hbrqlpf的专栏

个人资料

hbrqlpf

访问： 149994次
积分： 2319分
排名： 第5189名

原创： 62篇 转载： 93篇
译文： 0篇 评论： 35条

shell 原理

分类： Unix/Linux 2008-08-31 17:43 3757人阅读 评论(0) 收藏 举报

shell 脚本 unix command 数据结构 编程

1、进程

在Unix术语中，一个可执行程序是一个机器指令及其数据的序列，一个进程是程序运行时的内存空间和设置。

进程存在于用户空间，用户空间是存放运行的程序和它们的数据的一部分内存空间。

就像管理磁盘的多个文件，内核管理内存中的多个进程，为它们分配存储空间，并记录内存分配情况。

Unix系统中的内存分为系统空间和用户空间，进程存在于用户空间。建立一个进程时，内核要找到存放程序指令和数据的空闲内存页。内核还要建立数据结构来存放相应的内存分配情况和进程属性。

2、**shell**主循环

shell是一个管理进程和运行程序的程序，Unix系统有很多可用的shell，shell的主要功能：

（1）运行程序

（2）管理输入和输出

（3）可编程

shell主循环：

```
while(!end_of_input)
    get command
    excute command
        wait for command to finish
```

为了写一个shell，需要学会：

（1）运行一个程序

（2）建立一个进程

（3）等待exit()

2.1、一个程序如何运行另一个程序

exec系统调用从当前进程中把当前程序的机器指令清除，然后在空的进程中载入调用时指定的程序代码，最后运行这个程序。

```
int execvp (const char * file,const char * argv[])
```

execvp载入由file指定的程序到当前进程，然后试图运行它。execvp将以NULL结尾的字符串列表传递给程序。

2.2、如何建立新的进程

```
pid_t fork(void)
```

进程调用fork，当控制转移到内核中的fork代码后，内核做：

（1）分配新的内存块和内核数据结构

（2）复制原来的进程到新的进程

（3）向运行进程集添加新的进程

（4）将控制返回给两个进程

子进程不是从main函数的开始，而是从fork返回后的地方开始它的生命之旅。不同的进程，fork返回值是不同的。在子进程中fork返回0，在父进程中fork返回子进程号。根据fork的返回值可以判断自己是子进程还是父进程。

2.3、父进程如何等待子进程的退出

```
pid_t wait(int * statusptr)
```

系统调用wait做两件事。首先，wait暂停调用它的进程直到子进程结束。然后，wait取得子进程结束时传给exit的值。

文章存档

2009年01月 (1)
2008年11月 (1)
2008年10月 (15)
2008年09月 (12)
2008年08月 (6)
展开

阅读排行

HTML语法 (11584)
硬盘主引导记录(MBR)及 (7370)
JAVA数据类型转换 (4654)
shell 原理 (3746)
段式内存管理和页式内存 (3648)
C++ Primer 笔记 (3594)
MyEclipse + Tomcat + E (2910)
在bochs上运行grub引导 (2823)
VC 笔记 (2589)
Linux RPM 命令参数使用 (2577)

评论排行

JAVA数据类型转换 (6)
硬盘主引导记录(MBR)及 (6)
CPU 组织 (5)
HTML语法 (3)
C++ Primer 笔记 (2)
详解:数据库名、实例名、 (1)
段式内存管理和页式内存 (1)
MyEclipse + Tomcat + E (1)
const关键字用法 (1)
合理规划您的硬盘分区 (1)

推荐文章

最新评论

C++ Primer 笔记
mdchao2009: 好东西
硬盘主引导记录(MBR)及其结构详
poopooptt: very 感谢
MyEclipse + Tomcat + Eclipse 开
lennyf: 学习了，写的真详细啊
JAVA数据类型转换
dengyi21: 回复 jackpk：short s = i 时，有个 i 先转换为 int 的自动转换。
CPU 组织
zhaoliguang2011: 谢谢楼主详细说明！！
JAVA数据类型转换
wyit007: 认同贴主的写法！byte i=99;short s=i;（这种写法编译器是编译不过去的，是报错的）S...
基础shell编程
jqc2009: 好,非常好，
硬盘主引导记录(MBR)及其结构详
liuna915: LZ写的很好哈，很感谢。我有个问题想问下哈：主引导扇区和引导扇区有啥区别啊？文中说“其中的引导扇区是...
CPU 组织
Laic: 非常实用，感谢LZ

当子进程调用exit，内核唤醒父进程同时将子进程传给exit的参数。

父进程调用wait时传给一个整形变量地址给函数。内核将子进程的退出状态保存在这个变量中。如果子进程调用exit退出，那么内核把exit的返回值存放到这个整形变量中；如果进程是被杀死的，那么内核将信号序号存放在这个变量中，这个整数由3部分组成——8个bit是记录退出值，7个bit是记录信号序号，另一个bit用来指明发生错误并产生了内核映像。

wait系统调用挂起调用它的进程直到得到这个进程的子进程的一个结束状态。结束状态是退出值或者信号序号。如果有一个子进程已经退出或被杀死，对wait的调用立即返回。Wait返回结束进程的PID。如果statusptr不是NULL，wait将退出状态或者信号复制到statusptr指向的整数中。如果调用的进程没有子进程也没有得到终止状态值，则wait返回-1。

2.4、小结：**shell**如何运行程序

shell用fork建立新进程，用exex在新进程中运行用户指定的程序，最后shell用wait命令等待新进程结束。wait系统调用同时从内核取得退出状态或者信号序号以告知子进程是如何结束的。

3、**shell**编程

shell是一个编程语言解释器，这个解释器解释从键盘输入的命令，也解释存储在脚本中的命令序列。

3.1、**shell**脚本语言

shell脚本是一个包含一系列命令的文件，运行一个脚本就是运行这个文件中的每个命令。可以用一个shell脚本在一次请求中来执行多个命令。

shell脚本的执行：shell解释程序会fork+exec执行这个脚本命令，在exec调用中，内核会检查脚本的第一行（如：#!/bin/sh）找到来执行脚本的解释程序，然后装入这个解释程序，由它来解释执行脚本。

脚本中的元素：

（1）变量

（2）用户输入

（3）控制

（4）环境

3.2、**shell**中的流程控制

shell中的if语句作用和其他语言的if语句相同；条件检测。如果条件的值为正，则有一部分代码被执行。在shell中，条件是一个命令，返回正值意味着命令运行成功。

Unix程序以0退出表示成功，脚本中的if…then语句基于以0退出表示成功这个假设。

if语句是如何工作的：

（1）shell运行if之后的命令

（2）shell检查命令的exit状态

（3）exit的状态为0意味着成功，非0意味着失败

（4）如果成功，shell执行then部分的代码

（5）如果失败，shell执行else部分的代码

（6）关键字fi标识if块的结束

3.3、**shell**变量：局部和全局

shell包括两类变量：局部变量和环境变量。

3.3.1、**shell**变量的使用

shell变量的操作：

（1）赋值：var=value

（2）引用：$var

（3）删除：unset var

（4）输入：read var

（5）列出变量：set

（6）全局化：export var

变量的值是字符串，变量都是字符串类型的，没有数值类型的变量。所有的操作都是字符串操作。

3.3.2、变量的存储

要在shell中增加变量，必须有个地方能存放这些变量的名称和值，而且这个变量存储系统必须能够分辨局部和全局变量。

实现：

```
struct var{
    char * str;//name=val string
int global;//a Boolean
};
```

HTML语法

static struct var tab[MAXVARS];

源： shell

set是shell的一个命令，而不是一个由shell运行的程序，这就像if和then这些关键字由shell自己处理一样，为了将set与要执行的程序区分开，将set设置为内置的命令。

命令varname=value告诉shell在变量表里添加一项，赋值语句也是内置的命令。

3.4、环境变量

环境变量设置的值被许多程序使用，环境不是shell的一部分，但shell包括一些可以让用户读取和修改环境的命令

3.4.1、使用环境

（1）env：列出环境变量

（2）更新环境：var=value;export var

3.4.2、什么是环境以及它的工作机理

环境是每个程序都可以存取的一个字符串数组，数组中的每个字符串都是以var=value这样的形式出现，数组的地址被存放在一个名为environ的全局变量里。环境就是environ指向的字符串数组，读环境就是读这个字符串数组，改变环境就是改变字符串，改变这个数组中的指针或者将这个全局变量指向其他数组。

fork完整地复制父进程，包括代码和数据，数据中包括了环境。exec清除原来进程中的所有代码和数据，插入新程序的代码和数据。只有通过参数execvp传递的数据和存储在环境中的字符串可以从旧程序复制到新程序。

实现：

```
void setup()
{
   extern char ** environ;
   VLenviron2table(environ);//初始化shell，将环境中的变量添加到自己的变量列表中
}
if(pid=fork()==-1)
   perror(“fork”);
else if(pid==0){
environ=VLtable2environ();//将变量列表中的全局变量进行拷贝构建环境列表，执行命令前将environ指向新的列表
execvp(argv[0],argv[]);
}
```

4、**I/O**重定向和管道

4.1、标准**I/O**与重定向的若干概念

3个标准文件描述符：

标准输入（0:stdin）：需要处理的数据流

标准输出（1:stdout）：结果数据流

标准错误输出（2:stderr）：错误消息流

默认的连接——tty：

通常，shell命令行运行Unix系统工具时，stdin，stdout和stderr连接在终端上。因此，工具从键盘读取数据并把输出和错误消息写到屏幕。

通过使用重定向标志，命令cmd>filename告诉shell将文件描述符1定位到文件，于是shell就将文件描述符与指定的文件连接起来。重定向操作由shell来完成

文件描述符是一个数组的索引号，每个进程都有其打开的一组文件，这些打开的文件被保持在一个数组中。文件描述符即为某文件在此数组中的索引。

最低可用文件描述符原则：当打开文件时，为此文件安排的文件描述符总是此数组中最低可用位置的索引。

4.2、如何将**stdin**定向到文件

进程并不是从文件读数据，而是从文件描述符读数据，如果将文件描述符0定位到一个文件，那么此文件就成为标准输入的源。

4.3、为其他程序重定向**I/O**

在fork执行之后，子进程仍然在运行shell程序，并准备执行exec。exec将替换进程中运行的程序，但它不会改变程序的属性和进程中所有的连接。打开的文件并非是程序的代码也不是数据，他们属于进程的属性，因此exec调用并不改变它们。

shell使用进程通过fork产生的子进程与子进程调用exec之间的时间间隔来重定向标准输入、输出到文件。

（1）父进程调用fork

（2）子进程调用close(1)

（3）子进程调用create(“filename”,m)

（4）子进程使用exec执行新程序

4.4、管道编程

管道是内核中的一个单向数据通道，管道有一个读取端和一个写入端。

4.4.1、创建管道

int pipe(int array[2])

系统调用pipe来创建管道并将其两端连接到两个文件描述符。array[0]为读数据端的文件描述符，而array[1]为写数据端的文件描述符。像一个打开的文件的内部情况一样，管道的内部实现隐藏在内核中，进程只能看见两个文件描述符。类似于open调用，pipe调用也使用最低可用文件描述符。

4.4.2、使用**fork**来共享管道

当进程创建了一个管道之后，该进程就有了连向管道两端的连接。当这个进程调用fork的时候，它的子进程也得到了这两个连向管道的连接。父进程和子进程都可以将数据写到管道的写数据端口，并从读数据端口将数据读出。两个进程都可以读些管道，但是当一个进程读，另一个进程写的时候，管道的使用效率最高。

4.4.3、使用**pipe**、**fork**以及**exec**

pipe(p)

fork()

父进程：close(p[1])，dup2(p[0],0)，close(p[0])，exec"sort"

子进程：close(p[0])，dup2(p[1],1)，close(p[1])，exec"who"

shell管道的实现：

shell首先创建管道，然后调用fork创建两个新进程，再将标准输入和输出重定向到创建的管道，最后再通过exec程序来执行两个程序。

上一篇 后台执行命令

下一篇 文本过滤

主题推荐 shell 全局变量 数据结构 内存分配 流程控制

猜你在找

Nagios warning
linux下使用libiconv进行汉字编码问题的处理——
#《算法竞赛入门经典》勘误
linux shell 比较两个浮点数
awk提取一串字符中的数字
Speculative Execution in Hadoop
TCP是通过什么方式来提供可靠传输的
Linux网络编程之使用TCP传输文件
也谈哈希表
bash脚本，自动输入sudo的密码

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap