# Linux进程管理实验

## 实验报告

### 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。

### 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。

1. 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。

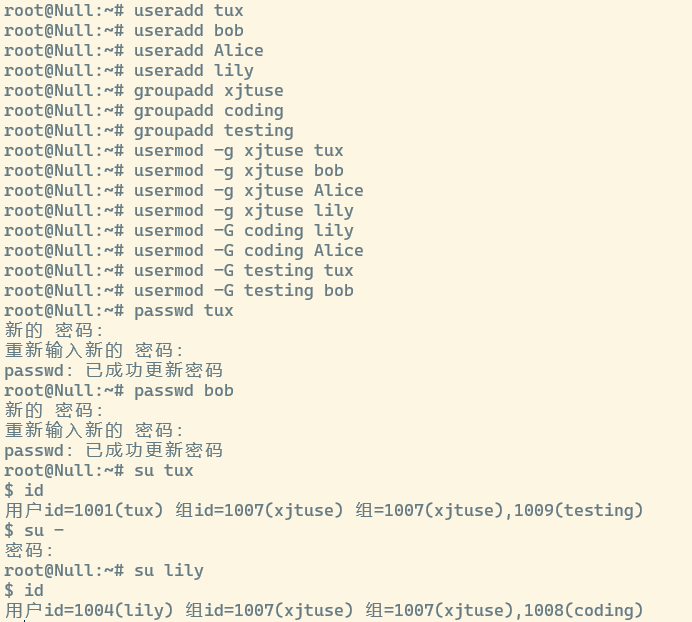
要求：父进程先接收子进程P1发来的消息，然后再接收子进程P2发来的消息。

### 题目分析及基本设计过程分析

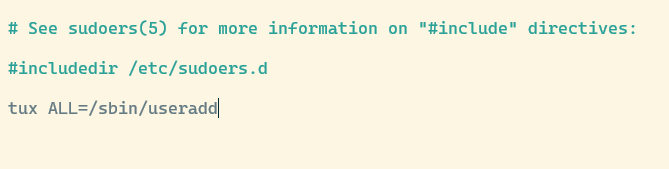
1. 使用useradd指令来添加用户、groupadd来添加组、usermod将用户加入到组、passwd修改密码。利用su进行用户切换，并利用id等指令查看用户所在的组。
2. 修改/etc/sudoers来指定用户可以或不可以输入的命令。应向该文件中添加 tux ALL=/sbin/useradd 来赋予tux添加用户的权限。
3. 使用 /etc/crontab 文件控制系统作业。应当在该文件中添加这样的作业 10 18 \* \* 5 root tar tux来在每周五的18时10分备份用户tux的数据。但由于无法等待到周五下午查看结果，将实际的作业更改为 \*/1 \* \* \* \* root tar tux表示每1分钟备份用户tux的数据。
4. 使用Linux系统中的fork、wait、exit、getpid、kill和signal等系统调用来实现进程控制。使用fork来创建子进程，并根据其返回值判断在子进程还是父进程中。在父进程中，利用signal接收软中断信号，若未接收到信号则在while循环中等待。子进程中利用signal接收父进程发送的信号，若未接收到信号也在while循环中等待。父进程接收到软中断信号后，向子进程发送终止信号，并使用wait等待子进程终止，之后将自己终止。
5. 使用pipe（int pipeid[2]）来创建一个管道。使用fork来创建和分辨子进程。在子进程中，使用write（pipeid[1],buf,size)向管道写入数据。由于管道是共享的，使用lockf将pipeid[1]保护起来，写入之后释放。在父进程中，等待两个子进程完成写入，并使用read （pipeid[0],buf,size)从管道读取数据。

### 运行截图和相关说明

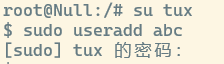
1. 新建用户tux、bob、Alice、lily，组xjtuse、coding、testing。设置用户密码并将其加入组。利用usermod -g设置主组，-G设置附加组。



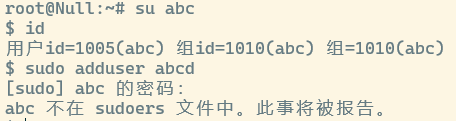
1. 修改/etc/sudoers，赋予tux添加用户的权限。



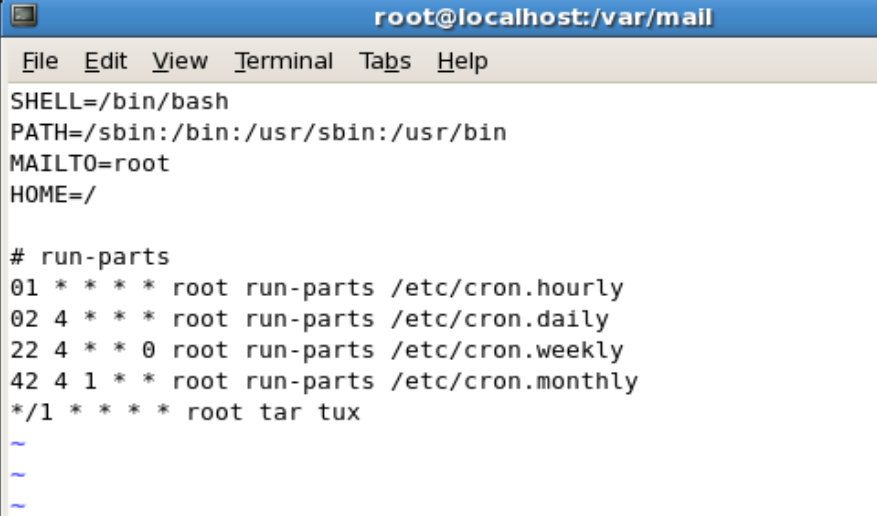
切换到用户tux，可以添加用户abc。



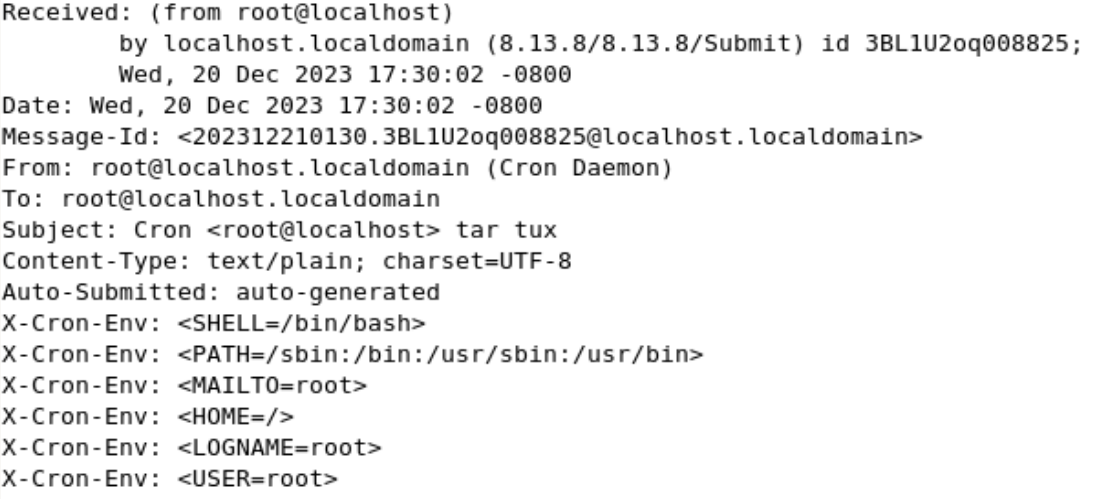
切换到用户abc，用户abc无法执行useradd。



1. 修改/etc/crontab

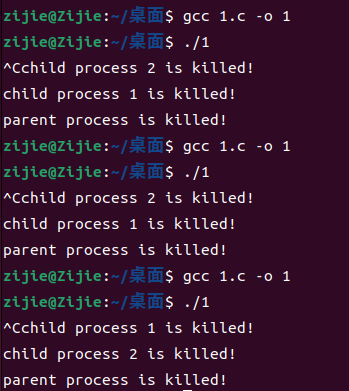


cron执行定时任务



1. 代码如下：
2. # include<stdio.h>
3. # include<signal.h>
4. # include<unistd.h>
5. # include<sys/wait.h>
6. #include<stdlib.h>
7. int wait\_mark;
8. void waiting()
9. {
10. while(wait\_mark!=0);
11. }
12. void stopwaiting()
13. {
14. wait\_mark=0;
15. }
16. int main()
17. {
18. int pid1,pid2;
19. while((pid1=fork())==-1);
20. if(pid1>0)
21. {
22. while((pid2=fork())==-1);
23. if(pid2>0)
24. { *//Parent process*
25. wait\_mark=1;
26. signal(SIGINT,stopwaiting);
27. waiting();
28. kill(pid2,16);
29. kill(pid1,17);
30. wait(0);
31. wait(0);
32. printf("parent process is killed!\n");
33. exit(0);
34. }
35. else
36. {*//Child process 2*
37. wait\_mark=1;
38. signal(SIGINT,SIG\_IGN);
39. signal(16,stopwaiting);
40. waiting();
41. printf("child process 2 is killed!\n");
42. exit(0);
43. }
44. }
45. else
46. {*//Child process 1*
47. wait\_mark=1;
48. signal(SIGINT,SIG\_IGN);
49. signal(17,stopwaiting);
50. waiting();
51. printf("child process 1 is killed!\n");
52. exit(0);
53. }
54. }

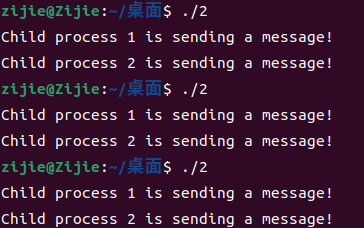
代码运行结果如下：



可以看出，三次运行中，子进程1和子进程2结束的先后顺序可能存在差异。这是因为父进程发送kill指令后，两个子进程的运行是并行的，由于系统调度、运行环境等差异，它们的结束时间可能存在差异。但由于父进程调用了wait而被阻塞，父进程不会在它们结束之前中止。如果需要指定子进程结束的顺序，可以使用waitpid等函数等待前一进程的终止。

1. 代码如下：
2. #include<stdio.h>
3. #include<unistd.h>
4. #include<signal.h>
5. #include<sys/wait.h>
6. #include<stdlib.h>
7. int main()
8. {
9. int pid1,pid2;
10. int fd[2];
11. char data[100];
12. pipe(fd);               *//创建管道*
13. while((pid1 = fork()) == -1);   *//创建子进程*
14. if(pid1 == 0)                   *//子进程1*
15. {
16. lockf(fd[1],1,0);
17. sprintf(data,"Child process 1 is sending a message!");
18. write(fd[1],data,50);
19. lockf(fd[1],0,0);
20. sleep(1);
21. exit(0);
22. }
23. else
24. {
25. while((pid2 = fork()) == -1);   *//创建子进程*
26. if(pid2 == 0){                   *//子进程2*
27. lockf(fd[1],1,0);
28. sprintf(data,"Child process 2 is sending a message!");
29. write(fd[1],data,50);
30. lockf(fd[1],0,0);
31. sleep(1);
32. exit(0);
33. }
34. else{*//父进程*
35. wait(0);
36. read(fd[0],data,50);
37. printf("%s\n",data);
38. read(fd[0],data,50);
39. printf("%s\n",data);
40. close(0);
41. exit(0);
42. }
44. }
45. return  0;
46. }

代码运行结果如下：

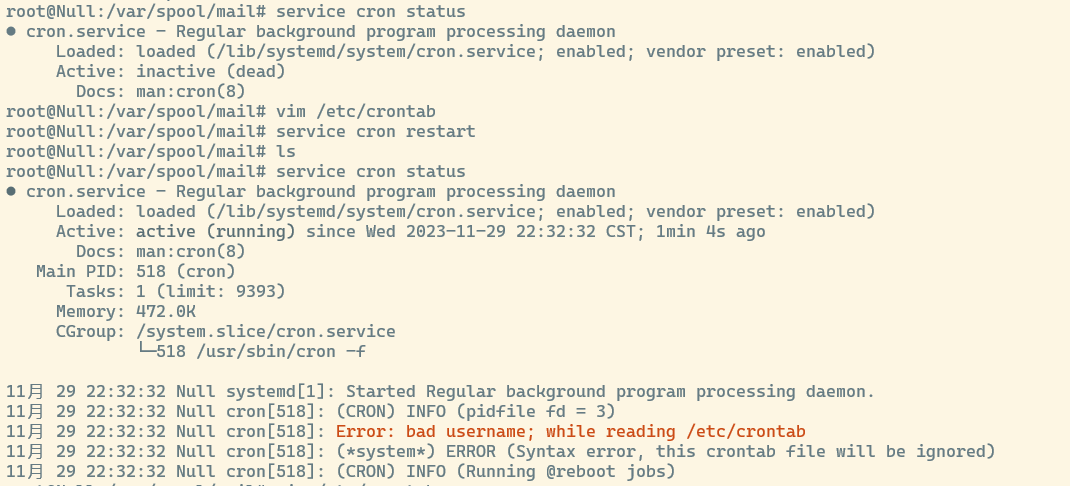


可以看出子进程和父进程成功实现了管道通信，并满足题目先接收子进程1的消息的要求。

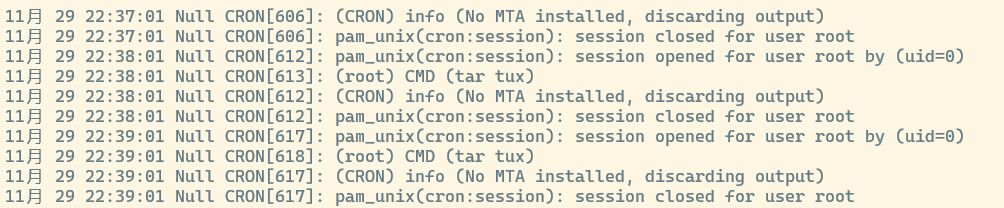
### 实验中出现的问题和解决

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. 设置了cron定时任务后，收不到发送的邮件。

首先查看了cron服务的状态，发现没有启动，利用指令service cron restart将其启动。

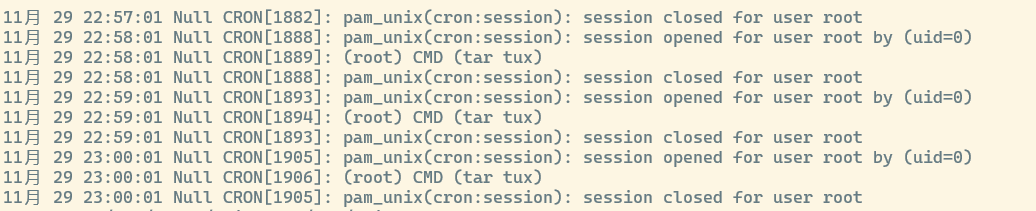


启动后服务报错Error: bad username; while reading /etc/crontab。检查文件发现，作业的第五个值应该填写用户名root而忘记填写。修改后，仍无法收到邮件。



此时服务提示(CRON) info (No MTA installed, discarding output)。根据网络搜索结果，似乎是未安装postfix所致。然而利用apt安装并启动服务后，仍收不到邮件。

查看cron的日志得知此时cron的定时任务已在执行，收不到邮件可能是ubuntu系统的某种特性。（Windows Subsystem for Linux和VMware Workstation中的Ubuntu均不行，但定时任务可以正常执行）



因此最终更换为Red Hat系统执行该操作，能够成功收到执行邮件。

1. 运行所编写的程序，按下delete时程序无响应。这是因为Ctrl-C才是发送SIG\_INT软中断信号。

按下Ctrl-C后，只有父进程中止提示，没有子进程中止提示。经过调试发现，这是因为没有在子进程中指定SIG\_INT对应的操作。在Linux中，SIG\_INT默认对应的操作是直接中止程序，所以子进程在接收到软中断信号后，直接中止了。应当在子进程部分指定signal(SIG\_INT,SIG\_IGN)将该信号忽略。

### 实验体会

本次实验使我学习到了Linux系统中进程的相关概念、进程管理中fork、wait、signal、pipe等的方法和用户和用户组管理、sudo、sudoer、定时任务等Linux系统指令。在进行定时任务相关实验时，遇到了一些无法解决的问题，初步推断是ubuntu系统原因，更换为Red Hat系统完成了实验。实验增进了我的命令行操作能力、问题解决能力和操作系统知识储备。