****

**操作系统及安全**

**实 验 报 告**

**实验单元：** 进程控制

**班 级：** 4211090901

**学 号：**

**姓 名：**

# 实验目的

1. 认识程序和进程；
2. 掌握常用的进程管理的命令；
3. 掌握fork系统调用并能编写并发程序。

# 实验内容

1. 利用 man ps 详细了解 ps 命令的使用方法

命令：man ps

截图：

1. 命令 ps aux

解释：

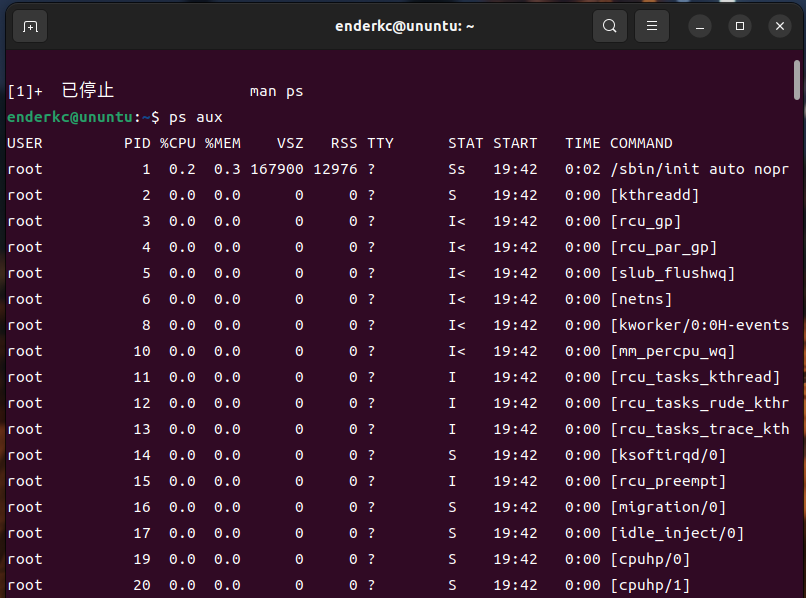
ps：用于显示进程信息的命令。

a：显示所有用户的进程，而不仅仅是当前用户的进程。

u：以详细格式显示进程信息，包括用户、进程ID、CPU占用、内存占用、启动时间等。

x：显示没有终端控制的进程，通常是守护进程（daemons）等。

截图：

命令：ps axif

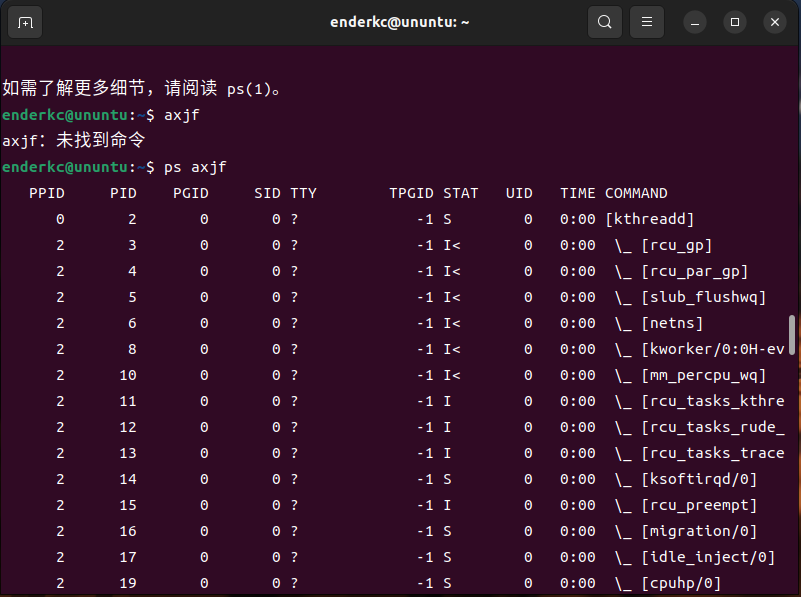
解释：

ps：用于显示进程信息的命令。

a：显示所有进程，包括其他用户的进程。

x：显示没有终端控制的进程，通常是守护进程（daemons）等。

j：以"作业控制格式"显示进程信息，包括进程ID、进程组ID、作业控制进程ID等。

截图：

命令：ps -afxo user,ppid,pid,pgid,command

解释：

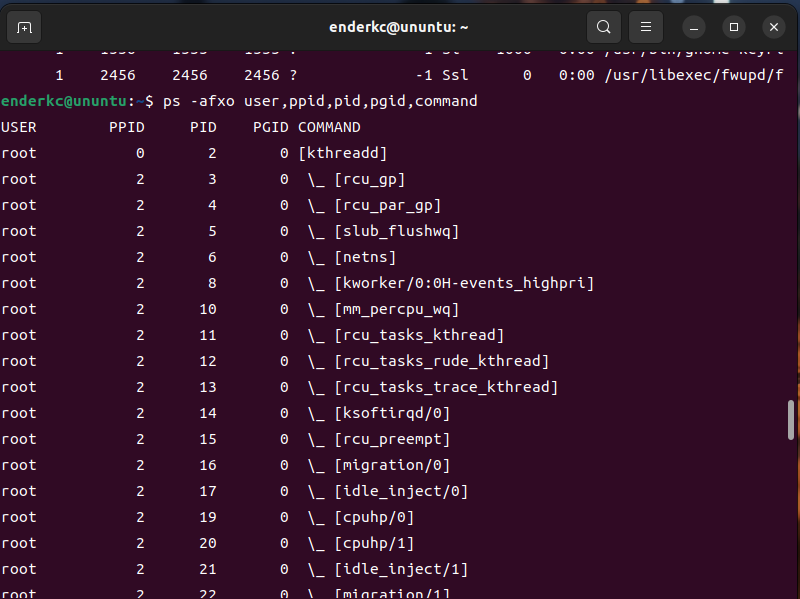
ps：用于显示进程信息的命令。

-a：显示所有进程，包括其他用户的进程。

-f：以完整格式显示进程信息。

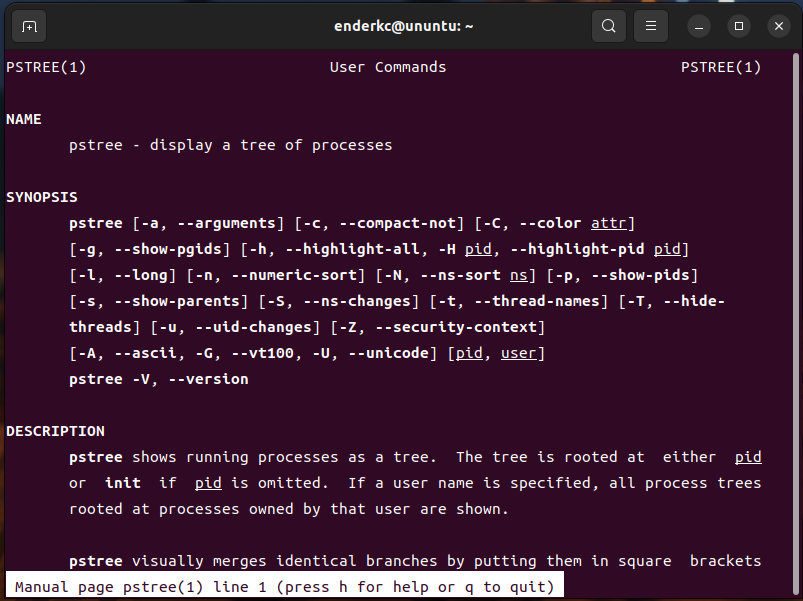
-x：显示没有终端控制的进程，通常是守护进程（daemons）等。

-o user,ppid,pid,pgid,commandv：自定义输出格式，仅显示指定的列信息，包括用户、父进程ID、进程ID、进程组ID和命令行。

截图：

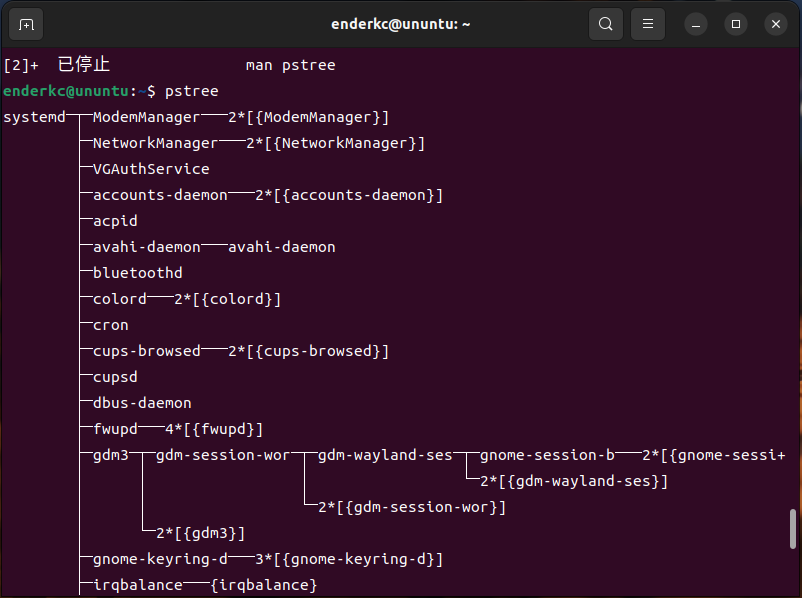
1. 请利用“man pstree”详细了解 pstree 命令的使用方法。

命令：man pstree

截图：

命令：pstree

解释：pstree 命令的主要功能是以树状结构的形式显示系统中所有进程的关系，包括父进程和子进程。通常，树状结构的根是 init 进程（通常具有进程ID 1），而其他进程都以其父进程为基础排列在树中。

截图：

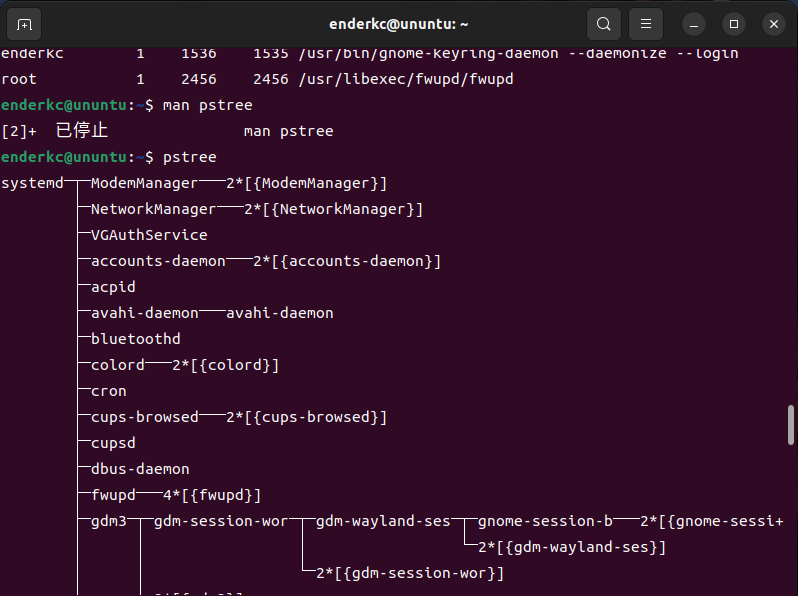
命令：pstree -up

解释：

pstree：用于显示进程树的命令。

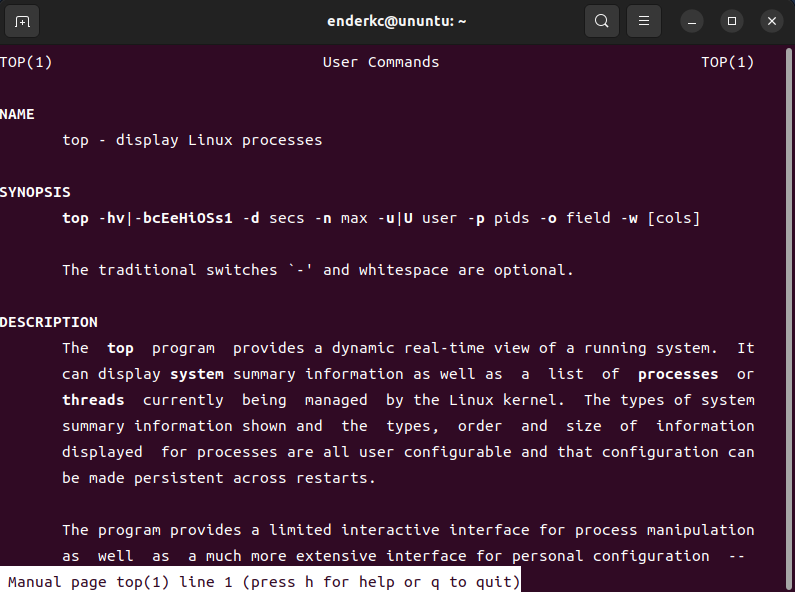
-u：显示每个进程的用户名。这会将每个进程的用户名添加到树状结构中，以帮助你了解哪个用户拥有哪些进程。

-p：显示每个进程的PID（进程ID）。这会在树状结构中显示每个进程的PID，以帮助你更清晰地识别每个进程。

截图：

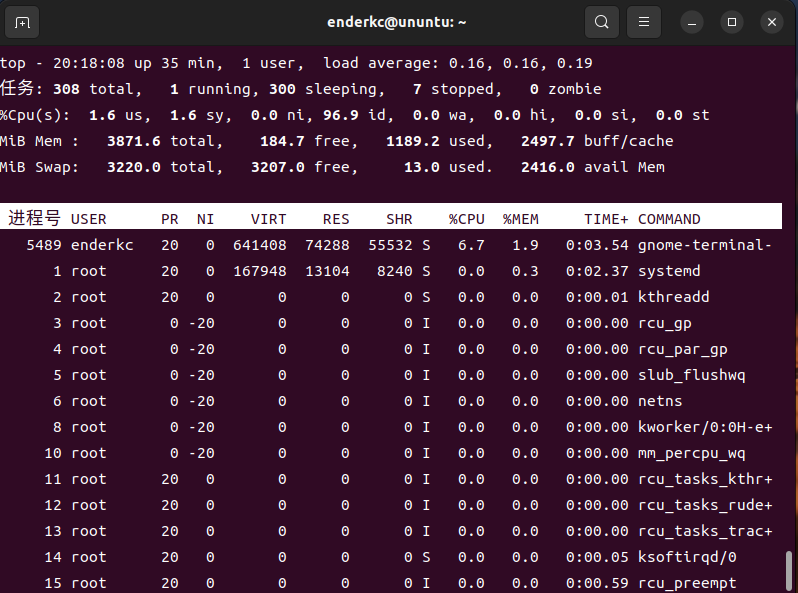
1. 请利用“man top”详细了解 top 命令的使用方法。

命令：man top

截图：

命令：top

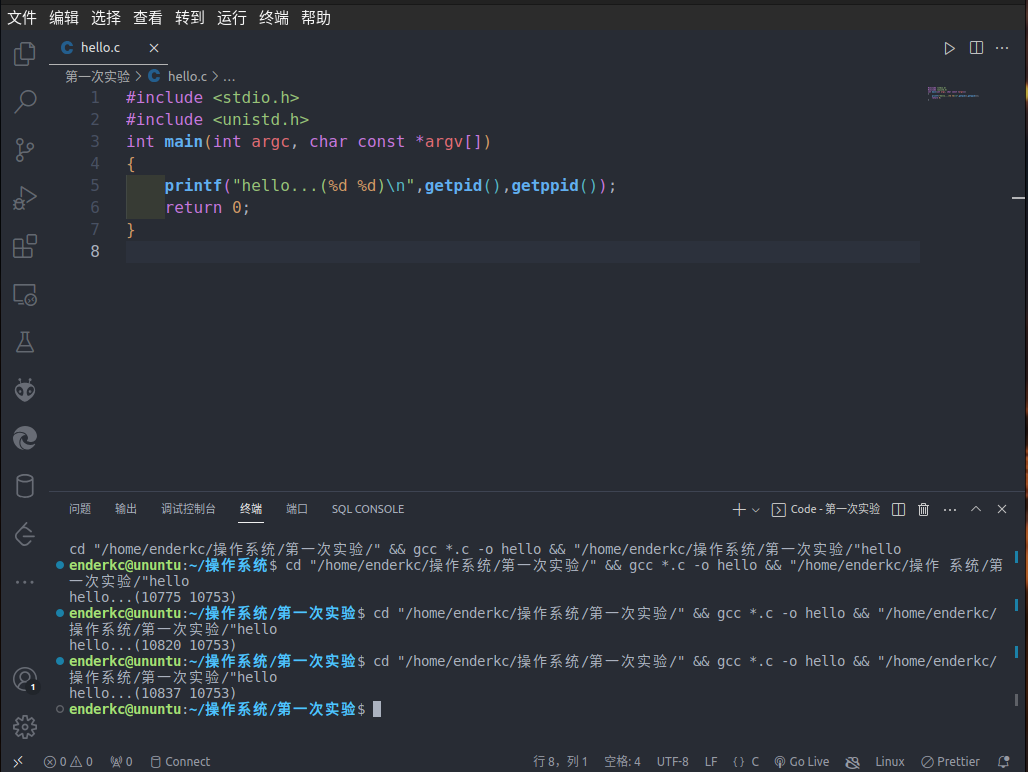
解释：top 是一个常用的Linux/Unix命令，用于实时监视系统的进程和系统资源使用情况。当你运行 top 命令时，它会显示一个动态更新的进程列表和系统性能信息。

截图：

1. 操作：

编译：gcc hello.c –o hello.out

执行：./hello.out

多次执行 hello.out，查看执行结果，并对结果进行解释。尝试回答以

1. 程序和进程的关系，是一一对应的吗？

不，程序和进程不是一一对应的。程序是你的源代码或可执行文件，而进程是程序的运行实例。你可以多次执行同一个程序，每次执行都会创建一个新的进程来运行该程序。所以，程序和进程是一对多的关系。在这个示例中，每次执行 "./hello.out" 都会创建一个新的进程，但它们都是运行相同的程序。

1. 进程是被谁创建的？

进程是由操作系统创建的。当你执行 "./hello.out" 时，操作系统会为该程序创建一个新的进程，为其分配资源，包括内存和CPU时间，并为该进程分配唯一的PID（进程ID）。这个新进程会执行 "hello.c" 程序中的代码，然后输出当前进程的PID（getpid()）和父进程的PID（getppid()）。

1. 代码：

#include <unistd.h>

#include <stdio.h>

int main(int argc, char const \*argv[])

{

\_\_pid\_t fpid;

int count = 0;

fpid = fork();

if (fpid < 0)

{

printf("error in fork!");

}

else if (fpid == 0)

{

printf("i am the child process, my process id is %d\n", getpid());

printf("我是爹的儿子\n");

count++;

}

else

{

printf("i am the paret process, my process id is %d\n", getpid());

printf("我是孩子他爹");

count++;

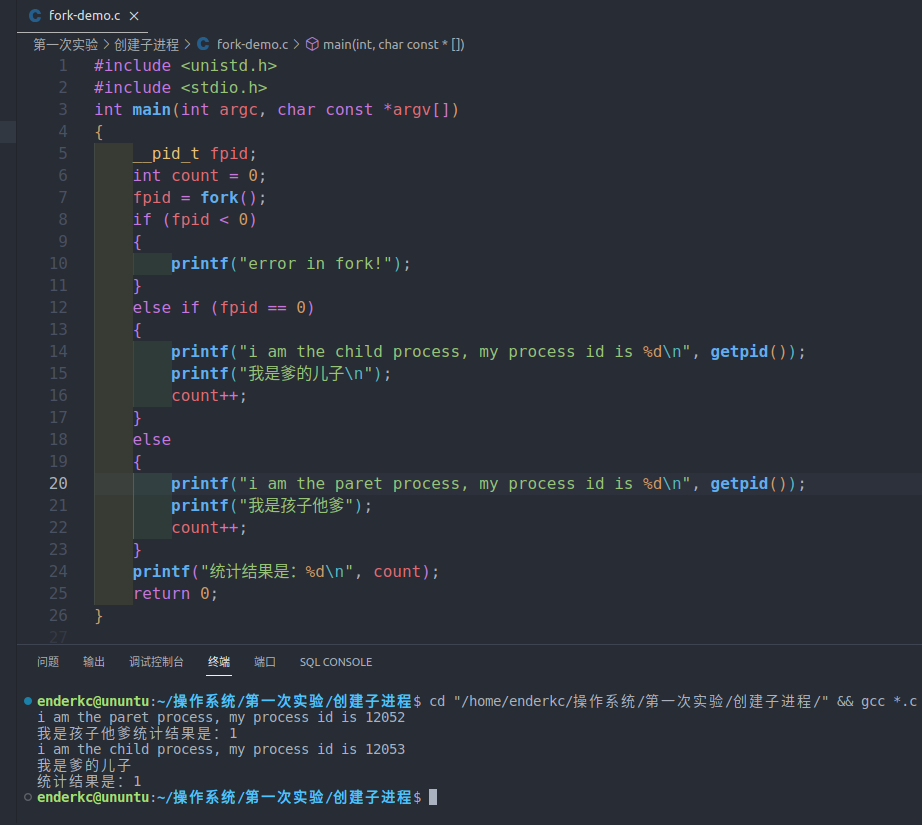
}

printf("统计结果是：%d\n", count);

return 0;

}

运行截图：

1. fork 的作用是什么？

fork 是一个在Unix和类Unix操作系统中常见的系统调用（函数），其作用是创建一个新的进程，新进程是调用进程（父进程）的副本，通常称为子进程。fork 的作用和特点如下：

创建新进程：fork 调用会创建一个与父进程几乎完全相同的子进程。子进程包括父进程的代码、数据、堆栈、文件描述符等等。子进程与父进程几乎一模一样，只有进程ID（PID）不同。

并发执行：一旦子进程创建成功，父进程和子进程将并发执行，它们独立地执行各自的代码路径，不会相互干扰。

多任务编程：fork 是实现多任务编程的重要手段之一。通过创建多个进程，可以并发执行多个任务，实现并行性。

父子关系：子进程的父进程是调用 fork 的进程，可以使用 getppid() 函数来获取子进程的父进程ID。父进程可以通过子进程的PID来识别和管理子进程。

返回值：fork 在父进程中返回子进程的PID，在子进程中返回0，这个特性可以用于区分父进程和子进程。

1. fork 的返回值有几种情况？

父进程中的返回值：在父进程中，fork 函数返回子进程的进程ID（PID）。这个PID是子进程的唯一标识符，通过它可以识别和管理子进程。因此，父进程会得到一个大于0的整数，表示子进程的PID。

子进程中的返回值：在子进程中，fork 函数返回0。这个0值表示当前代码在子进程中执行。这通常用于在子进程中执行与父进程不同的任务或程序，例如通过 exec 函数加载一个新的程序。

fork 的返回值有两种情况：

在父进程中，返回子进程的PID（大于0的整数）。

在子进程中，返回0。

(3) 在源码 1.2 中，局部变量 count 的输出结果是什么？为什么？

局部变量输出count的结果都是1

因为fork返回的结果在父进程中返回的结果是子进程的pid，子进程中fork返回的结果是0，在父进程和子进程中局部变量的值互不影响

1. fork-demo2.c

代码：

#include <unistd.h>

#include <stdio.h>

int main(int argc, char const \*argv[])

{

int i = 0;

for (i = 0; i < 2; i++)

{

\_\_pid\_t fpid = fork();

if(fpid == 0){

printf("%d child %d %d %d \n",i,getppid(),getpid(),fpid);

}else{

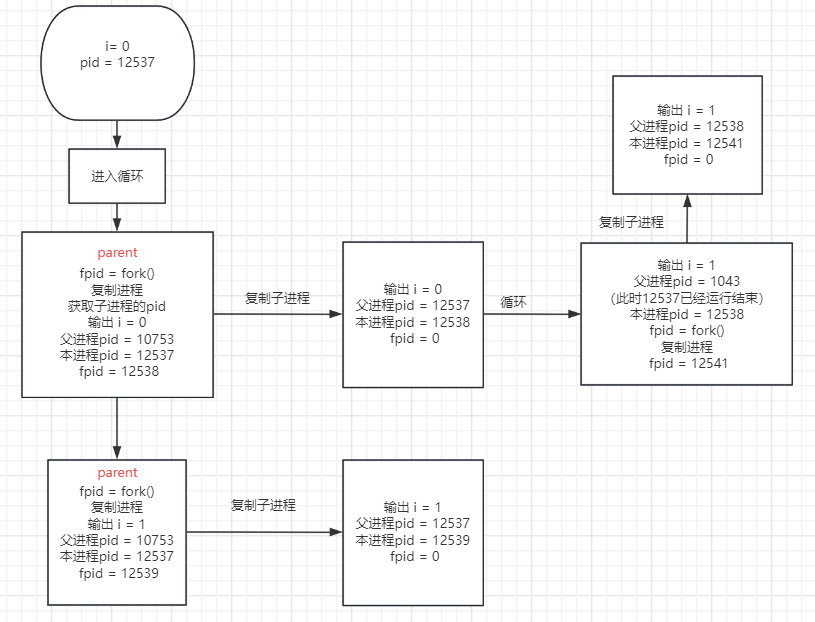
printf("%d parent %d %d %d \n",i,getppid(),getpid(),fpid);

}

}

return 0;

}

绘图：

1. 自行设计一个有意思的fork()创建子进程的实例

代码：

// 自行设计一个有意思的fork()创建子进程的实例

#include <unistd.h>

#include <stdio.h>

int **main**(int argc, char const \*argv[])

{

**printf**("Begin.....\n");

    for (int i = 0; i < 3; i++)

    {

        \_\_pid\_t fpid = **fork**();

        if (fpid == 0)

        {

**printf**("我是父进程%d的子进程%d\n",**getppid**(),**getpid**());

            break;

        }else{

**printf**("我是父进程pid = %d,创建了子进程fpid = %d\n",**getpid**(),fpid);

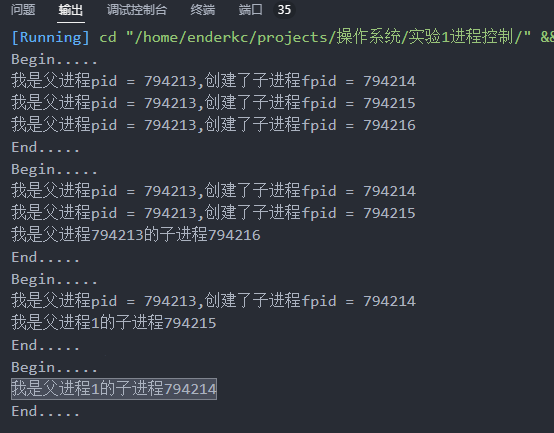
        }

    }

**printf**("End.....\n");

    return 0;

}

运行截图：

1. 利用execve替换程序的执行内核

代码：

#include <stdio.h>

#include <unistd.h>

int **main**(int argc, char\* argv[]){

        char \*args[]={"ls","-l","/home/enderkc",**NULL**};

        char \*envp[]={0,**NULL**};

**execve**("/bin/ls",args,envp);

}

运行截图：

1. 利用fork创建子进程，替换子形成的执行内核（任务自定）

代码：

#include <stdio.h>

#include <unistd.h>

int **main**(int argc, char\* argv[]){

        \_\_pid\_t fpid = **fork**();

        if (fpid ==0)

        {

**printf**("我是子进程,我的父进程pid = %d\n",**getppid**());

            char \*args[]={"ls","-l","/home/enderkc",**NULL**};

            char \*envp[]={0,**NULL**};

**execve**("/bin/ls",args,envp);

        }else{

**printf**("==========================\n");

**printf**("我是父进程pid = %d,我什么都不用做。。。。\n",**getpid**());

**printf**("==========================\n");

        }

}

运行截图：

# 实验总结

**完成该实验用到了哪些知识点（罗列）**

**完成实验过程中，自己在技术、实践能力，解决问题方面的收获**

* 学到了ps，top命令的基本使用
* Fork（）函数会复刻一个和父进程一样的子进程，同时会保留父进程的运行状态
* 父进程和子进程的概念是相对的，当子进程再创建一个进程的时候，它就变成父进程了。
* Execve可以替换程序运行的内核，可以配合fork进行多进程的操作

**遇到的问题及具体解决方案。**

* 不会分析具体程序的运行过程，解决方案：找相关博客，ChatGPT