实验3 进程控制与线程创建

实验目的

1. 掌握Linux中如何加载子进程自己的程序；

2. 掌握父进程通过创建子进程完成某项任务的方法；

3. 掌握系统调用exit()和\_exit()调用的使用；

4.掌握Linux线程的基本概念和使用。

实验内容

1. 设计两个程序hello.c和load.c，hello.c的功能是输出“hello world！”；load.c完成的功能是：创建一个子进程，子进程执行hello.c编译后的程序，父进程输出一行字符串（内容自定）后等待子进程结束（wait函数）。使用execl（）加载子进程的程序。子进程在execl前后各输出一次进程的pid，分析结果。

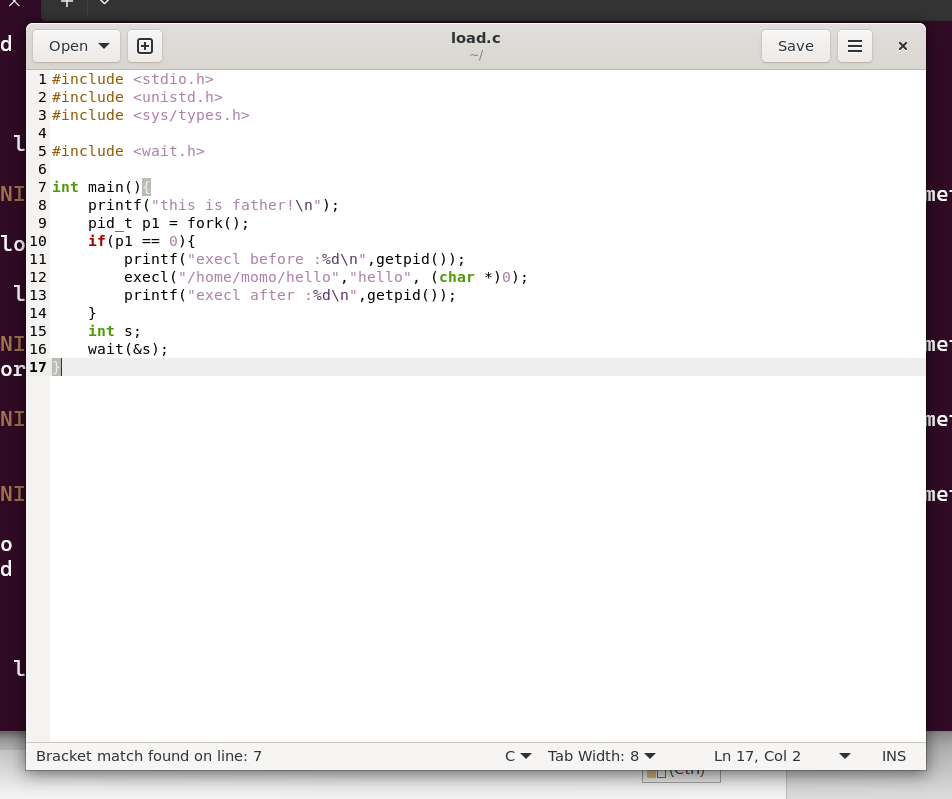
**要求：1.写出程序代码及运行截图**

1. **总结exec系列函数的作用**

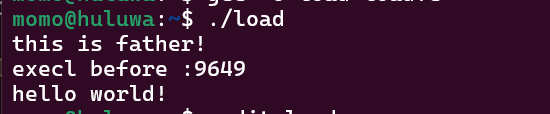
**hello.c**



load.c



结果：



当进程调用一种execl函数时，该进程完全由新程序代换，而新程序则从其main函数开始执行。因为调用execl并不创建新进程，所以前后的进程ID并未改变。execl只是用另一个新程序替换了当前进程的正文、数据、堆和栈段。

　　用另一个新程序替换了当前进程的正文、数据、堆和栈段。

当前进程的正文都被替换了，那么execl后的语句，即便execl退出了，都不会被执行。

所以输出结果上只有子进程调用execl前的pid而execl后的printf语句未被执行。

2．运行以下程序，注意程序的执行结果和printf函数（以行为单位进行缓冲的）的关系。

#include<stdlib.h>

#include <stdio.h>

#include <unistd.h>

int main()

{ printf("this is first line ");

sleep(2);

printf("this is the second line ");

sleep(2);

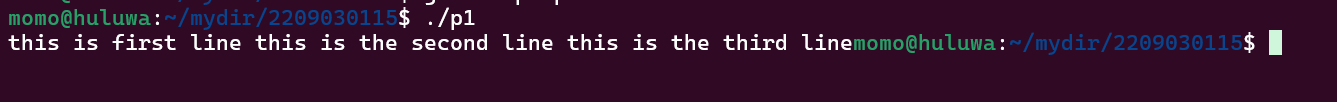
printf("this is the third line");

return 0;

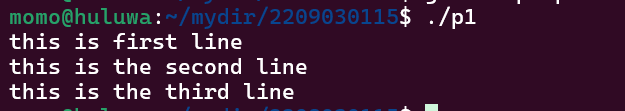
}

实验步骤：

（1）运行以上程序，查看运行结果



（2）在每条printf语句后都加上\n，再运行以上程序，观察两种结果的不同，分析printf缓冲的作用。



在使用\n时，程序会先刷新缓冲区，再进行sleep，所以结果上来看就是不使用printf缓冲时，程序会先进行sleep休眠，而后最后将所有的printf语句一块刷新打印出来。如果添加\n，则是在sleep之前即\n的时候便刷新缓冲区，即打印一句执行下面的sleep再进行打印再sleep。

**要求：写出程序代码及运行截图**

3. 运行以下程序，分析程序执行结果。

#include<stdlib.h>

#include <stdio.h>

#include <unistd.h>

main()

{ int p;

printf("This is the fisrt line");

p=fork();

if (p>0)

{ wait(0);

printf("this is parent ");

}

else{

printf("this is child first\n");

printf("this is child second");

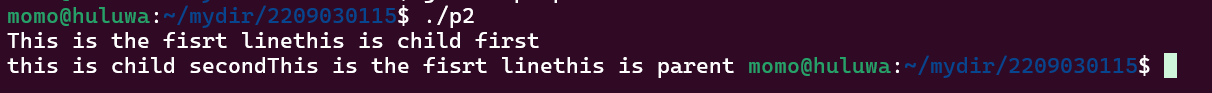
exit(0);

}

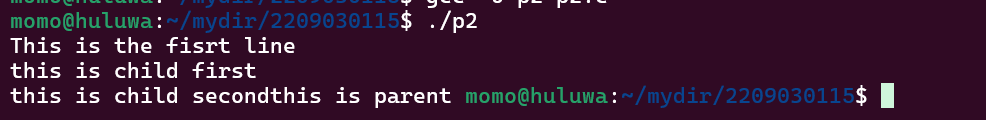
}

实验步骤：

开始时的结果：

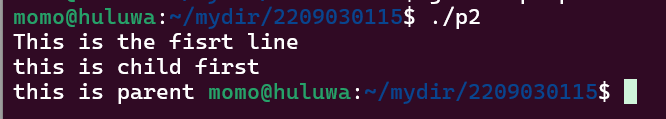


1. 分析程序的执行结果。若第一条printf语句后加上\n结果如何？



会将第一行单独输出，而不是跟着child first一块输出。

1. 若exit(0)换为\_exit(0)结果会怎样？



将会不打印子进程中的缓存。即不打印子进程中的“this is chile second ”，因为该输出语句没有添加\n，所以缓存区没有更新，而\_exit(0)在退出子程序时不会会将缓冲区的数据写完再结束进程到内核中去，而是直接结束进程进入到内核中。

1. 分析printf和fork函数，printf和\_exit函数一起使用时特点。

在不刷新printf缓冲区的情况时，（不加\n）不会立即输出内容，而是先存入缓冲区。

fork会创建子进程，子进程会复制父进程的缓冲区。所以子进程可能会将父进程缓冲区的内容随着子进程一块输出。

如果在fork之后子进程或父进程调用\_exit，则该进程中的printf缓冲区内容可能不会输出，因为\_exit不会刷新缓冲区。

为了确保printf的内容被输出，可以在调用\_exit之前使用fflush(stdout)来手动刷新缓冲区。

**要求：写出三种情况运行截图及结果分析**

1. 运行以下程序，回答问题。

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

void \*run(void \*ptr){

pid\_t pid;pthread\_t tid;

pid = getpid(); //获取进程id

tid = pthread\_self(); //获取线程id

printf("In thread,process id:%lu,thread id:%lu\n",(unsigned long)pid,(unsigned long)tid);

return 0;

}

int main(){

pthread\_t tid[3],mtid;

pid\_t pid;int i;

int ret=0;

for (i=0;i<3;i++){

ret=pthread\_create(&tid[i],NULL,run,NULL);

if(ret) {

printf("create thread failed ");

return 0;

}

}

for(i=0;i<3;i++) pthread\_join(tid[i],NULL);

pid = getpid(); //获取进程id

mtid = pthread\_self(); //获取线程id

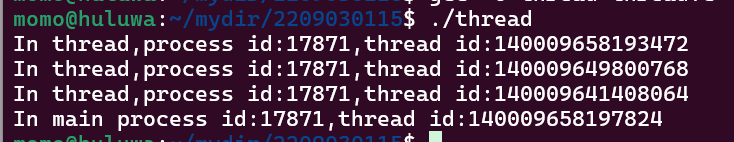
printf("In main process id:%lu,thread id:%lu\n",(unsigned long)pid,(unsigned long)mtid);

return 0;

}

1. 编辑程序thread.c，编译运行，运行结果是？

编译时使用如下命令：gcc -o thread thread.c -pthread



1. 根据程序结果分析，程序中的进程和线程有何关系？

所有的线程都属于同一个进程，具有相同的进程id，每个线程有自己的线程id。

进程是操作系统中资源分配的基本单位。在这个程序中，只创建了一个进程，即主程序main函数启动的那个进程。这个进程有一个进程ID，可以通过getpid()获取。

线程是进程内的一个执行流，是CPU调度的基本单位。在这个程序中，主进程创建了三个线程，每个线程都执行run函数。这三个线程共享同一个进程的资源，比如虚拟地址空间、打开的文件描述符等。

**要求：写出运行截图及结果分析**