进程调度策略，通过给不同用户指派不同的级别，使不同级别用户创建的进程的时间片不同，以此使他们的周转时间不同，高级别用户将获得比低级别用户更短的周转时间，使各级别的用户得到与其级别相适应的周转时间，更加体现进程调度的公平性原则；并且，一般情况下，该调度策略下O级用户的周转时间比在O(1)调度策略下短。

O(1)调度器的研究紧跟了调度算法研究领域的前沿；用户级别的引入、基于用户级别的进程调度策略的提出，开拓了进程调度研究的新思路，一定程度上促进了我国自主操作系统的研究和发展。