

# Linux 内核系统调用扩展研究

张步忠<sup>1</sup>, 金海平<sup>2</sup>

(1. 安庆师范学院 计算机系, 安徽 安庆 246011;

2. 安庆师范学院 数学系, 安徽 安庆 246011)

**摘 要:** 系统调用是操作系统内核提供给用户使用内核服务的接口。Linux 操作系统由于其自由开放性, 用户可在原有基础上, 添加新的系统调用, 以便提供更多的服务。基于 Linux 2.4 内核, 文中研究了 Linux 操作系统系统调用的实现机制, 并以基于数据库的文件系统功能扩展为例, 说明系统调用扩展的实现过程。

**关键词:** Linux; 系统调用; 扩展

**中图分类号:** TP316.8

**文献标识码:** A

**文章编号:** 1673-629X(2007)05-0163-03

## Research of Linux Kernel System Call Expansion

ZHANG Bu-zhong<sup>1</sup>, JIN Hai-ping<sup>2</sup>

(1. Department of Computer, Anqing Teachers College, Anqing 246011, China;

2. Department of Mathematics, Anqing Teachers College, Anqing 246011, China)

**Abstract:** The system call is a user interface, which the operating system kernel provides to users using kernel service. Because of Linux operating system freedom and openness, users may expand new, specific application system call as the original foundation, which can provide to us more services. In this paper, based on Linux 2.4 kernel, realization mechanism of Linux system call is introduced. A case of function expansion of file system based on database is used to demonstrate the realization process.

**Key words:** Linux; system call; expansion

### 1 系统调用

现代计算机系统一般出于安全性和系统保护等方面的需要, 划分出不同的操作权限。如 Intel x86 系列从硬件上划分了 4 个不同的操作等级: 0 级最高, 可拥有所有的系统操作权限, 3 级最低, 操作是受约束的, 特别是一些特权指令是不能运行的, 想要运行运行这些指令也只有转到更高的权限级别。一些操作系统根据需要, 如 Unix 操作系统<sup>[1]</sup>也划分了用户和内核两个不同的操作级别, 将进程管理、内存管理、设备管理等关键性的核心功能放在内核级, shell 接口、用户程序等放在用户级运行。并把两者放在不同的地址空间, 用户和内核空间是不能直接通信和相互调用程序的, 用户也不能直接访问硬件, 从而防止了一部分低手段的用户程序侵害, 提高了安全性。

但是很多情况下, 需要内核程序提供服务, 如用户登录需要创建一个新的会话进程。为了既不破坏这种

机制, 又能让用户使用内核控制的资源, 操作系统提供了一种用户使用内核服务的机制, 即系统调用。系统调用函数通常由用户进程在用户态下调用, 内核通过 system\_call 函数响应系统调用产生的软中断, 在正确访问核心栈、系统调用开关表之后陷入到操作系统内核中进行处理。

基于此, 文中主要讨论了源代码开放的 Linux 环境下的系统调用实现机制, 并针对基于数据库的文件系统的用户接口扩展的实际需要, 介绍如何扩展内核系统调用。

### 2 Linux 系统调用实现过程

Linux 操作系统采用类似 Unix 内核方式, 把操作系统也分成系统状态和用户状态<sup>[2]</sup>。

Linux 下每个进程都有 4GB 的虚拟空间, 但这 4GB 的空间分成两部分: 0~3GB 部分称为用户空间, 它是用户进程的私有空间; 3~4GB 成为内核空间, 由所有进程及内核共享使用。

Linux 系统调用由两部分组成: 内核函数和接口函数。接口函数是提供给用户态的应用程序接口, 一

收稿日期: 2006-08-12

基金项目: 安徽省高校青年教师科研计划资助项目(2006jq210)

作者简介: 张步忠(1980-), 男, 安徽无为, 讲师, 硕士, 研究方向为操作系统、中间件技术。

般以 C 形式的库函数给出。内核函数是 Linux 内核中设置了一组用于实现各种系统功能子程序,并将它们提供给用户调用。在用户需要访问系统内核管理的资源时,用户进程将被挂起,接口函数将转到内核态,内核检验用户请求的合法性,尝试执行内核函数,并把结果反馈给用户进程,然后用户进程重新启动。

Linux 中每个系统调用都有一个唯一的编号,称为系统调用号<sup>[3]</sup>,这也是系统调用在入口地址表中位置的序号。

在用户程序调用系统调用时,系统调用程序通过 80H 中断指令,穿过硬件的陷阱门,由用户态切换到系统态,到达所有系统调用的总入口程序 system\_call,在此,保存系统调用号,检查系统调用号是否“超标”,然后在系统调用跳转表(sys\_call\_table[])中找到相应内核函数指针,通过寄存器,将用户入口参数传入,跳转到相应函数执行。执行完毕后到达 ret\_from\_sys\_call,恢复用户使用系统调用的现场,回到用户程序中,调用返回值仍通过寄存器传到用户空间。

相关代码和具体过程如下:

```
ENTRY(system_call) # arch/i386/kernel/entry.S
    pushl %eax # save orig_eax
    SAVE_ALL
    GET_CURRENT(%ebx)
    testb $0x02,tsk_ptrace(%ebx) # PT_TRACESYS
    jne tracesys
    cmpl $(NR_syscalls),%eax
    jae badsys
    call *SYMBOL_NAME(sys_call_table)(,%eax,4)
    movl %eax,EAX(%esp) # save the return value
ENTRY(ret_from_sys_call)
    cli
    cmpl $0,need_resched(%ebx)
    jne reschedule
    cmpl $0,sigpending(%ebx)
    jne signal_return
restore_all:
    RESTORE_ALL
```

调用宏过程“SAVE\_ALL”保存现有通用寄存器值,寄存器值的压栈不但可以保存系统调用前寄存器中的数据,而且提供了一种传递参数的方法(%eax 存放系统调用号,%ebx,%ecx,%edx,%esi,%edi 分别为参数 1~5,故系统调用最多可带 5 个参数)。堆栈中的结构,与该过程所要传递的 pt\_regs 结构类型的参数结构一致。在该宏中,还使 ds 和 es 指向内核的数据段,使 fs 指向用户的数据段,使进程进入核心态。

GET\_CURRENT(%ebx)宏调用是取得当前进程

的 task\_struct 结构,寄存器 %ebx 存放 task\_struct 结构的指针。

“testb \$0x02,tsk\_ptrace(%ebx)”检测当前进程是否正跟踪系统调用,如果不是,直接调用该系统调用函数,执行完后直接返回。

然后检查 %eax 中的系统调用号是否超出范围。/include/linux/sys.h 中定义为 260 个。如果是正常的系统调用,后面的 call 指令以 sys\_call\_table 为基地址,eax 寄存器中的内容(即系统调用的序号)乘以 4 为偏移量,即得到所需调用的系统调用函数的入口地址,调到相应的系统调用,执行系统调用。

执行完毕,movl %eax,EAX(%esp)将服务程序执行的返回值存在 %eax 中,通过 %eax 传到用户空间。

函数返回以后,流程进入 ret\_from\_sys\_call,该过程内处理一些系统调用返回前应该处理的事情,如检测 bottom half 缓冲区,判断 CPU 是否需要重新调度等,之后,系统调用返回。“RESTORE\_ALL”是 system\_call 的退出点,到此调用宏,使进程离开核心态,恢复各寄存器值并返回到用户调用时状态。

### 3 扩展系统调用

#### 3.1 系统调用扩展

了解系统调用的原理和过程后,就可以增加、修改自己的系统调用了,尤其在实际应用中,需要增加或完善系统功能的时候。编写新的内核函数,在系统调用跳转表中增加新的表项,再重新编译内核,使新的系统调用生效,用户程序即可调用了。

分析 Linux 源代码发现,在 x86 体系下,Linux 的系统调用也是通过 0x80 中断指令实现的,通过陷阱指令,从用户空间进入内核空间,涉及到的源代码有:

```
arch/i386/boot/bootsect.S
arch/i386/Kernel/setup.S
arch/i386/boot/compressed/head.S
arch/i386/kernel/head.S
init/main.c
arch/i386/kernel/traps.c
arch/i386/kernel/entry.S
arch/i386/kernel/irq.h
include/asm-386/unistd.h
```

增加系统调用真正要修改文件只有 include/asm-386/unistd.h 和 arch/i386/kernel/entry.S 两个。改动系统调用表,在 arch/i386/kernel/entry.S 中增加系统调用表项:

```
long SYMBOL_NAME(sys_new_call) /* 259, We use it. */
```

在 include/asm-386/unistd.h 中添加增加的系统调用号:



```
# define NR_new_call 259
```

将 new\_call() 函数定义的头文件 new\_call\_proc.h 加到 kernel/sys.c 中, 否则内核会找不到增加的系统调用。

编译新内核:

```
[root@zh root]# make clean //首先清除已生成的目标文件及其他文件
```

```
[root@zh root]# make xconfig //配置新内核
```

```
[root@zh root]# make dep //理顺各文件之间的依存关系
```

```
[root@zh root]# make bzImage //编译压缩的内核
```

```
[root@zh root]# make modules_install //编译模块
```

最后装入新内核, 编辑系统引导文件, 计算机能启动新内核。

### 3.2 基于数据库的文件系统中的系统调用扩展

文件系统解决了数据以文件的形式存储、文件在存储介质上的管理等问题。其中, 应用最广泛、影响最深的是树状层次结构的文件系统。但一直以来, 文件系统的文件存储方式、提供给应用程序的 API 接口却大大变化, 而存储的数据量却一直在增长。因此人们一直在致力于改善文件系统的存储性能和访问手段, 其中一种类型就是基于数据库的文件系统。

基于数据库的文件系统<sup>[4]</sup>(DBFS)综合了文件系统和数据库系统的优点, 在传统文件系统功能的基础上, 将数据库的数据管理功能引入到文件系统中。文件内容、文件属性都以表记录的形式存放在数据库中; 文件的存储格式统一, 如二进制数据文件、多媒体文件、流媒体文件统一存储; 可以使用元数据来描述文档的属性, 并可扩充、自定义文件的属性; 支持海量存储, 更强的并发控制与协同控制的能力, 基于内容访问等, 访问非常方便。

但是现有操作系统下的文件系统接口并不支持该类功能。为了使用户能有 DBFS 提供的新兴的强大功能, 并保留原有接口, 拟扩展 Linux 系统调用, 实现新功能的内核接口。

增加的系统调用如下:

```
asmlinkage long sys_dbfs_call(int which, char * file_path, char * u_comment)
```

```
{
    struct inode * inode;
    struct file * file;
    struct dbfs_server * server;
    struct dbfs_operations * dbfs_op;
    .....

    file = filp_open(path, O_RDONLY, 0); /* path 为 file_path
    的内核存在。找该文件 */
    error = PTR_ERR(file);
    if(IS_ERR(file)) goto out_error; /* 错误, 直接跳转到错误
```

```
出口 */
```

```
inode = file->f_dentry->d_inode; /* 找到该文件的 inode */
```

```
if ((server = &inode->i_sb->u.dbfs_sb.df_server) == NULL) goto out_error;
```

```
if ((dbfs_op = inode->u.dbfs_i.dbfs_op) == NULL) /*
取得 dbfs_operations 函数结构指针 */
```

```
goto out_error;
```

```
switch(which) { /* 判断用户的调用 */
```

```
case 0: /* check in */
```

```
retval = dbfs_op->check_in(user, svr_address, path, comment);
```

```
break;
```

```
.....
```

```
fput(file); /* 释放资源 */
```

```
out_error:
```

```
return error;
```

```
}
```

编译该内核, 并装入新内核, 编辑系统启动引导配置文件 /etc/lilo.conf, 使 LILO 能启动新内核。重新启动进入新内核, 编写一个用户程序<sup>[5]</sup>测试之:

```
syscall3(int, dbfs_call, int, which, char *, file_path, char *, u_comment)
```

```
main()
```

```
{
```

```
.....
```

在主函数 main 前必须申明调用 syscall, 其中 3 表示该系统调用只有 3 个入口参数, 第一个 int 表示系统调用的返回值为整型, dbfs\_call 为系统调用函数名, 第二个 int 表示入口参数的类型为整型, which 为入口参数名, 后面的依次类推。

## 4 结束语

文中介绍了现代操作系统的系统调用的原理和 Linux 下的系统调用的实现过程, 并以基于数据库的文件系统中的接口扩展需要, 详细介绍了如何在 Linux 下扩展新的系统调用。Linux 操作系统是单一内核, 因此扩展内核系统调用时, 要小心谨慎, 否则可能会导致整个内核崩溃无法使用。另外内核系统调用扩展后确实可以带来更多的方便, 但也导致与标准内核的不兼容性, 这就需要根据实际情况决定。

## 参考文献:

[1] Stallings W. 操作系统——内核与设计原理[M]. 第4版.

(下转第 169 页)

模块化思想,每个模块可以单独进行设计和实现,便于系统的更改与后期维护。

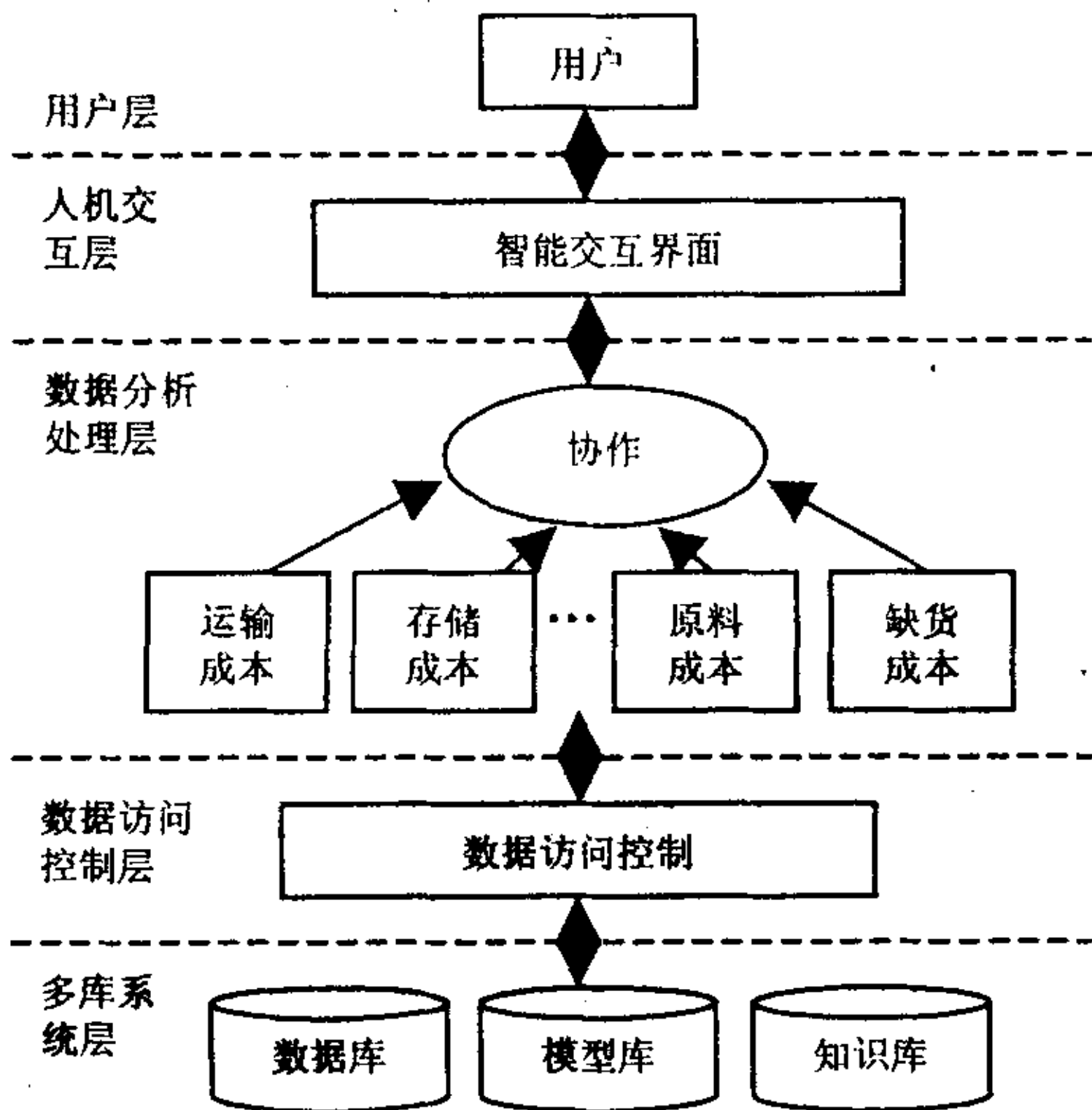


图 4 ERP 采购成本分析子系统层次模型

2.3 Multi-Agent 的协作与通信

在数据分析处理 Agent 中,由于各子 Agent 之间共享部分结果和全局目标,文中采用基于结果共享的协作方式(如图 5 所示)进行群体 Agent 协作求解。

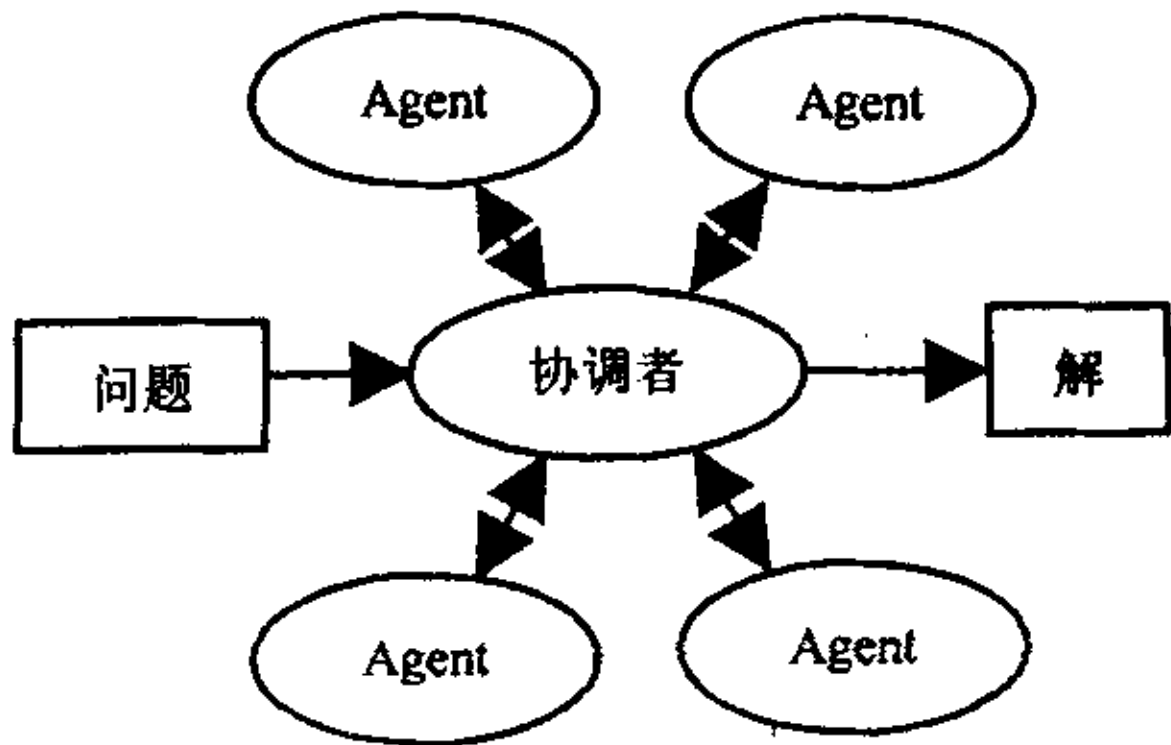


图 5 基于结果共享的协作方式

各子 Agent 分配到任务后,就当前环境状态下进行子问题求解,并把求解环境和结果反馈给协调者,由协调者进行判断是否达到全局目标和各子 Agent 之间是否存在冲突,存在冲突则由协调者根据相关规则进行协调,各子 Agent 在新环境下重新进行求解。

基于采购成本分析系统的特殊性,在智能交互界面 Agent、数据分析处理 Agent 和数据访问控制 Agent 之间采用点对点的直接通信方式;在数据分析处理子

Agent 之间采用基于黑板模式(Black-Board Mode)的广播通信,各子 Agent 之间通过黑板共享数据和结果信息<sup>[4,5]</sup>。

2.4 系统决策与效率分析

推理模块是智能决策支持系统的核心部分,推理模块通过多库的协同交互工作,对用户所要求的任务进行分析和推理,为用户提供决策支持和帮助。在模型库中存储了决策所需的各种模型。在 ERP 采购成本分析系统中,主要用到的模型有:回归模型、动态规划、聚类分析模型、模糊推理模型、神经网络、遗传算法等模型。

从系统运行结果来看,根据粗略计算,系统决策响应时间平均能提高 30% 以上,而且系统更趋稳定,后期维护也更方便和简单。

3 结束语

文中对具有大数据流的 MAS 建模进行研究,在 ERP 采购成本分析子系统的 Multi-Agent 建模中取得良好的效果,但数据分析处理子 Agent 之间的协作求解还需要进一步的研究,包括 Agent 之间的协作方式和冲突消解策略,这也是影响决策速度和决策准确性的重要方面。

参考文献:

[1] 杜玉强,王明哲.基于 Agent 的决策支持系统的构建[J].微机发展,2003,13(2):66-68.  
[2] Oragani A S, Giorgini P. Multi-agent Environment[C]//Lecture Note in Artificial Intelligence, ECAI'96 Workshop. [s.l.]:[s.n.],1996:103-116.  
[3] 威尔.采购与供应链[M].梅绍祖,阮笑雷,巢来春译.北京:清华大学出版社,2002.  
[4] 刘向军.MAS 通讯与协作机制构建[J].机械设计与制造工程,2002,31(3):21-23.  
[5] 李建民,石纯一.DAI 中多 Agent 协调方法及其分类[J].计算机科学,1998,25(2):9-12.

(上接第 165 页)

北京:电子工业出版社,2001.  
[2] 郭玉东.Linux 操作系统结构分析[M].西安:西安电子科技大学出版社,2002.  
[3] 毛德操,胡希明.Linux 内核源代码情景分析[M].杭州:浙江大学出版社,2001.  
[4] Oracle Ltd. Oracle Content Management Software Development Kit(SDK)Developer Reference Release 9.0.3[EB/OL].

2002. <http://www.oracle.com/technology/documentation/ifs.html>.  
[5] Rusling D A. Linux 编程白皮书[M].北京:机械工业出版社,2000.  
[6] Bach M J. Unix 操作系统设计[M].北京:机械工业出版社,2000.