

实 验 报 告

|  |  |
| --- | --- |
| **实验项目名称** | 进程控制 |
| **课程名称** | 操作系统 |
| **学生姓名** | 赵孜硕 |
| **学生学号** | 202008064710 |
| **所在班级** | 人工智能201 |
| **学科专业** | 人工智能 |
| **任课教师** | 徐飞 |
| **完成日期** | 2023年4月14日 |

# 实验一 进程控制

### 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；掌握Linux操作系统的进程创建和终止操作，体会父进程和子进程的关系及进程状态的变化；进一步认识并发执行的实质，编写并发程序。

### 二、实验平台：

虚拟机：VMware Workstation 15 Pro

操作系统：Ubuntu18.048

编辑器：Gedit | Vim

编译器：Gcc

### 三、实验内容：

（1）编写一段程序，使用系统调用fork()创建两个子进程，当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示“身份信息”：父进程显示“Parent process! PID=xxx1 PPID=xxx2”；子进程显示“Childx process! PID=xxx PPID=xxx”。多运行几次，观察记录屏幕上的显示结果，并分析原因。

说明：

xxx1为进程号，用getpid()函数可获取进程号；

xxx2为父进程号，用getppid()函数可获取父进程号；

Childx中x为1和2，用来区别两个子进程；

wait()函数用来避免父进程在子进程终止之前终止。

代码：

#include<stdio.h>

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<stdio.h>

int main(){

pid\_t Child1;

Child1=fork();

if(Child1<0){

printf("error\n");

exit(0);

}else if(Child1==0){

printf("Child1 process!\n");

printf("PID=%d PPID=%d\n",getpid(),getppid());

exit(0);

}else{

pid\_t Child2;

Child2=fork();

if(Child2<0){

printf("error\n");

exit(0);

}else if(Child2==0){

printf("Child2 process!\n");

printf("PID=%d PPID=%d\n",getpid(),getppid());

exit(0);

}else{

wait(0);

printf("Parent process!\n");

printf("PID=%d PPID=%d\n",getpid(),getppid());

exit(0);

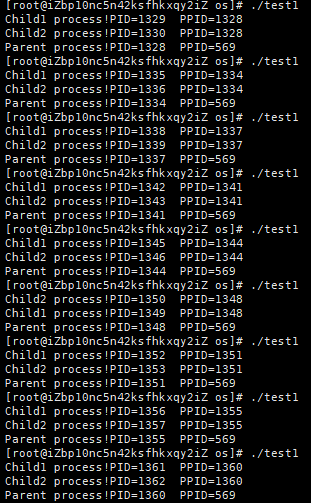
}

}

return 0;

}

运行结果：



可以观察都Child1和Child2的PPID与Parent的PID相同，这是因为Child1和Child2都是由Parent进程创建而来的，所以他们的父进程PID是Parent的PID。

1. fork()和exec()系列函数能同时运行多个程序，利用上述函数将下面单进程顺序执行的程序single.c改造成可并发执行3个进程的程序multi\_process.c；并用time命令获取程序的执行时间，比较单进程和多进程运行时间，并分析原因。

单线程代码：

#include <stdio.h>

#define NUM 5

int main(void)

{

void print\_msg(char \*m);

print\_msg("Good ");

print\_msg("Morning ");

print\_msg("007\n"); //将007替换为本人学号

return 0;

}

void print\_msg(char \*m)

{

int i;

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

printf("%s",m);

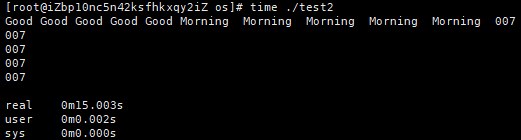
fflush(stdout);

sleep(1);

}

}

单线程single.c结果：



多线程代码：

#include<stdio.h>

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<stdio.h>

#define NUM 5

int main(){

void print\_msg(char \*m);

pid\_t Child1;

Child1=fork();

if(Child1<0){

printf("error\n");

exit(0);

}else if(Child1==0){

print\_msg(" Good ");

exit(0);

}

pid\_t Child2;

Child2=fork();

if(Child2<0){

printf("error\n");

exit(0);

}else if(Child2==0){

print\_msg(" Morning ");

exit(0);

}

print\_msg(" 202008064710 ");

printf("\n");

return 0;

}

void print\_msg(char \*m){

int i;

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

printf("%s",m);

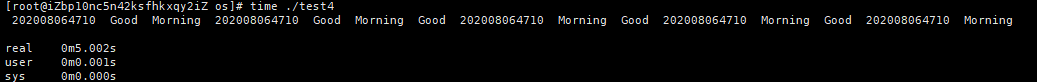
fflush(stdout);

sleep(1);

}

}

多线程multi\_process.c结果：



观察到单进程花费了15s，是多进程的三倍。原因是单进程只能一次完成一个输出所以总共需要花费15秒才能将内容全部输出。而多进程可以用三个进程共同完成任务，每个进程同时花费5秒就可以将输出完成。

## 四、实验心得

通过本次实验，我学会了进程的控制和创建线程，加深对进程概念的理解，明确了进程和程序的区别；掌握Linux操作系统的进程创建和终止操作，体会父进程和子进程的关系及进程状态的变化；进一步认识并发执行的实质，编写并发程序。还理解了fork函数的使用以及作用。受益良多！