


深圳大学 计算机与软件学院

College of Computer Science and Software Engineering of Shenzhen University


# 系统编程

## 基于TaiShan服务器/openEuler OS 的实践

第一讲：课程介绍 & 进程控制 （下）

## 上一讲内容

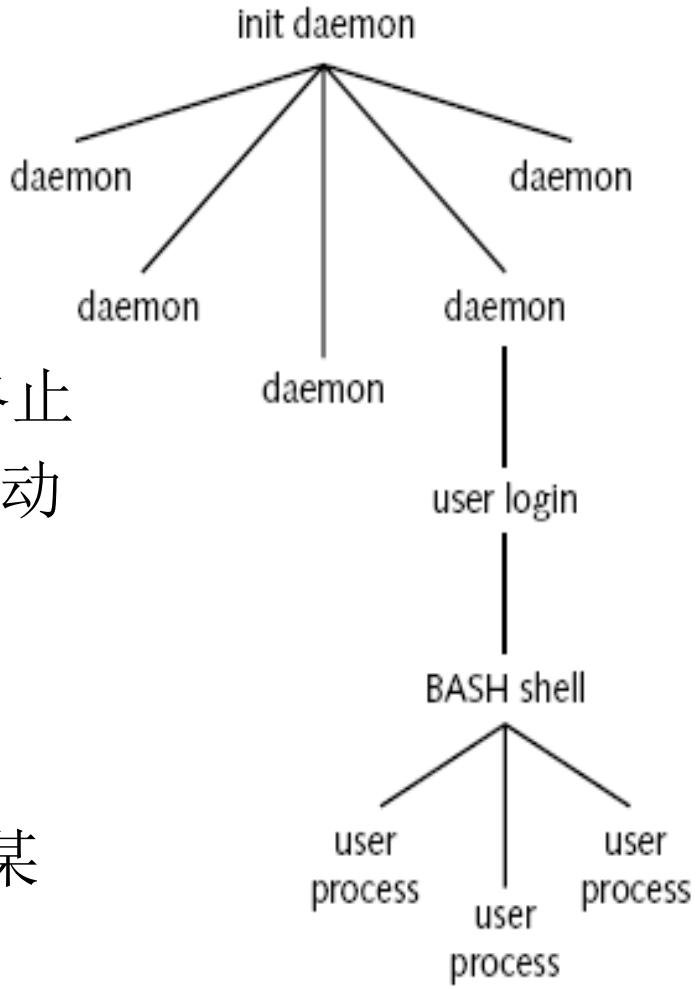
- 进程的基本概念
- 进程控制相关的系统调用

## 本讲内容

- 创建守护进程
  - 守护进程的概念
    - ✓守护进程、控制终端、进程组、会话
  - 如何编码实现守护进程
    - ✓守护进程特征
    - ✓参考实现源代码

# 守护进程

- 守护进程(Daemon)
  - 后台服务程序 - 系统服务
  - 进程名字往往以’d’结尾
  - 生存周期比较长
    - 系统装入时启动，关闭时候终止
      - 从启动脚本/etc/rc.d中启动
      - 由作业规划进程crond启动
    - 用户终端启动
    - 独立于控制终端
  - 周期性执行某种任务或等待处理某些发生的事件
  - 例子:httpd, ftpd, lqd,crond



# 思考题一 (并行熟悉Shell命令）

- 找出系统中名字以’d’结尾的进程

```
[szu@taishan02-vm-10 ~]$ ps -el | grep d$ | more
4 S     0       1       0  0  80   0 -  1853 -      ?        00:02:59 systemd
1 S     0       2       0  0  80   0 -     0 -      ?        00:00:02 kthreadd
1 I     0       6       2  0  60 -20 -     0 -      ?        00:00:00 kworker/0:0H-kblockd
1 I     0      10       2  0  80   0 -     0 -      ?        00:00:57 rcu_sched
1 I     0      18       2  0  60 -20 -     0 -      ?        00:00:00 kworker/1:0H-kblockd
1 I     0      23       2  0  60 -20 -     0 -      ?        00:00:00 kworker/2:0H-kblockd
1 I     0      28       2  0  60 -20 -     0 -      ?        00:00:00 kworker/3:0H-kblockd
1 S     0      31       2  0  80   0 -     0 -      ?        00:00:33 kauditd
1 S     0      33       2  0  80   0 -     0 -      ?        00:00:02 khungtaskd
1 S     0      37       2  0  85   5 -     0 -      ?        00:00:00 ksmd
1 S     0      38       2  0  99  19 -     0 -      ?        00:00:00 khugepaged
1 I     0      40       2  0  60 -20 -     0 -      ?        00:00:00 kintegrityd
1 I     0      41       2  0  60 -20 -     0 -      ?        00:00:00 kblockd
1 I     0      42       2  0  60 -20 -     0 -      ?        00:00:00 md
1 S     0      44       2  0 -40   - -     0 -      ?        00:00:00 watchdogd
1 I     0      99       2  0  60 -20 -     0 -      ?        00:00:00 kthrotld
1 I     0     107       2  0  60 -20 -     0 -      ?        00:00:00 kmpath_rdacd
1 I     0     108       2  0  60 -20 -     0 -      ?        00:00:00 kaluad
1 I     0     340       2  0  60 -20 -     0 -      ?        00:00:02 kworker/0:1H-kblockd
```

# 思考题二 (并行熟悉Shell命令）

- 找出系统中名字以’k’开头的进程

```
[szu@taishan02-vm-10 ~]$ ps  -el | sed 's/[ ] [ ]*/ /g' | cut -d ' ' -f14 | grep ^k | more
kthreadd
kworker/0:0H-kblockd
ksoftirqd/0
ksoftirqd/1
kworker/1:0H-kblockd
ksoftirqd/2
kworker/2:0H-kblockd
ksoftirqd/3
kworker/3:0H-kblockd
kdevtmpfs
kauditd
khungtaskd
kcompactd0
ksmd
khugepaged
kintegrityd
kblockd
kswapd0
kthrotld
```

# 控制终端

- 终端
  - Linux系统与用户进行交互的界面
- 控制终端
  - 从一个终端启动的进程及其后继进程都继承与该终端的连接，而该终端称为这些进程的控制终端。
  - 控制终端中断/退出，都会给连接到该终端的前台进程组进程发送相应的信号
  - 守护进程要避免不必要的干扰和交互

# 5步创建守护进程

① 创建后台孤儿进程

② 创建脱离控制终端的进程

③ 更改当前工作目录

④ 关闭所有继承的打开文件资源

⑤ 设置文件创建的预设权限

✓ 提高服务可用性

✓ 提高资源利用率

✓ 提高数据安全性

# ① 创建后台孤儿进程

- 创建子进程
- 父进程退出

```
pid = fork();

if (pid > 0) exit(0);
```

- ✓ 接下来的工作都由子进程执行
- ✓ 子进程为孤儿进程
- ✓ 子进程后台执行
- ✓ 用户可以在该shell终端执行其他的命令

# 思考 & 验证：经过第一步后与终端脱离了吗？

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(void)
{
        pid_t ret;
        int count = 0;

        if ((ret = fork()) > 0) exit(0);
        printf("count = %d \n", count);
        while (1) {};
        exit(0);
}
```

```
[szu@taishan02-vm-10 process]$ ./orphen
count = 0
[szu@taishan02-vm-10 process]$ ps -eo ppid,pid,pgid,sid,tty,tpgid,stat,command
| grep PID; ps -eo ppid,pid,pgid,sid,tty,tpgid,stat,command | grep orphen
   PPID     PID    PGID     SID TT         TPGID STAT COMMAND
 409204  428311  428310  409204 pts/3     428310 S+   grep --color=auto PID
      1  428288  428287  409204 pts/3     428312 R    ./orphen
 409204  428313  428312  409204 pts/3     428312 S+   grep --color=auto orphen
[szu@taishan02-vm-10 process]$
```

# 进程组和会话

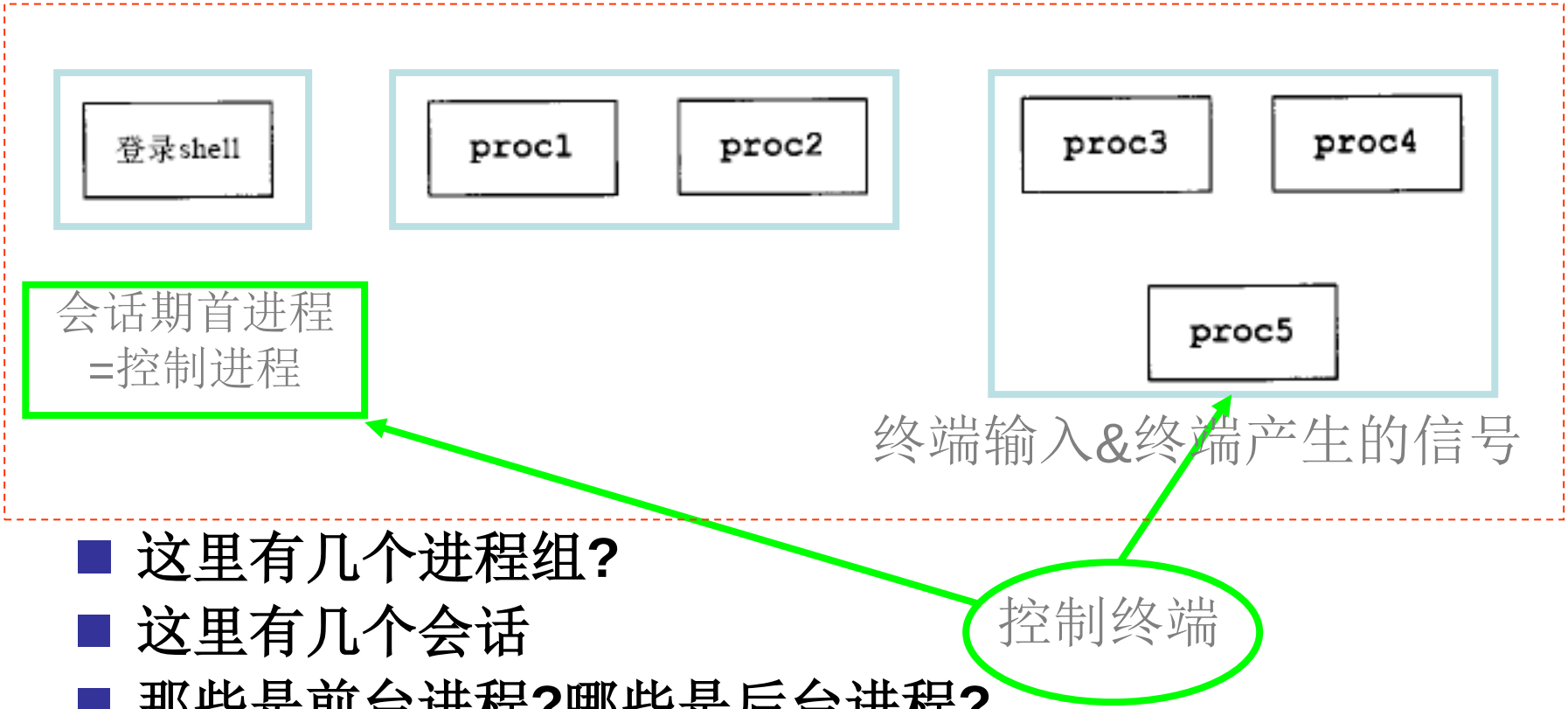
- **进程组**
  - Linux系统中，进程一定属于一个进程组
  - 进程组是一个或多个进程的集合
  - 每个进程组都有进程组Id
    - 进程组Id = 进程组组长的进程Id
  - 进程组组长退出后，进程组可继续存在
- **进程运行exec将脱离原来的进程组**

# 进程组和会话

- 进程组
- 会话
  - 一个或多个进程组的集合
  - 会话首进程：
    - 新建会话时，会话中唯一进程的ID = 会话ID (SID)
  - 一个会话连接一个控制终端
  - （默认）登陆SHELL连接控制终端并发起的会话
    - 登录SHELL成为会话首进程
    - 一个前台进程组
    - 多个后台进程组组成

# 进程组 & 会话

```
proc1 | proc2 &
proc3 | proc4 | proc5
```




- 这里有几个进程组?
- 这里有几个会话
- 那些是前台进程?哪些是后台进程?

# 进程组和会话

- 进程组
  - 一个或多个进程的集合
  - 作业控制
  - getpgrp() & setpgid()
- 会话
  - 一个或多个进程组的集合
  - 控制终端
  - setsid()

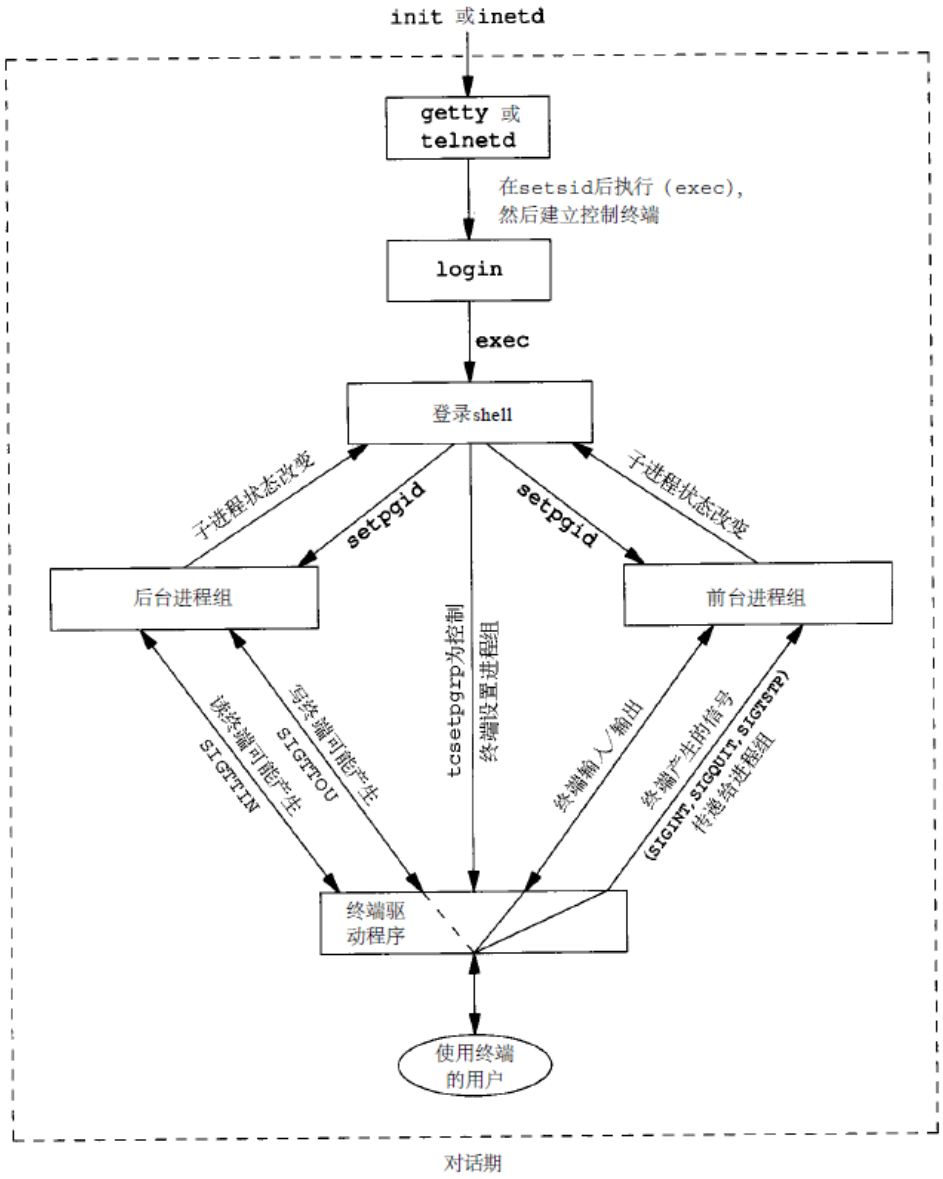



图9-8　对于前台、后台作业以及终端驱动程序的作业控制功能摘要

使用fork()创建子进程，继承父进程的

① 控制终端

② 会话

③ 进程组

守护进程必须创建新的会话

- 脱离父进程的影响

《Unix环境高级编程》
第九章

# setsid()

- **setsid()函数创建新的会话，**

  #include<unistd.h>

  pid_t setsid(void)

  - 调用进程成为新会话的首进程
  - 唯一的进程组
  - 进程组ID为调用进程的进程ID

- **成功返回的条件：调用进程不为会话首进程**
  - 第一步中调用fork的父进程退出，该子进程不可能成为进程组的领头进程

- **新的会话首进程脱离原控制终端的连接**

# ② 创建脱离控制终端的进程

- 调用**setsid()**创建新的会话
  - 创建会话的进程成为该会话的首进程
  - 脱离原来的控制终端
- 调用**fork()**创建子进程
- 会话首进程退出
  - ✓ 无法与控制终端建立连接

# ③ 更改当前目录

- 使用**fork**函数产生的子进程将继承父进程的当前工作目录
- 当进程没有结束，其工作目录不能被卸载
- 守护进程一般将其工作目录更改到根目录下

```
#include<unistd.h>
int chdir(const char *path);
```

# ④ 关闭所有继承的打开文件描述符

- 新产生的进程从父进程继承了打开的文件描述符，如果不使用，应该关闭它
- 守护进程是运行在系统后台的，不应该在终端有任何的输出，因此应该重定向标准输入/输出/错误输出
- 可以使用**dup2**函数将标准输入/输出/错误输出重定向到**/dev/null**设备（空设备，向其写入数据没有任何输出。

  **#include<unistd.h>**

  **int dup2(int oldfd,int newfd)**

# dup2()函数使用例子

```
int fd;
//将标准输入输出重定向到空设备
fd = open ("/dev/null", O_RDWR, 0);
if (fd != -1)
{
dup2 (fd, STDIN_FILENO);
dup2 (fd, STDOUT_FILENO);
dup2 (fd, STDERR_FILENO);
if (fd > 2)
  close (fd);
}
```

# ⑤ 设置文件创建的预设权限

- 守护进程常要创建一些临时文件
- 这些文件不希望被别的用户查看
- 可以使用**umask**函数修改文件的权限，创建掩码的取值，以满足守护进程的要求

```
#include<sys/types.h>
#include<sys/stat.h>
mode_t umask(mode_t mask);
```

# 权限掩码

- 当你创建一个新的文件时候，它的缺省属性如何确定？如何设置？
- umask：指定在建立文件时预设的权限掩码
  - 语法：umask [-s] [权限掩码]

    权限掩码是由3个八进制的数字所组成,将现有的存取权限减掉掩码后,就是建立文件时预设的权限
  - umask值与权限:umask是从权限中”拿走”相应的权限.
    - 对目录,最大值:7
    - 对普通文件,最大值:6
      - 系统不容许你创立一个文本文件时就赋予它可执行的权限
      - 创建后,用chmod命令增加

# 权限掩码

- umask值与权限的对照表:umask是从权限中”拿走”相应的权限
  - R(4)
  - W(2)
  - X(1)

| umask | 文件 | 目录 |
|---|---|---|
| 0 | 6 | 7 |
| 1 | 6 | 6 |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

# 权限掩码

- **umask值与权限的对照表:umask是从权限中”拿走”相应的权限**
  - R(4)
  - W(2)
  - X(1)

| umask | 文件 | 目录 |
|---|---|---|
| 0 | 6 | 7 |
| 1 | 6 | 6 |
| 2 | 4 | 5 |
| 3 | 4 | 4 |
| 4 | 2 | 3 |
| 5 | 2 | 2 |
| 6 | 0 | 1 |
| 7 | 0 | 0 |

# 权限掩码

- 当你创建一个新的文件时候，它的缺省属性如何确定？如何设置？
- umask：指定在建立文件时预设的权限掩码
  - 语法：umask [-s] [权限掩码]

    权限掩码是由3个八进制的数字所组成,将现有的存取权限减掉掩码后,就是建立文件时预设的权限
  - 设置文件

| csh | sh |
| --- | --- |
| /etc/profile<br>$[Home]/.bash_profile<br>$[Home]/.profile | /etc/bashrc<br>$[Home]/.bashrc |

# What does it mean?

**$vi .bashrc**

```
# .bashrc
#Source global definitions
if [ -f /etc/bashrc ]; then
        ./etc/bashrc
fi
# User specific aliases and functions
JAVA_PATH=/usr/bin/java
export JAVA_PATH
```

每次修改了**.bashrc**等环境变量配置文件,保存退出后都要执行以下命令以使之生效

**$source .bashrc**

# 课堂练习

**$vi /etc/bashrc**

```
# /etc/bashrc
…
if [ $UID –gt 199 ] && [  “`id –gn`”=“`id –un`” ]; then
        umask 002
else
        umask 022
fi
…
```

请问这段SHELL程序分别将目录权限和文件权限设置为多少?

# 一个简单守护进程的实例源代码

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>
```

```
int
daemon (int nochdir, int noclose)
{
  pid_t pid;
  pid = fork ();
  /* 如果创建进程失败 */
  if (pid < 0)
   {
     perror ("fork");
     return -1;
   }
  /* 父进程退出运行 */
  if (pid != 0)
    exit (0);
  /* 成为会话领头进程 */
  pid = setsid();
  if (pid < -1)
   {
     perror ("setsid");
     return -1;
   }
  /* 将工作目录修改成根目录 */
  if (! nochdir)
    chdir ("/");
  /* 将标准输入输出重定向到空设备 */
  if (! noclose)
   {
     int fd;
     fd = open ("/dev/null", O_RDWR, 0);
     if (fd != -1)
   {
     dup2 (fd, STDIN_FILENO);
     dup2 (fd, STDOUT_FILENO);
     dup2 (fd, STDERR_FILENO);
     if (fd > 2)
       close (fd);
   }
   }
  umask (0027);
  return 0;
}
```

```
int main(void)
{
    daemon(0,0);
    sleep(1000);

    return 0;
}
```

运行结果测试如下：

**$./a.out**

**$ps axj | grep a.out**

**1 27109 27109 27109 ? -1 Ss 500 0:00 ./a.out**

**……　　grep a.out**

# 实验过程 & 实验报告 温馨提示 & 提醒

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <time.h>
#include <syslog.h>
```

```
int init_daemon(void)
{
     int pid;
     int i;

     /*忽略终端I/O信号，STOP信号*/
     signal(SIGTTOU,SIG_IGN);
     signal(SIGTTIN,SIG_IGN);
     signal(SIGTSTP,SIG_IGN);
     signal(SIGHUP,SIG_IGN);

     pid = fork();
     if(pid > 0) { exit(0); /*结束父进程，使得子进程成为后台进程*/ }
     else if(pid < 0) {  return -1; }

     setsid(); /*建立一个新的进程组,在这个新的进程组中,子进程成为这个进程组的首进程,以使该进程脱离所有终端*/

     /*再次新建一个子进程，退出父进程，保证该进程不是进程组长，同时让该进程无法再打开一个新的终端*/
     pid=fork();
     if( pid > 0) { exit(0); }
     else if( pid< 0) { return -1; }

     for(i=0;i< NOFILE;close(i++)); /*关闭所有从父进程继承的不再需要的文件描述符*/

     chdir("/"); /*改变工作目录，使得进程不与任何文件系统联系*/

     umask(0); /*将文件当时创建屏蔽字设置为0*/

     signal(SIGCHLD,SIG_IGN); /*忽略SIGCHLD信号*/

     return 0;
}
```

```
int main()
{
        time_t now;
        init_daemon();
        syslog(LOG_USER|LOG_INFO,"测试守护进程! \n");
        while(1) {
                sleep(8);
                time(&now);
                syslog(LOG_USER|LOG_INFO,"系统时间: \t%s\t\t\n",ctime(&now));
        }
}
```

编译运行此程序。然后用 ps –ef 查看进程状态，该进程的状态如下：

```
UID        PID    PPID  C  STIME   TTY          TIME    CMD
zzy      30074       1  0  20:37   ?        00:00:00   ./daemon
```

从结果可以看出该进程具备守护进程的所有特征。

查看系统日志的结果片段如下：

```
Jan 28 20:38:43 zzy daemonproc: 系统时间:      Sun Jan 28 20:38:43 2007
Jan 28 20:38:51 zzy daemonproc: 系统时间:      Sun Jan 28 20:38:51 2007
Jan 28 20:38:59 zzy daemonproc: 系统时间:      Sun Jan 28 20:38:59 2007
```

注意：使用 syslog 函数前，首先需要配置/etc/syslog.conf 文件，在该文件的最后加入一行 user.* /var/log/test.log。然后重新启动 syslog 服务，命令分别为：/etc/init.d/syslog stop 和/etc/init.d/syslog start。这样 syslog 的信息就会写入 test.log 文件中。

〈〈进程控制〉〉 -- 〈〈Linux C编程实战〉〉第7章

**自己百度下Linux syslog/rsyslog 日志服务器架设攻略**

# 其他可能影响日志记录的原因

1. SELinux 缺省会通过 Linux 审计系统（auditd）将日志写在 /var/log/audit/audit.log 内，而这项务服缺省为启用的。
2. 假若 auditd 并未运行，信息将会被写进 /var/log/messages
3. 简单的修正:

#vi /etc/selinux/config

SELINUX=“”

#shutdown –r now

或者: SELINUX=disabled

或者: # setenforce 0

# 问题描述

- 实验第四题我想把有关守护进程的所有信息都写入到/home/padric/log/daemon里面，我在rsyslog.conf配置文件里面加入了代码，我以为他会自己创建不存在的log目录和daemon文件，但是发现系统没有创建，又自己创建了该文件，重启了一次系统，可是该日志文件里面依旧什么内容都没有，是因为没有启动写入服务吗？在哪里开启？
- /etc/rsyslog.conf加入的代码

```
#### RULES ####

#Log Daemon Information in /home/log/daemonlog
#Insert at 2012.3.25 15:30pm
#daemon.*                                                /home/padric/log/daemon
daemon.*                                                /home/padric/log/daemon
```

# 解决方法

**1)看看你的syslogd/rsyslogd守护进程有没有启动**

```
[szu@taishan02-vm-10 ~]$ ps aux | grep "rsyslog"
root        1277  0.0  0.9 676288 69440 ?        Ssl  Apr18  11:58 /usr/sbin/rsyslogd -n -iNONE
szu      1114339  0.0  0.0 214016  1536 pts/0    S+   20:49   0:00 grep --color=auto rsyslog
```

**2)对应的头文件定义:**

```
[szu@taishan02-vm-10 include]$ pwd
/usr/include
[szu@taishan02-vm-10 include]$ find ./ -name "*" -exec grep "LOG_INFO" {} \; -print
grep: ./: Is a directory
grep: ./asm-generic: Is a directory
grep: ./gnu: Is a directory
grep: ./scsi: Is a directory
grep: ./scsi/fc: Is a directory
grep: ./gdb: Is a directory
grep: ./sys: Is a directory
#define LOG_INFO        6       /* informational */
    { "info", LOG_INFO },
./sys/syslog.h
grep: ./sound: Is a directory
```

# 解决方法

1)看看你的syslogd/rsyslogd守护进程有没有启动
2)对应的头文件定义:
3)修改配置文件/etc/rsyslog.conf

```
#rsyslog v3 config file

# if you experience problems, check
# http://www.rsyslog.com/troubleshoot for assistance

#### MODULES ####

$ModLoad imuxsock.so    # provides support for local system logging (e.g. via logger command)
$ModLoad imklog.so      # provides kernel logging support (previously done by rklogd)
#$ModLoad immark.so     # provides --MARK-- message capability

# Provides UDP syslog reception
#$ModLoad imudp.so
#$UDPServerRun 514

# Provides TCP syslog reception
#$ModLoad imtcp.so
#$InputTCPServerRun 514


#### GLOBAL DIRECTIVES ####

# Use default timestamp format
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat

# File syncing capability is disabled by default. This feature is usually not required,
# not useful and an extreme performance hit
#$ActionFileEnableSync on


#### RULES ####

# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none                /var/log/messages
*.info;mail.none;authpriv.none;cron.none                /home/yuhong/log/messages

# The authpriv file has restricted access.
authpriv.*                                              /var/log/secure

# Log all the mail messages in one place.
"/etc/rsyslog.conf" 79L, 2823C
```

# 解决方法

1)看看你的syslogd/rsyslogd守护进程有没有启动

2)对应的头文件定义:

3)修改配置文件/etc/rsyslog.conf

4)重新启动rsyslogd

```
#service rsyslogd restart
```

5)运行你的守护进程

如果你的守护进程已经启动,停止并重新启动

6)检查结果:

```
#tail -10 /var/log/messages
#tail -10 /home/yuhongf/log/messages
```

```
Oct 17 19:24:02 taishan02-vm-2021-1 sixth_one[8916]: 测试守护进程!
Oct 17 19:24:02 taishan02-vm-2021-1 [/bin/bash]: [./sixth_one] return code=[0], execute success
 by [caoyixuan2019282129(uid=1001)] from [pts/0 (192.168.122.1)]
Oct 17 19:24:10 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:24:10 2021
Oct 17 19:24:18 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:24:18 2021
Oct 17 19:24:26 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:24:26 2021
Oct 17 19:24:34 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:24:34 2021
Oct 17 19:24:42 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:24:42 2021
Oct 17 19:24:50 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:24:50 2021
Oct 17 19:24:58 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:24:58 2021
Oct 17 19:25:06 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:25:06 2021
Oct 17 19:25:14 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:25:14 2021
Oct 17 19:25:22 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:25:22 2021
Oct 17 19:25:30 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:25:30 2021
Oct 17 19:25:38 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:25:38 2021
Oct 17 19:25:46 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:25:46 2021
Oct 17 19:25:54 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:25:54 2021
Oct 17 19:26:02 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:26:02 2021
Oct 17 19:26:10 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:26:10 2021
Oct 17 19:26:18 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:26:18 2021
Oct 17 19:26:26 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:26:26 2021
Oct 17 19:26:34 taishan02-vm-2021-1 sixth_one[8916]: 系统时间: #011Sun Oct 17 19:26:34 2021
```

2. 根据课件描述的守护进程创建步骤分别编程创建守护进程a.out/thread4/b
运行及测试方法如下：$./a.out；　　$ps -axj | grep a.out

```
[yuansheng@yuansheng ~]$ ps -axj | grep thread4
Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ
 2338  2576  2576  2338 pts/0     2576 S+     500   0:00 vim thread4.c
 2622  2644  2643  2622 pts/2     2643 S+     500   0:00 grep thread4
[yuansheng@yuansheng ~]$ a
```

测试一

```
[zlb@zhenglinbo ~]$ ps axj | grep b
    0     1     1     1 ?           -1 Ss       0   0:03 /sbin/init
    2    13     0     0 ?           -1 S        0   0:00 [bdi-default]
    2    15     0     0 ?           -1 S        0   0:00 [kblockd/0]
    2    21     0     0 ?           -1 S        0   0:00 [ksuspend_usbd]
    2    22     0     0 ?           -1 S        0   0:00 [khubd]
    2    42     0     0 ?           -1 S        0   0:00 [usbhid_resumer]
    2   387     0     0 ?           -1 S        0   0:01 [jbd2/dm-0-8]
    1   473   473   473 ?           -1 S<s      0   0:00 /sbin/udevd -d
    2   836     0     0 ?           -1 S        0   0:00 [jbd2/sda1-8]
    1   893   893   893 ?           -1 Ss       0   0:00 /usr/bin/system-setup-keyb
  991   993   993   993 ?           -1 S<sl     0   0:00 /sbin/audispd
  993  1007   993   993 ?           -1 S<       0   0:00 /usr/sbin/sedispatch
    1  1018  1015   892 ?           -1 Sl       0   0:00 /sbin/rsyslogd -c 4
```

测试二

```
[root@localhost LL]# gcc test6.c
[root@localhost LL]# ./a.out
[root@localhost LL]# ps -axj|grep a.out
Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ
    1  2656  2656  2656 ?           -1 Ss       0   0:00 ./a.out
 2656  2657  2656  2656 ?           -1 S        0   0:00 ./a.out
    1  2679  2679  2679 ?           -1 Ss       0   0:00 ./a.out
 2679  2680  2679  2679 ?           -1 S        0   0:00 ./a.out
    1  2734  2734  2734 ?           -1 Ss       0   0:00 ./a.out
 2734  2735  2734  2734 ?           -1 S        0   0:00 ./a.out
 2573  2802  2801  2547 pts/1     2801 S+       0   0:00 grep a.out
```

测试三

# 写日志的重要性

第一个重要收获是进一步了解了代码编程中日志记录的重要性。我在平时学习中编写的很多学习型项目中几乎从未考虑过加入日志模块来记录程序运行状况，很多时候只是简单的通过终端输出来检测程序运行是否正常。但在实际项目中往往需要对日志进行详细的记录，比如前端与后端接口的通信情况、后端数据库数据事务的执行情况等等都需要详细准确记录诸如执行时间、执行结果、错误类型等信息。同时在代码中可能出现的异常都要进行捕获并详细记录信息。通过对日志进行统计、分析、综合，能有效地掌握网站运行状况，发现和排除错误原因，更好的加强系统的维护和管理。↵

2019级腾班霍晓雨的专业实习报告