**云南大学软件学院**

**Linux应用开发实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **何志颖** | **年级** | **2015级** | **班级** | **软件工程** |
| **学号** | **20151120237** | **专业** | **软件工程** | **序号** | **31** |
| **实验名称** | **实验五 进程控制与通信实验** | | | **成绩** |  |

指导教师 (签名):

**实验五 进程控制与通信实验**

1. **实验目的**

1、掌握进程的常用终端命令；

2、掌握用system、exec函数簇、fork函数创建进程；

3、掌握wait、waitpid函数的应用；

4、掌握守护进程在监控中的应用并编写守护进程；

5、掌握常用中断的方法；

6、掌握signal函数实现信号处理程序；

7、掌握应用管道实现信号处理的方法。

1. **实验任务与要求**

1、用exec函数创建进程；

2、应用fork( )函数创建子进程，在父子进程中执行不同的任务；

3、守护进程的编写与应用；

4、管道读写程序的编写与应用；

5、应用命名管道实现简单的聊天程序设计；

6、应用消息队列实现简单的聊天程序；

7、共享内存通信机制程序设计。

1. **实验内容**

**任务一：程序设计**

用exec函数簇中的6个函数替换修改课件6-3.c源程序并调试结果。（源程序给出替换部分的代码即可）

**任务二：程序设计**

（1）在父子进程中分别编写循环程序，应用函数sleep的不同参数等，体现程序中父子进程的并发运行。

（2）在父子进程中分别执行不同的任务，例如在子进程中执行文件编辑任务，在父进程中执行网络连通情况的测试任务，子进程退出后父进程才退出。

**任务三：程序设计**

waitpid函数的应用，要求子进程用sleep等待10秒，父进程用waitpid函数等待子进程正常结束，父进程在等待的时候不阻塞，每1秒在屏幕上输出一行文字，若发现子进程退出，打印出等待进程的进程号(PID)和退出状态。请编写一程序进行调试。

提示：要父进程周期行的检查某个特定的子进程是否终止，在waitpid（pid, &stauts,option）函数的option要如何设置？

**任务四：程序调试**

设计三个并发的守护进程在后台运行，其中第一子进程写守护进程的运行日志记录，第二子进程child2则监控进程中是否有gedit工具调用，第二子进程child3则检查自己是否有新邮件到达，若有则将邮件内容输出到一个主目录下文件。

**任务五：学习巩固**

（1）使用命令kill，显示linux环境下的信号。分析这些信号的特点。

（2）下列是进程调度中常用的函数，通过网络资源查找这些函数的功能与用法。

|  |  |
| --- | --- |
| 函 数 | 功能与用法 |
| Kill |  |
| Raise |  |
| alarm |  |
| pause |  |
| signal |  |
| sigemptyset |  |
| sigfillset |  |
| sigaddset |  |
| sigdelset |  |
| sigismember |  |
| sigprocmask |  |

**任务六：程序设计**

编写一个C程序，完成以下功能。

（1）父进程创建2个子进程P1、P2；

（2）父进程捕捉从键盘上通过CTRL+C键发来的中断信号；

（3）父进程获得中断信号后使用系统调用kill()向两个子进程分别发终止执行信号SIGUSR1和SIGUSR2；

（4）子进程捕捉到各自的信号后分别输出：

child1 is killed by parent!和child2 is killed by parent!

然后终止执行（无先后次序的要求）；

（5）父进程等待两个子进程终止后输出以下信息，然后终止执行：

parent process is killed!

**四、实验算法及程序**（涉及编程设计的时候需要提供源程序，并有注释）

任务一：

（1）

（2）



（3）

（4）

（5）



（6）



任务二：

(1)#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main()

{

pid\_t result;

result = fork();

if(result < 0)

{

perror("创建子进程失败了！");

}

else if(result == 0)

{

printf("子进程创建成功！\n");

for(int i =0; i < 10; i++)

{

printf("子进程在运行...\n");

sleep(3);

}

}

else

{

for(int i = 0; i < 5; i++)

{

printf("父进程在运行...\n");

sleep(5);

}

}

}

(2)#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/wait.h>

int main()

{

pid\_t pid,wpid;

pid = fork();

int status,i;

if(pid < 0)

{

perror("创建子进程失败！");

}

else if(pid == 0)

{

printf("进程号为%d子进程中执行文本编辑任务...\n",getpid());

execlp("/bin/vi","vi","test.txt",NULL);

sleep(5);

exit(5);

}

else

{

printf("父进程中执行网络连通情况的测试任务...\n");

execlp("/bin/ping","ping","113.55.4.24","-c 5",NULL);

wpid = wait(&status);

i = WEXITSTATUS(status);

// printf("等待的进程号为%d的子进程退出了，退出状态为%d。\n父进程

这时才可以退出！\n",wpid,i);

}

}

任务三：

#include<stdlib.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

int main(void)

{

pid\_t pid,wpid;

pid = fork();

int status;

if(pid < 0)

{

perror("创建子进程失败了！\n");

exit(1);

}

else if(pid == 0)

{

printf("这是子进程在运行,休眠10秒...\n");

sleep(10);

exit(5);

}

else

{

printf("这是父进程在运行...\n");

while(1)

{

waitpid(pid,&status,WNOHANG);

printf("等待子进程退出...\n");

if(WIFEXITED(status) == 0)

{

sleep(1);

printf("父进程：hello world...");

}

else

{

printf("子进程退出了！\n");

printf("等待的进程的进程号是:%d，退出状态为:%d\n",pid,WEXITSTATUS(status));

break;

}

}

}

}

任务四：

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

int main()  
{  
pid\_t child1,child2,child3;  
struct stat buf;  
int i,check=0,j=0;  
time\_t t;  
DIR \* dir;  
struct dirent \* ptr;

child1=fork();  
if (child1>0)  
exit(0); /\*父进程退出\*/  
else if(child1<0)  
{  
perror("创建子进程失败");  
exit(1);  
}

/\*第一子进程\*/  
setsid();  
chdir("/");  
umask(0);  
for(i=0;i<NOFILE;++i)  
close(i);  
openlog("守护进程程序信息",LOG\_PID,LOG\_DAEMON);

child2=fork();  
if (child2==-1)  
{  
perror("创建子进程失败");  
exit(2);  
}  
else if (child2==0)/\*第二子进程中的child2\*/  
{   
i=0;  
while(i++<100){  
system("ps -ef|grep gedit> /home/king/gedit.log");  
stat("home/king/gedit.log",&buf);

if (buf.st\_size>180 && check==0)

{  
t=time(0);   
syslog(LOG\_INFO,"gedit开始时间为： %s\n",asctime(localtime(&t)));  
check=1;   
}   
if (buf.st\_size<180 && buf.st\_size>0 && check==1)  
{   
t=time(0);  
syslog(LOG\_INFO,"gedit结束时间为： %s\n",asctime(localtime(&t)));  
check=0;  
}  
sleep(1);  
}   
}  
else  
{ /\*在第一子进程下继续创建进程\*/  
child3=fork();  
if (child3<0){  
perror("创建子程序失败");  
exit(3);  
}  
else if (child3==0){ /\*第二子进程child3用来查看邮件\*/  
j=0;  
dir=opendir("/var/spool/mail/king");  
while (j<6){  
j++;  
sleep(10);  
if ((ptr=readdir(dir))!=NULL){   
system("cat /var/spool/mail/king/\* > /home/king/mail.log");  
}   
}  
closedir(dir);  
}  
else   
{ /\*第一子进程写日志来记录守护进程的运行\*/  
t=time(0);  
syslog(LOG\_INFO,"守护进程开始时间为： %s\n",asctime(localtime(&t)));  
waitpid(child2,NULL,0);  
waitpid(child3,NULL,0);  
t=time(0);  
syslog(LOG\_INFO,"守护进程结束时间为： %s\n",asctime(localtime(&t)));  
closelog();  
while (1)  
sleep(10);  
}   
}  
}

任务六：

#include<stdio.h>

#include<signal.h>

#include<unistd.h>

#include<stdlib.h>

void waiting(),stop(),alarming();

int wait\_mark;

int main(void)

{

int p1,p2;

if(p1 = fork())

{

if(p2 = fork())

{

wait\_mark = 1;

signal(SIGINT,stop);

signal(SIGALRM,alarming);

waiting();

kill(p1,16);

kill(p2,17);

printf("parent process is killed!\n");

exit(0);

}

else

{

wait\_mark = 1;

signal(17,stop);

signal(SIGINT,SIG\_IGN);

while(wait\_mark != 0)

{

lockf(1,1,0);

}

printf("child2 is killed by parent!\n");

lockf(1,0,0);

exit(0);

}

}

else

{

wait\_mark = 1;

signal(16,stop);

signal(SIGINT,SIG\_IGN);

while(wait\_mark != 0)

{

lockf(1,1,0);

}

printf("child1 is killed by parent!\n");

lockf(1,0,0);

exit(0);

}

}

void waiting()

{

sleep(5);

if(wait\_mark != 0)

{

kill(getpid(),SIGALRM);

}

}

void alarming()

{

wait\_mark = 0;

}

void stop()

{

wait\_mark = 0;

}

**五、实验结果记录及分析**

任务一：

1. execl():





1. execv():





1. execlp():





1. execle():





1. execve():





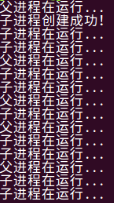
1. execvp():



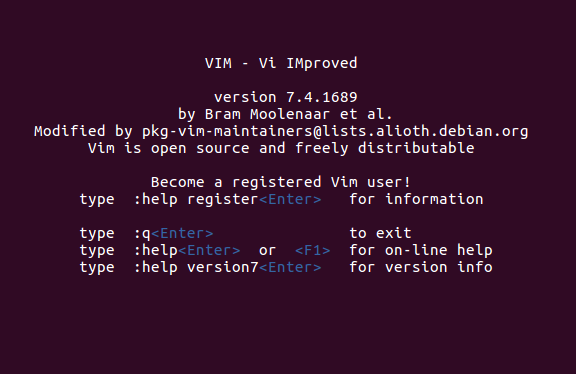


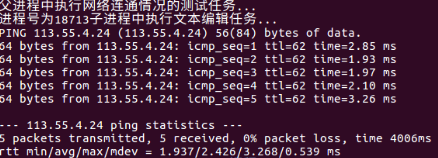
任务二：

1. 在父子进程中分别编写循环程序，应用函数sleep的不同参数等，体现程序中父子进程的并发运行。

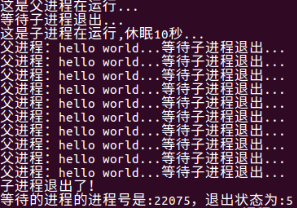


(2)在父子进程中分别执行不同的任务，例如在子进程中执行文件编辑任务，在父进程中执行网络连通情况的测试任务，子进程退出后父进程才退出。

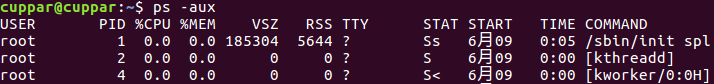




任务三：

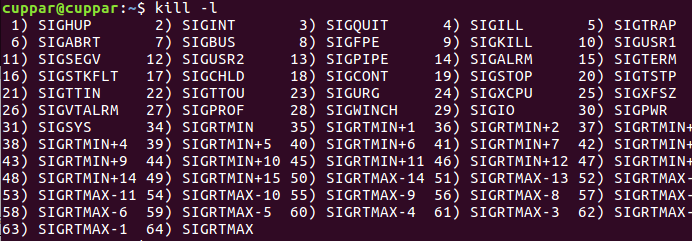


任务四：



任务五：

(1)使用命令kill，显示linux环境下的信号。分析这些信号的特点。



分析特点：

这些信号都是由“SIG”开头的；很多信号都是与机器的体系结构相关的信号值，能处理一些默认动作。

(2)下列是进程调度中常用的函数，通过网络资源查找这些函数的功能与用法。

|  |  |
| --- | --- |
| 函 数 | 功能与用法 |
| Kill | **功能描述：用于向任何进程组或进程发送信号。**  用法：**#include <sys/types.h>  #include <signal.h>**  **int kill(pid\_t pid, int sig);** |
| Raise | 功能描述：函数对程序发送指定的信号*signal*.  用法：**#include <signal.h>**  **int raise( int signal );** |
| alarm | 功能描述：设置信号传送闹钟，即用来设置信号SIGALRM在经过参数seconds秒数后发送给目前的进程。如果未设置信号SIGALARM的处理函数，那么alarm()默认处理终止进程。  用法：int alarm(unsigned int seconds); |
| pause | 功能描述：令目前的进程暂停(进入睡眠状态), 直到被信号(signal)所中断。  用法：#include <unistd.h>  int pause(void); |
| signal | 功能描述：为指定的信号安装一个新的信号处理函数。  用法：#include <signal.h>  void (\*signal(int signo, void (\*func)(int)))(int); |
| sigemptyset | 功能描述：将参数set 信号集初始化并清空  用法：#include <signal.h>  int sigemptyset(sigset\_t \*set); |
| sigfillset | 功能描述：初始化信号集合,只是将信号集合设置为所有信号的集合  用法：#include<signal.h>  int sigfillset(sigset\_t \* set); |
| sigaddset | 功能描述：将信号signo 加入到信号集合之中,sigdelset 将信号从信号集合中删除.  用法：#include<signal.h>  int sigaddset(sigset\_t \*set,int signum); |
| sigdelset | 功能描述：  用法：#include<signal.h>  nt sigdelset(sigset\_t \* set,int signum); |
| sigismember | 功能描述：查询信号是否在信号集合之中.  用法：#include<signal.h>  int sigismember(const sigset\_t \*set,int signum); |
| sigprocmask | 功能描述：将指定的信号集合set 加入到进程的信号阻塞集合之中去,如果提供了oset 那么当前的进程信号阻塞集合将会保存在oset 里面  用法：#include<signal.h>  int sigprocmask(int how, const [sigset\_t](https://baike.baidu.com/item/sigset_t/4481187" \t "/home/jasmine/Documents\\x/_blank) \*restrict set, sigset\_t \*restrict oldset); |

任务六：







**六、实验心得**

通过本次试验，使我掌握了如何使用system、exec函数簇、fork函数创建进程，还有wait、waitpid函数的应用，为以后的linux学习打下了基础。