操作系统实验报告

班级：计算机科学与技术 21(4)班 姓名：陈昊天 学号：2021329600006

实验一： 熟悉 Linux 命令及进程管理

一、 实验目的

加深对进程概念的理解，明确进程和程序的区别。
进一步认识并发执行的实质。
分析进程争用资源的现象，学习解决进程互斥的方法。

二、 实验内容和步骤

2.1 进入和退出系统

登录系统

```
ubuntu@VM-4-13-ubuntu:~$ ssh -p 2200 root@chenhaotian.top
The authenticity of host '[chenhaotian.top]:2200 ([124.223.64.95]:2200)' can't be established.
ED25519 key fingerprint is SHA256:h1F3ssbUfftbty9EiCOPQei3BYasCV4YVu8WXYjj4sk.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[chenhaotian.top]:2200' (ED25519) to the list of known hosts.
root@chenhaotian.top's password:
Linux VM-4-11-debian 5.10.0-19-amd64 #1 SMP Debian 5.10.149-2 (2022-10-21) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Dec  8 13:36:05 2023 from 124.223.64.95
root@VM-4-11-debian:~#
```

退出系统

```
root@VM-4-11-debian:~# exit
注销
Connection to chenhaotian.top closed.
ubuntu@VM-4-13-ubuntu:~$
```

修改口令

```
root@VM-4-11-debian:~# passwd
新的 密码：
重新输入新的 密码：
passwd：已成功更新密码
root@VM-4-11-debian:~#
```

2.2 文件与目录操作

显示文件目录命令 ls

```
root@VM-4-11-debian:~# ls
gost  hexo  install.sh  kuma_install.sh  nezha.sh  ping0  uptime  uptime-kuma
```

改变当前目录命令 cd

```
root@VM-4-11-debian:~# cd /etc
root@VM-4-11-debian:/etc# cd
```

建立子目录 mkdir

```
root@VM-4-11-debian:~# mkdir os_test
root@VM-4-11-debian:~# ls
gost  hexo  install.sh  kuma_install.sh  nezha.sh  os_test  ping0  uptime  uptime-kuma
```

删除子目录命令 rmdir

```
root@VM-4-11-debian:~# rmdir os_test
root@VM-4-11-debian:~# ls
gost  hexo  install.sh  kuma_install.sh  nezha.sh  ping0  uptime  uptime-kuma
```

删除文件命令 rm

```
root@VM-4-11-debian:~# touch 1.txt
root@VM-4-11-debian:~# ls
1.txt  gost  hexo  install.sh  kuma_install.sh  nezha.sh  ping0  uptime  uptime-kuma
root@VM-4-11-debian:~# rm 1.txt
root@VM-4-11-debian:~# ls
gost  hexo  install.sh  kuma_install.sh  nezha.sh  ping0  uptime  uptime-kuma
```

文件改名命令 mv

```
root@VM-4-11-debian:~# touch 1.txt
root@VM-4-11-debian:~# ls
1.txt  gost  hexo  install.sh  kuma_install.sh  nezha.sh  ping0  uptime  uptime-kuma
root@VM-4-11-debian:~# mv 1.txt 2.txt
root@VM-4-11-debian:~# ls
2.txt  gost  hexo  install.sh  kuma_install.sh  nezha.sh  ping0  uptime  uptime-kuma
```

文件复制命令 cp

```
root@VM-4-11-debian:~# cp 2.txt 3.txt
root@VM-4-11-debian:~# ls
2.txt  3.txt  gost  hexo  install.sh  kuma_install.sh  nezha.sh  ping0  uptime  uptime-kuma
```

显示文件的内容 more 或者 less

```
root@VM-4-11-debian:~#
root@VM-4-11-debian:~# more /etc/nginx/sites-enabled/default
```

```
server {
        listen 80 default_server;
        return 403;
}

server {
        server_name www.chenhaotian.top;
        return 301 https://chenhaotian.top$request_uri;
    listen 80;
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/www.chenhaotian.top/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/www.chenhaotian.top/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
        listen 3000 ssl;
        server_name chenhaotian.top www.chenhaotian.top;
        location / {
                proxy_pass http://127.0.0.1:3001;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ =404;
        }
    ssl_certificate /etc/letsencrypt/live/www.chenhaotian.top/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/www.chenhaotian.top/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {
    listen 80;
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/www.chenhaotian.top/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/www.chenhaotian.top/privkey.pem; # managed by Certbot
--更多--(55%)
[1] 0:more*                                                    "VM-4-11-debian" 13:49 08-12月-23
```

查找文件 find

```
root@VM-4-11-debian:~# find /root -name 2.txt
/root/2.txt
root@VM-4-11-debian:~#
```

重定向与管道

```
root@VM-4-11-debian:~# echo "Hello, World!" > 1.txt
root@VM-4-11-debian:~# cat 1.txt
Hello, World!
```

```
root@VM-4-11-debian:~# vim 1.txt
root@VM-4-11-debian:~# cat 1.txt
Hello, World!
123
456
778
12
333
root@VM-4-11-debian:~# cat 1.txt | grep "3"
123
333
root@VM-4-11-debian:~#
```

2.3 进程管理及作业控制

启动进程

前台启动

```
root@VM-4-11-debian:~/os_test# sleep 300
```

后台启动

```
root@VM-4-11-debian:~/os_test# sleep 300 &
[2] 758127
root@VM-4-11-debian:~/os_test#
```

恢复到前台运行

```
root@VM-4-11-debian:~/os_test# fg %2
sleep 300
```

2.4 进程查看

who 命令

```
root@VM-4-11-debian:~# who
root     pts/0        2023-12-08 13:31 (120.199.34.115)
root     pts/1        2023-12-08 14:16 (tmux(758522).%0)
root     pts/2        2023-12-08 13:33 (120.199.34.115)
root     pts/3        2023-12-08 13:34 (124.160.201.20)
```

ps 命令

```
root@VM-4-11-debian:~# ps
    PID TTY          TIME CMD
 758523 pts/1    00:00:00 bash
 758542 pts/1    00:00:00 ps
```

top 命令

```
top - 14:17:31 up 201 days, 14:37,  4 users,  load average: 0.06, 0.02, 0.00
任务:  96 total,   1 running,  95 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.0 us,  1.0 sy,  0.0 ni, 97.7 id,  0.3 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   1982.3 total,     99.5 free,    408.7 used,   1474.1 buff/cache
MiB Swap:   4096.0 total,   3945.4 free,    150.6 used.   1403.4 avail Mem

  进程号 USER      PR  NI    VIRT    RES    SHR    %CPU  %MEM     TIME+ COMMAND
 767358 root      20   0   11.0g 169840  17984 S   3.0   8.4   5302:19 node /root/upti
 758371 root      20   0       0      0      0 I   0.3   0.0   0:00.18 kworker/1:0-events
4183821 www-data  20   0   68900  15972  13244 S   0.3   0.8   2:43.15 nginx
      1 root      20   0  165172   8008   5712 S   0.0   0.4   8:26.27 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:04.24 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
```

2.5 在线帮助

man 命令

```
LS(1)                                   User Commands                                   LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by default).  Sort entries alphabetically if none
       of -cftuvSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

       -B, --ignore-backups
              do not list implied entries ending with ~

       -c     with -lt: sort by, and show, ctime (time of last modification of file  status  information);  with
              -l: show ctime and sort by name; otherwise: sort by ctime, newest first

       -C     list entries by columns

       --color[=WHEN]
              colorize  the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info be-
              low
 Manual page ls(1) line 1 (press h for help or q to quit)
[0] 0:man*                                                    "VM-4-11-debian" 14:18 08-12月-23
```

help 命令

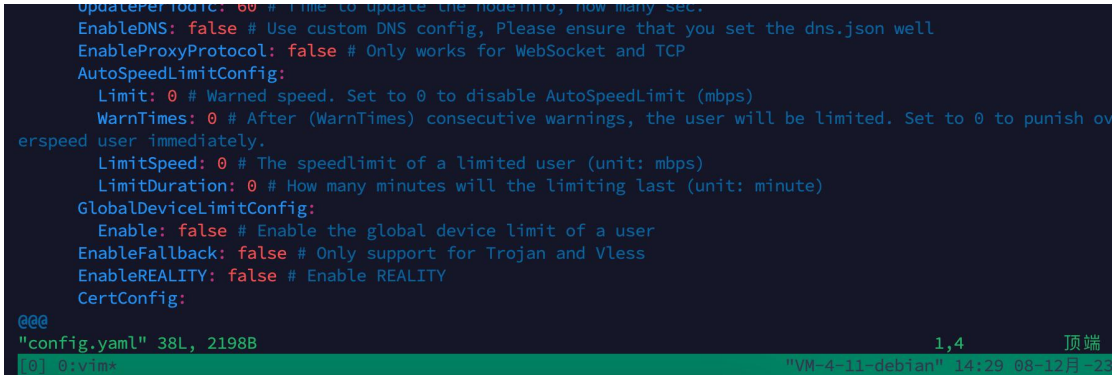
```
root@VM-4-11-debian:~# help
GNU bash, 版本 5.1.4(1)-release (x86_64-pc-linux-gnu)
这些 shell 命令是内部定义的。请输入 `help' 以获取一个列表。
输入 `help 名称' 以得到有关函数`名称'的更多信息。
使用 `info bash' 来获得关于 shell 的更多一般性信息。
使用 `man -k' 或 `info' 来获取不在列表中的命令的更多信息。

名称旁边的星号(*)表示该命令被禁用。

 job_spec [&]                                          history [-c] [-d 偏移量] [n] 或 history -anrw [文件名>
 (( 表达式 ))                                          if 命令; then 命令; [ elif 命令; then 命令; ]... [ el>
 . 文件名 [参数]                                       jobs [-lnprs] [任务声明 ...] 或 jobs -x 命令 [参数]
 :                                                     kill [-s 信号声明 | -n 信号编号 | -信号声明] 进程号 >
 [ 参数... ]                                           let 参数 [参数 ...]
 [[ 表达式 ]]                                          local [option] 名称[=值] ...
 alias [-p] [名称[=值] ... ]                           logout [n]
 bg [任务声明 ...]                                     mapfile [-d 分隔符] [-n 计数] [-O 起始序号] [-s 计数]>
 bind [-lpvsPSVX] [-m 键映射] [-f 文件名] [-q 名称] [-u>  popd [-n] [+N | -N]
 break [n]                                             printf [-v var] 格式 [参数]
 builtin [shell 内建 [参数 ...]]                       pushd [-n] [+N | -N | 目录]
```

whereis 命令

```
root@VM-4-11-debian:~# whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz
root@VM-4-11-debian:~#
```

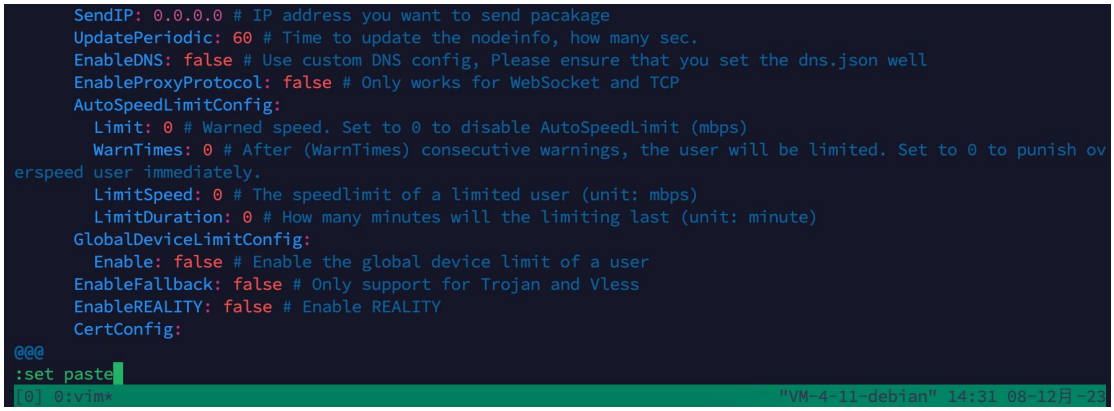
2.5 全屏幕文本编辑器 Vim

命令行模式



文本输入模式

末行模式



三、 代码及运行结果分析

程序 1

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

int main() {
    int i, j, k = -1;
```

```
    int status;   // 用于存储子进程的退出状态
    if ((i = fork()) > 0) {
        // 在父进程中
        j = wait(&status);   // 返回结束子进程的 PID
        printf("Parent Process! My PID is %d.\n", getpid());   // 打印父进程的 PID
        printf("i=%d,j=%d,k=%d\n", i, j, k);
    } else if (i == 0) {
        // 在子进程中
        k = getpid();
        printf("Child Process! My parent's PID is %d.\n",
                getppid());   // 打印父进程的 PID
        printf("i=%d,k=%d\n\n", i, k);
    } else {
        // fork 失败
        perror("fork failed");
        return 1;
    }
    return 0;
}
```

运行结果：

```
Child Process! My parent's PID is 71036.
i=0,k=71037

Parent Process! My PID is 71036.
i=71037,j=71037,k=-1
```

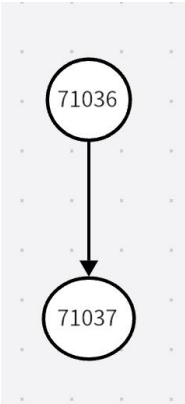
分析：

程序首先调用 fork() 创建一个子进程。fork() 返回的值是在父进程中是新创建的子进程的进程 ID（PID），而在子进程中返回 0。

在子进程中，i =0（fork() 在子进程中返回 0），k =子进程的 PID。

在父进程中，i =子进程的 PID，而 k 保持初始值-1。父进程调用 wait() 等待子进程结束。wait() 返回结束子进程的 PID，存储在 j 中。

进程树：



程序 2

```
#include <stdio.h>
```

```
#include <sys/wait.h>
#include <unistd.h>
main() {
    int p1, p2;
    while ((p1 = fork()) == -1)
        ;
    if (p1 == 0)
        printf("b.My process ID is %d\n", getpid());
    else {
        while ((p2 = fork()) == -1)
            ;
        if (p2 == 0)
            printf("c.My process ID is %d\n", getpid());
        else
            printf("a.My process ID is %d\n", getpid());
    }
}
```

运行结果：

```
a.My process ID is 71209
b.My process ID is 71210
c.My process ID is 71211
```

分析：

程序开始执行 while 循环，创建一个子进程。

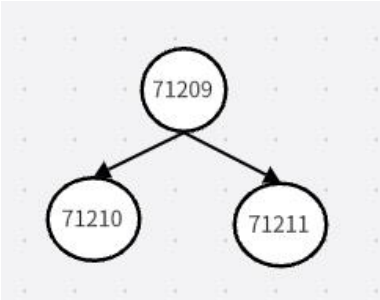
如果 fork() 调用成功，程序会检查 p1 的值来确定它是在父进程中还是子进程中执行。

如果 p1 是 0，说明当前是 p1 的子进程。

如果 p1 不是 0，说明当前是父进程，通过 while 创建第二个子进程。

父进程中的 p2 = 子进程的 PID，子进程中 p2 = 0。

进程树：



程序 3

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
main() {
    int m, n, k;
    m = fork();
```

```
    printf("PID:%d\t", getpid());
    printf("The return value of fork():%d\t\t", m);
    printf("he\n");
    n = fork();
    printf("PID:%d\t", getpid());
    printf("The return value of fork():%d\t\t", n);
    printf("ha\n");
    k = fork();
    printf("PID:%d\t", getpid());
    printf("The return value of fork():%d\t\t", k);
    printf("ho\n");
}
```

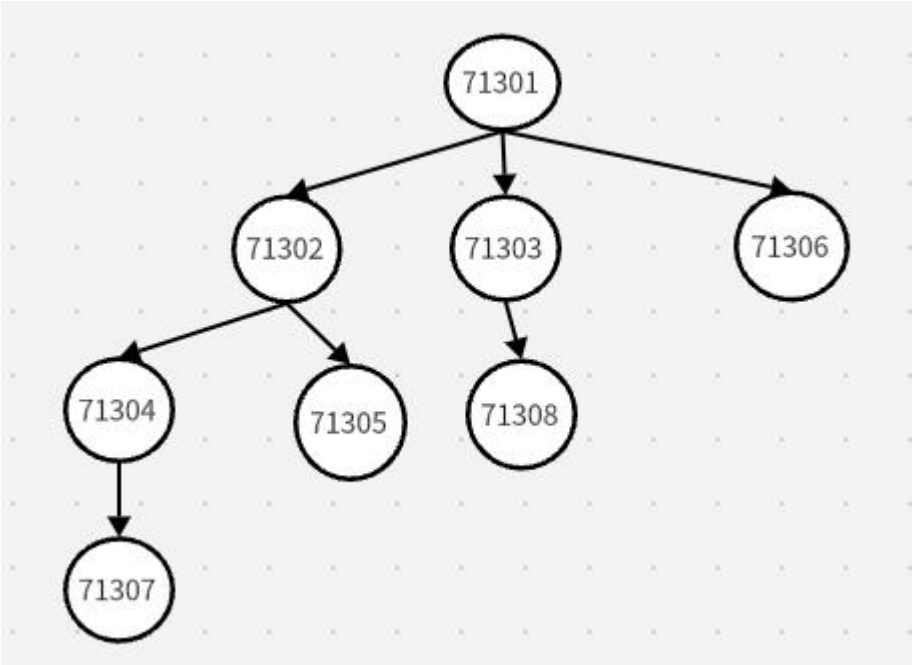
运行结果：

```
PID:71301       The return value of fork():71302                he
PID:71302       The return value of fork():0            he
PID:71302       The return value of fork():71304                ha
PID:71301       The return value of fork():71303                ha
PID:71304       The return value of fork():0            ha
PID:71301       The return value of fork():71306                ho
PID:71303       The return value of fork():0            ha
PID:71306       The return value of fork():0            ho
PID:71302       The return value of fork():71305                ho
PID:71305       The return value of fork():0            ho
PID:71304       The return value of fork():71307                ho
PID:71307       The return value of fork():0            ho
PID:71303       The return value of fork():71308                ho
PID:71308       The return value of fork():0            ho
```

分析：

有三个 fork()调用，创建 7 个新的子进程，加上原始的父进程，共有 8 个进程。

进程树：



程序 4

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

main() {
    int p1, p2, i;
    while ((p1 = fork()) == -1)
        ;
    if (p1 == 0)
        for (i = 0; i < 50000; i++) printf("son%d\n", i);
    else {
        while ((p2 = fork()) == -1)
            ;
        if (p2 == 0)
            for (i = 0; i < 50000; i++) printf("daughter%d\n", i);
        else
            for (i = 0; i < 50000; i++) printf("parent%d\n", i);
    }
}
```

运行结果：

```
son49610
daughter49382
son49611
daughter49383
son49612
dason49613
son49614
son49615
son49616
son49617
son49618
son49619
son49620
son49621
ughter49384
son49622
son49623
daughter49385
son49624
daughter49386
son49625
daughter49387
son49626
daughter49388
son49627
daughter49389
son49628
daughter49390
son49629
daughter49391
son49630
daughter49392
son49631
daughter49393
```

分析：
终端上的输出是 "son"、"daughter" 和 "parent" 消息的混合，每个消息后面跟着相应的迭代次数。由于进程是并行执行的，它们的打印操作将交织在一起，具体的输出顺序是不确定的，取决于操作系统的进程调度。

程序 5

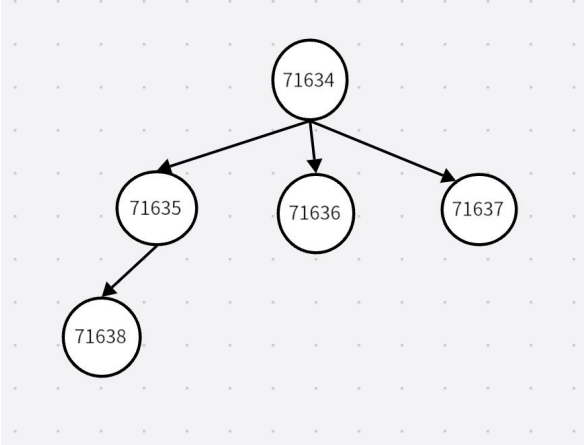
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main() {
    pid_t pid1, pid2, pid3, pid4;
    // P0 创建 P1
    pid1 = fork();
    if (pid1 < 0) {
        perror("fork failed");
        exit(1);
    } else if (pid1 == 0) {
        // P1
        printf("P1 with PID %d from PPID %d.\n", getpid(), getppid());
        // P1 创建 P4
        pid4 = fork();
        if (pid4 < 0) {
            perror("fork failed");
            exit(1);
        } else if (pid4 == 0) {
            // P4
            printf("P4 with PID %d from PPID %d.\n", getpid(), getppid());
            exit(0);
        }
        // P1 等待 P4 结束
        wait(NULL);
        exit(0);
    } else {
        // P0 创建 P2
        pid2 = fork();
        if (pid2 < 0) {
            perror("fork failed");
            exit(1);
        } else if (pid2 == 0) {
            // P2
            printf("P2 with PID %d from PPID %d.\n", getpid(), getppid());
            exit(0);
        } else {
```

```
            // P0  创建  P3
            pid3 = fork();
            if (pid3 < 0) {
                perror("fork failed");
                exit(1);
            } else if (pid3 == 0) {
                // P3
                printf("P3 with PID %d from PPID %d.\n", getpid(), getppid());
                exit(0);
            }
        }
    }
    wait(NULL);
    wait(NULL);
    wait(NULL);
    printf("P0 with PID %d.\n", getpid());
    return 0;
}
```

运行结果：

```
P1 with PID 73197 from PPID 73196.
P4 with PID 73199 from PPID 73197.
P2 with PID 73198 from PPID 73196.
P3 with PID 73200 from PPID 73196.
P0 with PID 73196.
```

进程图：



四、 心得体会

通过参与这次操作系统实验，我深刻体会到了理论知识与实际操作相结合的重要性。实验让我从实践中更加清晰地理解了进程的概念，包括进程与程序之间的区别、进程的并发执行以及进程争用资源的现象。我学会了如何通过编写 C 语言程序来模拟进程的创建、执行和同步，这不仅加深了我对进程控制的理解，也锻炼了我的编程能力。

在实验中，我熟悉了 Linux 命令行的操作，掌握了文件和目录管理、进程查看和作业控制等

多种基本命令。通过不断的实践，我对这些命令的功能和使用场景有了更加深刻的认识。实验过程中遇到的问题和困难，比如进程创建失败或资源竞争引发的问题，都是宝贵的学习机会。我通过查找资料、分析问题和尝试不同的解决方案，提高了自己解决实际问题的能力。

使用全屏幕文本编辑器 Vim 进行代码编写和编辑的经历，也极大地提升了我的文本处理技能。我意识到，掌握这些基本工具对于未来进行更高级的系统开发和维护工作至关重要。

这次实验不仅让我对操作系统的进程管理有了更加深入的认识，也增强了我使用 Linux 环境进行开发的实践能力。我相信这些经验将为我的计算机科学学习之路打下坚实的基础。