

# Linux进程管理实验

---

## 实验报告

---

### 一. 实验目的

熟练掌握Linux操作系统的使用，掌握Linux的系统的进程管理相关内容，掌握进程之间的通信方式。

进程是操作系统中最重要的概念，贯穿始终，也是学习现代操作系统的关键。通过本次实验，要求理解进程的实质和进程管理的机制。在Linux系统下实现进程从创建到终止的全过程，从中体会进程的创建过程、父进程和子进程的关系、进程状态的变化、进程之间的同步机制、进程调度的原理和以信号和管道为代表的进程间通信方式的实现。

### 二. 实验内容

1.在命令行新建多个普通用户，如tux, bob, Alice, lily等，给每个用户创建密码，并将这几个用户分到同一个组xjtuse中。再新建两个组coding和testing，使得某些用户也分别为其组用户。在root用户和新建用户之间切换，验证用户创建成功与否。（给出相关命令运行结果）

2.实现**sudo** 委托管理任务，给上述某一指定的普通用户赋予创建用户的权限。（给出相关配置文件和命令运行结果）

3.备份数据是系统应该定期执行的任务，请利用cron计划作业在每周五下午6:10对某用户（如tux）主目录下的文件进行备份（可使用tar 命令）。给出相关运行结果和邮件记录。

4.编制实现软中断通信的程序

使用系统调用fork()创建两个子进程，再用系统调用signal()让父进程捕捉键盘上发出的中断信号（即按delete键），当父进程接收到这两个软中断的某一个后，父进程用系统调用kill()向两个子进程分别发出整数值为16和17软中断信号，子进程获得对应软中断信号，然后分别输出下列信息后终止：

**Child process 1 is killed by parent !!**

**Child process 2 is killed by parent !!**

父进程调用wait()函数等待两个子进程终止后，输入以下信息，结束进程执行：

**Parent process is killed!!**

多运行几次编写的程序，简略分析出现不同结果的原因。

5.编制实现进程的管道通信的程序

使用系统调用pipe()建立一条管道线，两个子进程分别向管道写一句话：

**Child process 1 is sending a message!**

**Child process 2 is sending a message!**

而父进程则从管道中读出来自于两个子进程的信息，显示在屏幕上。

要求：父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。

### 三.题目分析及基本设计过程分析

1. 以下是上述内容的另一种表述方式:

#### 1. 用户和组管理

可以通过 `useradd` 命令创建新用户, 使用 `groupadd` 命令创建用户组。利用 `usermod` 命令将用户加入到指定的用户组中, 并通过 `passwd` 命令修改用户的密码。此外, 使用 `su` 命令可以切换到其他用户的身份, 而 `id` 命令则用于查看当前用户所属的组信息。

#### 2. sudo权限配置

通过编辑 `/etc/sudoers` 文件, 可以为用户 `lily` 分配特定命令的执行权限。例如, 添加 "`lily ALL=/usr/sbin/useradd`", 这样 `lily` 用户就可以执行 `useradd` 命令。

#### 3. 定时任务配置

利用 `/etc/crontab` 文件可以设置系统的定时任务。例如, 设置每周五18:10备份用户 `tux` 的数据, 任务为 `10 18 * * 5 root tar tux`。为了便于测试, 可以将任务改为每分钟执行一次, 即 `*/1 * * * * root tar tux`。

#### 4. 进程控制实现

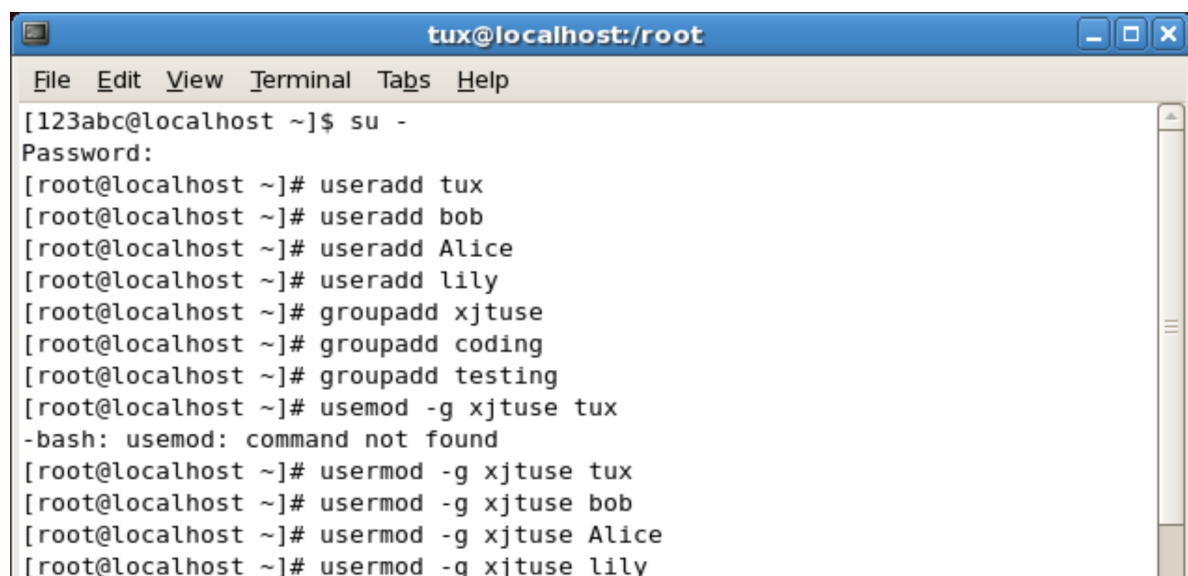
在Linux系统中, 可以使用 `fork`、`wait`、`exit`、`getpid`、`kill` 和 `signal` 等系统调用来实现进程控制。通过 `fork` 创建子进程, 并根据其返回值区分父子进程。父进程通过 `signal` 接收软中断信号, 如果没有收到信号, 则在 `while` 循环中等待。子进程也通过 `signal` 接收父进程的信号, 若未收到信号则在循环中等待。当父进程收到软中断信号后, 会向子进程发送终止信号, 并通过 `wait` 等待子进程结束, 然后自身终止。

#### 5. 管道通信实现

使用 `pipe(int pipeid[2])` 函数创建管道, 并通过 `fork` 创建子进程。子进程通过 `write(pipeid[1], buf, size)` 向管道写入数据, 并使用 `lockf` 对管道的写端进行加锁保护, 写入完成后释放锁。父进程在子进程完成写入后, 通过 `read(pipeid[0], buf, size)` 从管道读取数据。

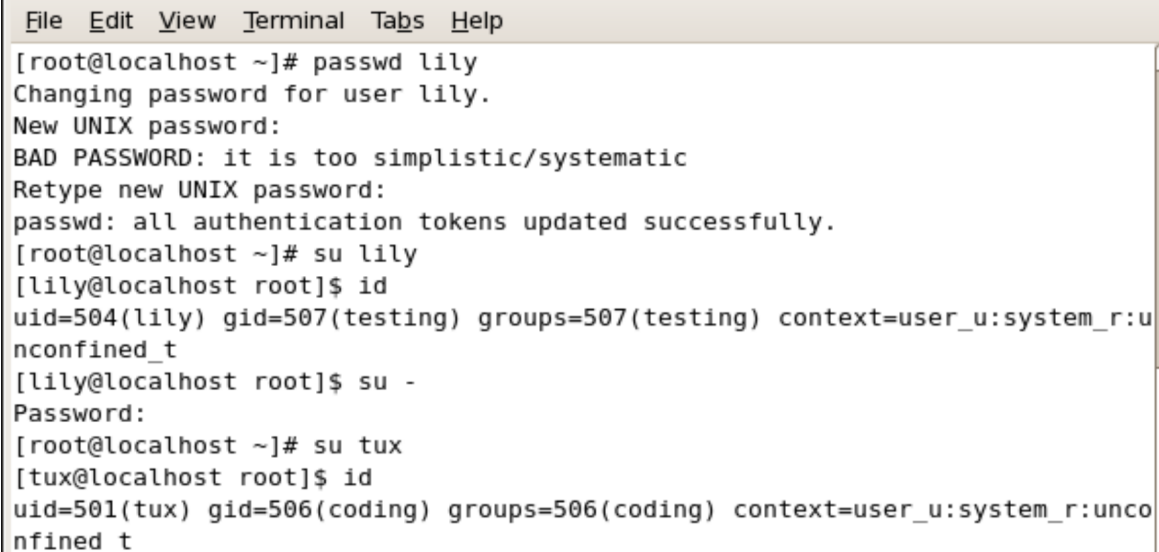
### 四. 运行截图和相关说明

1. 新建用户 `tux`、`bob`、`Alice`、`lily`, 组 `xjtuse`、`coding`、`testing`。设置用户密码并将其加入组。利用 `usermod -g` 设置主组, `-G` 设置附加组。



```
tux@localhost:/root
File Edit View Terminal Tabs Help
[123abc@localhost ~]$ su -
Password:
[root@localhost ~]# useradd tux
[root@localhost ~]# useradd bob
[root@localhost ~]# useradd Alice
[root@localhost ~]# useradd lily
[root@localhost ~]# groupadd xjtuse
[root@localhost ~]# groupadd coding
[root@localhost ~]# groupadd testing
[root@localhost ~]# usemod -g xjtuse tux
-bash: usemod: command not found
[root@localhost ~]# usermod -g xjtuse tux
[root@localhost ~]# usermod -g xjtuse bob
[root@localhost ~]# usermod -g xjtuse Alice
[root@localhost ~]# usermod -g xjtuse lily
```

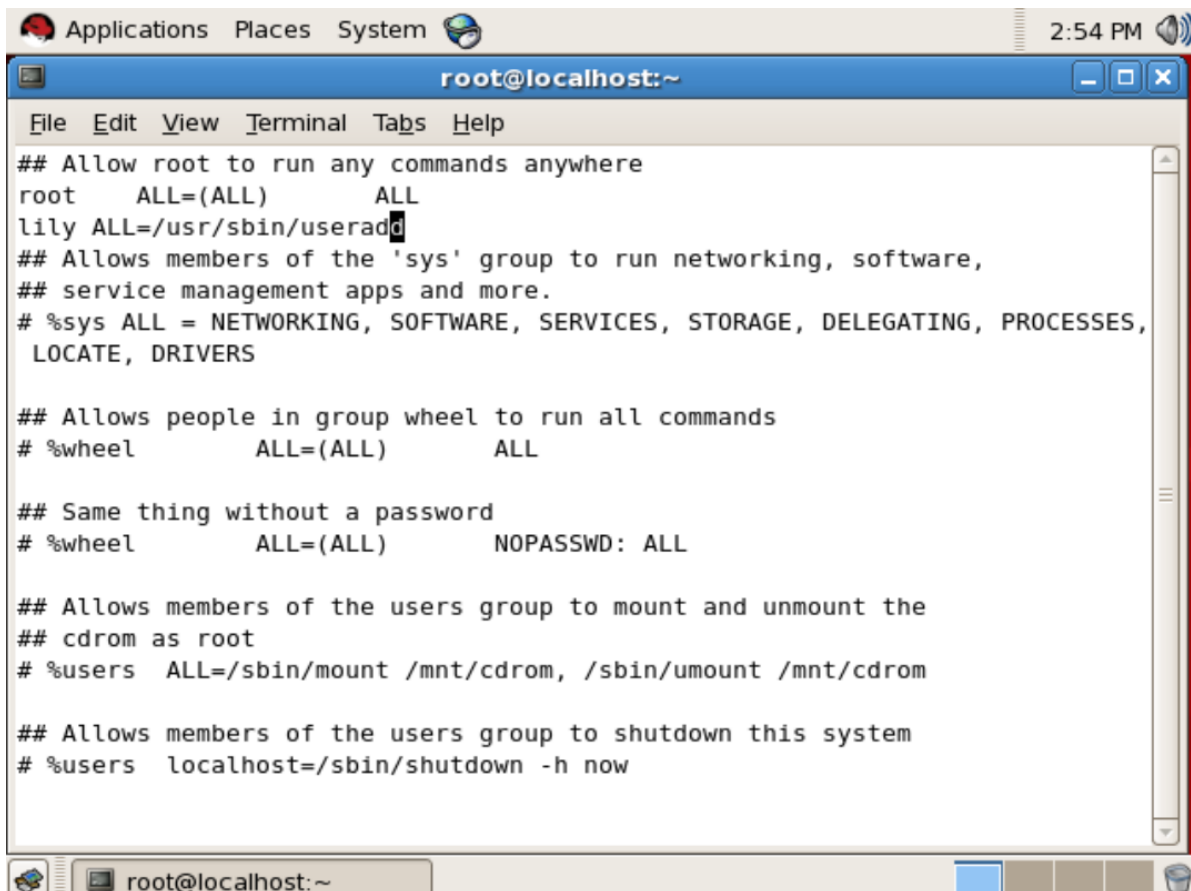
```
[root@localhost ~]# usermod -G coding tux
[root@localhost ~]# usermod -G coding bob
[root@localhost ~]# usermod -G testing Alice
[root@localhost ~]# usermod -G testing lily
```



A terminal window with a menu bar (File, Edit, View, Terminal, Tabs, Help). The output shows the process of changing the password for user 'lily' and then switching to that user. It also shows switching to user 'tux' and checking the effective user and group IDs.

```
File Edit View Terminal Tabs Help
[root@localhost ~]# passwd lily
Changing password for user lily.
New UNIX password:
BAD PASSWORD: it is too simplistic/systematic
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@localhost ~]# su lily
[lily@localhost root]$ id
uid=504(lily) gid=507(testing) groups=507(testing) context=user_u:system_r:unconfined_t
[lily@localhost root]$ su -
Password:
[root@localhost ~]# su tux
[tux@localhost root]$ id
uid=501(tux) gid=506(coding) groups=506(coding) context=user_u:system_r:unconfined_t
```

2. 修改/etc/sudoers, 赋予lily添加用户的权限。



A terminal window titled 'root@localhost:~' showing the content of the /etc/sudoers file. The file contains various privilege rules for different users and groups, including a rule for user 'lily' to run 'useradd'.

```
Applications Places System 2:54 PM
root@localhost:~
File Edit View Terminal Tabs Help
## Allow root to run any commands anywhere
root    ALL=(ALL)        ALL
lily ALL=/usr/sbin/useradd
## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES,
LOCATE, DRIVERS

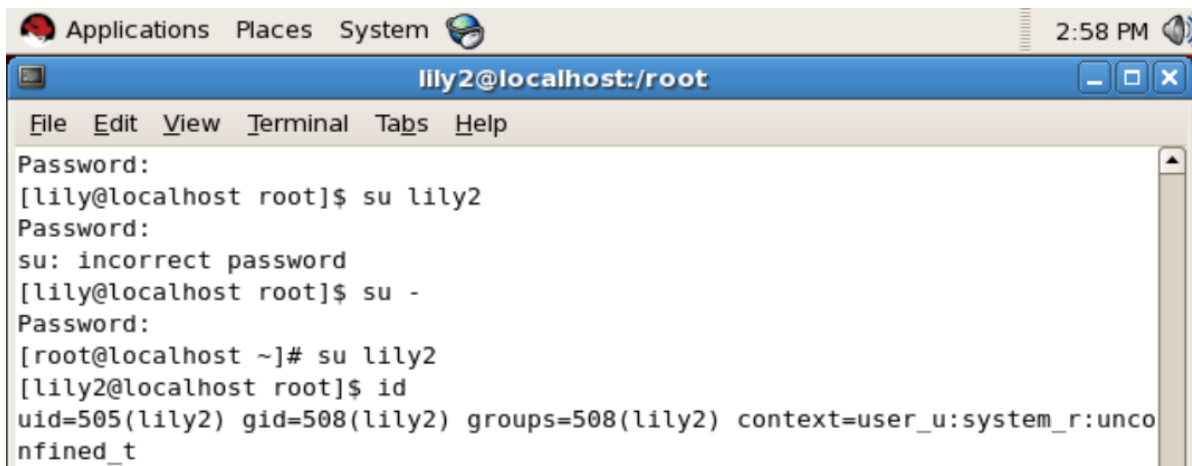
## Allows people in group wheel to run all commands
# %wheel    ALL=(ALL)        ALL

## Same thing without a password
# %wheel    ALL=(ALL)        NOPASSWD: ALL

## Allows members of the users group to mount and unmount the
## cdrom as root
# %users    ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom

## Allows members of the users group to shutdown this system
# %users    localhost=/sbin/shutdown -h now
```

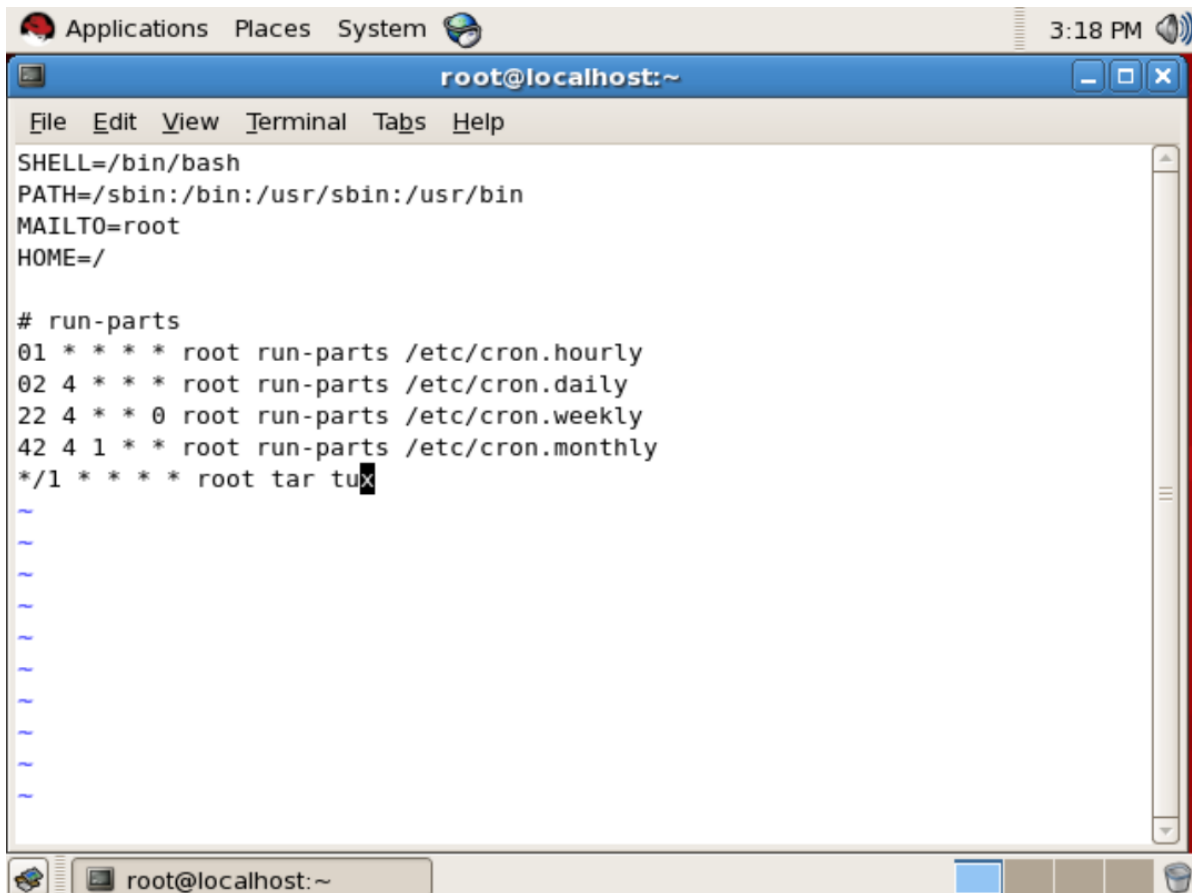
切换到用户lily, 可以添加用户abc。



A terminal window titled 'lily2@localhost:/root' with a menu bar (File, Edit, View, Terminal, Tabs, Help) and a status bar (Applications, Places, System, 2:58 PM). The terminal shows the following commands and output:

```
Password:
[lily2@localhost root]$ su lily2
Password:
su: incorrect password
[lily2@localhost root]$ su -
Password:
[root@localhost ~]# su lily2
[lily2@localhost root]$ id
uid=505(lily2) gid=508(lily2) groups=508(lily2) context=user_u:system_r:unconfined_t
```

3. 修改/etc/crontab



A terminal window titled 'root@localhost:~' with a menu bar (File, Edit, View, Terminal, Tabs, Help) and a status bar (Applications, Places, System, 3:18 PM). The terminal displays the contents of the /etc/crontab file:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
*/1 * * * * root tar tu
```

cron执行定时任务

```
Applications Places System 3:27 PM
root@localhost:~
File Edit View Terminal Tabs Help
[root@localhost ~]# mail
Mail version 8.1 6/6/93. Type ? for help.
"/var/spool/mail/root": 19 messages 19 new
>N 1 logwatch@localhost.l Wed Mar 12 04:02 42/1578 "Logwatch for loca"
N 2 logwatch@localhost.l Wed Mar 12 04:15 42/1578 "Logwatch for loca"
N 3 root@localhost.local Wed Mar 12 15:17 24/954 "Cron <root@localh"
N 4 root@localhost.local Wed Mar 12 15:18 24/954 "Cron <root@localh"
N 5 root@localhost.local Wed Mar 12 15:19 24/954 "Cron <root@localh"
N 6 root@localhost.local Wed Mar 12 15:20 24/954 "Cron <root@localh"
N 7 root@localhost.local Wed Mar 12 15:20 25/1026 "Cron <root@localh"
N 8 root@localhost.local Wed Mar 12 15:21 24/954 "Cron <root@localh"
N 9 root@localhost.local Wed Mar 12 15:21 25/1026 "Cron <root@localh"
N 10 root@localhost.local Wed Mar 12 15:22 24/954 "Cron <root@localh"
N 11 root@localhost.local Wed Mar 12 15:22 25/1026 "Cron <root@localh"
N 12 root@localhost.local Wed Mar 12 15:23 24/954 "Cron <root@localh"
N 13 root@localhost.local Wed Mar 12 15:23 25/1026 "Cron <root@localh"
N 14 root@localhost.local Wed Mar 12 15:24 24/954 "Cron <root@localh"
N 15 root@localhost.local Wed Mar 12 15:24 25/1026 "Cron <root@localh"
N 16 root@localhost.local Wed Mar 12 15:25 24/954 "Cron <root@localh"
N 17 root@localhost.local Wed Mar 12 15:25 25/1026 "Cron <root@localh"
N 18 root@localhost.local Wed Mar 12 15:26 25/1026 "Cron <root@localh"
&
```

```
Applications Places System 3:27 PM
root@localhost:~
File Edit View Terminal Tabs Help
N 17 root@localhost.local Wed Mar 12 15:25 25/1026 "Cron <root@localh"
N 18 root@localhost.local Wed Mar 12 15:26 25/1026 "Cron <root@localh"
& 18
Message 18:
From root@localhost.localdomain Wed Mar 12 15:26:05 2025
Date: Wed, 12 Mar 2025 15:26:02 -0700
From: root@localhost.localdomain (Cron Daemon)
To: root@localhost.localdomain
Subject: Cron <root@localhost> tar tux
Content-Type: text/plain; charset=UTF-8
Auto-Submitted: auto-generated
X-Cron-Env: <SHELL=/bin/bash>
X-Cron-Env: <PATH=/sbin:/bin:/usr/sbin:/usr/bin>
X-Cron-Env: <MAILTO=root>
X-Cron-Env: <HOME=/>
X-Cron-Env: <LOGNAME=root>
X-Cron-Env: <USER=root>

tar: You may not specify more than one '-Acdrux' option
Try `tar --help' or `tar --usage' for more information.
&
```

4. 代码如下:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
```

```

int wait_mark;

void waiting()
{
    while (wait_mark != 0);
}

void stopwaiting(int sig)
{
    wait_mark = 0;
}

int main()
{
    struct sigaction sa;
    int pid1, pid2;

    sa.sa_handler = stopwaiting;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;

    while ((pid1 = fork()) == -1);

    if (pid1 > 0)
    {
        while ((pid2 = fork()) == -1);

        if (pid2 > 0)
        {
            wait_mark = 1;
            sigaction(SIGINT, &sa, NULL);
            waiting();
            kill(pid2, SIGUSR1);
            kill(pid1, SIGUSR2);
            wait(NULL);
            wait(NULL);
            printf("parent process is killed!\n");
            exit(0);
        }
        else
        {
            wait_mark = 1;
            signal(SIGINT, SIG_IGN);
            sigaction(SIGUSR1, &sa, NULL);
            waiting();
            printf("child process 2 is killed!\n");
            exit(0);
        }
    }
    else
    {
        wait_mark = 1;
        signal(SIGINT, SIG_IGN);
    }
}

```

```

        sigaction(SIGUSR2, &sa, NULL);
        waiting();
        printf("child process 1 is killed!\n");
        exit(0);
    }
}

```

代码运行结果如下：

```

root@localhost:~
File Edit View Terminal Tabs Help
[123abc@localhost ~]$ su -
Password:
[root@localhost ~]# touch 111.c
[root@localhost ~]# vi 111.c
[root@localhost ~]# gcc -o 111 111.c
[root@localhost ~]# ./111
child process 2 is killed!
child process 1 is killed!
parent process is killed!
[root@localhost ~]# ./111

[root@localhost ~]# ./111
child process 1 is killed!
child process 2 is killed!
parent process is killed!
[root@localhost ~]#

```

在多次运行程序的过程中，子进程1和子进程2的结束顺序可能会有所不同。这是因为父进程在发送 `kill` 指令后，两个子进程的运行是并行的。由于操作系统的调度策略、当前运行环境的资源分配以及其他并发任务的干扰等因素，子进程的执行速度和结束时间可能会存在差异。然而，由于父进程通过 `wait` 函数被阻塞，它必须等待所有子进程结束之后才会继续执行，因此父进程不会在子进程结束之前提前退出。

如果需要确保子进程按照特定的顺序结束，可以使用 `waitpid` 函数来实现。`waitpid` 允许父进程明确地等待某个指定的子进程结束，从而可以精确控制子进程的结束顺序。例如，父进程可以先调用 `waitpid` 等待子进程1结束，然后再调用 `waitpid` 等待子进程2结束，这样就能保证子进程1总是先于子进程2结束。

5. 代码如下：

```

#include<stdio.h>
#include<unistd.h>
#include<signal.h>
#include<sys/wait.h>
#include<stdlib.h>

int main()
{
    int pid1, pid2;

```

```

int fd[2];
char data[100];

pipe(fd);

while((pid1 = fork()) == -1);

if(pid1 == 0)
{
    lockf(fd[1], 1, 0);
    sprintf(data, "Child process 1 is sending a message!");
    write(fd[1], data, 50);
    lockf(fd[1], 0, 0);
    sleep(1);
    exit(0);
}
else
{
    while((pid2 = fork()) == -1);

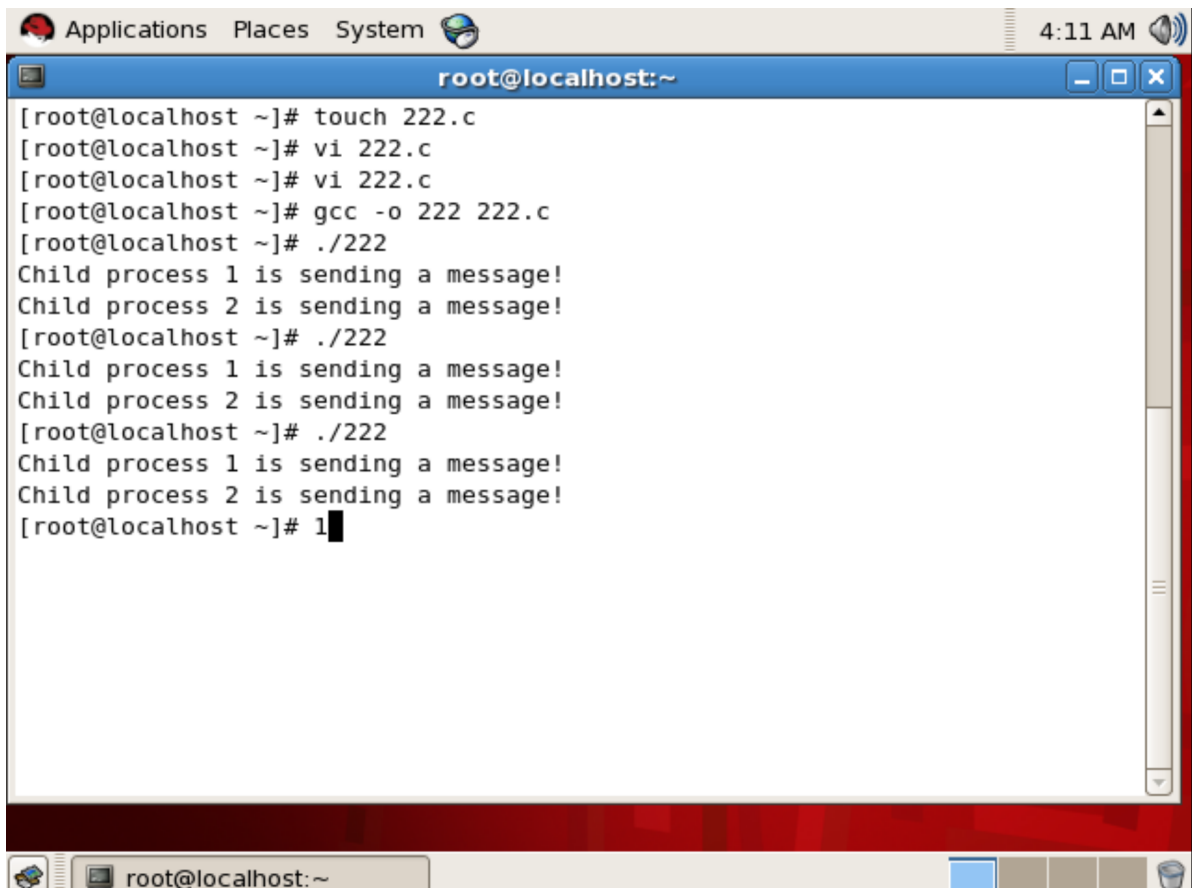
    if(pid2 == 0)
    {
        lockf(fd[1], 1, 0);
        sprintf(data, "Child process 2 is sending a message!");
        write(fd[1], data, 50);
        lockf(fd[1], 0, 0);
        sleep(1);
        exit(0);
    }
    else
    {
        wait(NULL);
        read(fd[0], data, 50);
        printf("%s\n", data);
        read(fd[0], data, 50);
        printf("%s\n", data);
        close(0);
        exit(0);
    }
}

return 0;
}

```

代码运行结果如下：





```
root@localhost:~  
[root@localhost ~]# touch 222.c  
[root@localhost ~]# vi 222.c  
[root@localhost ~]# gcc -o 222 222.c  
[root@localhost ~]# ./222  
Child process 1 is sending a message!  
Child process 2 is sending a message!  
[root@localhost ~]# ./222  
Child process 1 is sending a message!  
Child process 2 is sending a message!  
[root@localhost ~]# ./222  
Child process 1 is sending a message!  
Child process 2 is sending a message!  
[root@localhost ~]# 1
```

可以看出子进程和父进程成功实现了管道通信，并满足题目先接收子进程1的消息的要求。

## 五. 实验中出现的问题和解决

- 1.通过vim打开/etc/sudoers提示只读：应当在根用户root使用visudo来编辑该配置文件。
- 2.实验4中无论是按ctrl+C还是delete键均无响应：可能是因为在主进程中，`waiting()` 函数是一个忙等待（busy-waiting）机制，它会不断检查 `wait_mark` 是否为0。这种忙等待会占用大量CPU资源，导致系统运行缓慢。如果信号处理函数没有正确触发（例如信号被阻塞或信号处理函数未正确注册），主进程可能会陷入无限循环。因此使用 `sigaction()` 替代 `signal()`，以确保信号处理的可靠性。

## 六.实验体会

通过本次实验，我对Linux系统中的进程管理有了更深入的理解，包括进程的基本概念以及如何使用 `fork`、`wait`、`signal` 和 `pipe` 等系统调用进行进程的创建、管理和通信。此外，我还学习了Linux系统中用户和用户组管理、`sudo` 权限管理以及 `sudoers` 文件的配置，并尝试了定时任务的设置。在实验过程中，我遇到了一些问题，经过查阅资料成功将问题解决。这次实验不仅提升了我的命令行操作技能，还增强了我的问题解决能力，并进一步丰富了我的操作系统知识储备。