山东大学 计算机科学与技术 学院

操作系统 课程实验报告

学号:201800130005	姓名：张畅	班级： 2018 级计科 3 班
实验题目：进程控制实验		
实验学时：2	实验日期：2020.11.17	

实验目的：

加深对于进程并发执行概念的理解。实践并发进/线程的创建和控制方法。观察和体验进程的动态特性。进一步理解进程生命期期间创建、变换、撤销状态变换的过程。掌握进程控制的方法，了解父子进程间的控制和协作关系。练习 Linux 系统中进/线程创建与控制有关的系统调用的编程和调试技术。

实验过程中遇到和解决的问题：

（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

在实验的过程中，设计程序初期对进程间的函数调用并不熟悉，实验遇到了一定的阻碍，对给出的实例程序进行了反复的调试，了解了父子进程的函数调用和执行方法，按照其执行顺序和逻辑结构进行了程序的设计，取得了较好的结果。

在使用 tester.c 的测试程序进行程序调试时，由于没有阅读 tester.c 的代码，对其工作方式并不了解。通过阅读其每一行的程序，全面掌握了测试程序的工作机制，完成了程序正确性的测试。

实验步骤与内容：

1、了解父子进程的知识基础

PART1:进程的创建相关的程序调用

Fork

fork() 创建子进程并和其子进程并发执行。

函数原型：pid_t fork(void);

Exec

子进程可以通过 exec() 系统调用族装入一个新的执行程序。

函数原型：int execve(const char *path, const char *argv[], const char * envp[]);

Wait

父进程可以使用 wait() 或 waitpid()系统调用等待子进程的结束并负责收集和清理子进程的退出状态。

函数原型：pid_t wait(int *status);

函数原型：pid_t waitpid(pid_t pid,int *status,int option);

Getpid

pid_t getpid(void);

pid_t getppid(void);

PART2:进程控制的相关调用

Kill

kill 能够发送除杀死一个进程 (SIGKILL、SIGTERM、 SIGQUIT) 之外的其他信号,例如键盘中断 (Ctrl+C) 信号 SIGINT,进程暂停 (Ctrl+Z) 信号 SIGTSTP 等等。

函数原型：int kill(pid_t pid, int sig);

Pause

调用 pause 函数会令调用进程的执行挂起直到一个任意信号到来后再继续运行。

函数原型：int pause(void);

pause 挂起调用它的进程直到有任何信号到达。

调用进程但不自定义处理方法,则进行信号的默认处理。只有进程自定义了信号处理方法,捕获并处理了一个信号后,pause 才会返回调进程。\

Sleep

调用 sleep 函数会令调用进程的执行挂起睡眠指定的秒数或一个它可以响应的信号到来后继续执行。

函数原型：unsigned int sleep(unsigned int seconds);

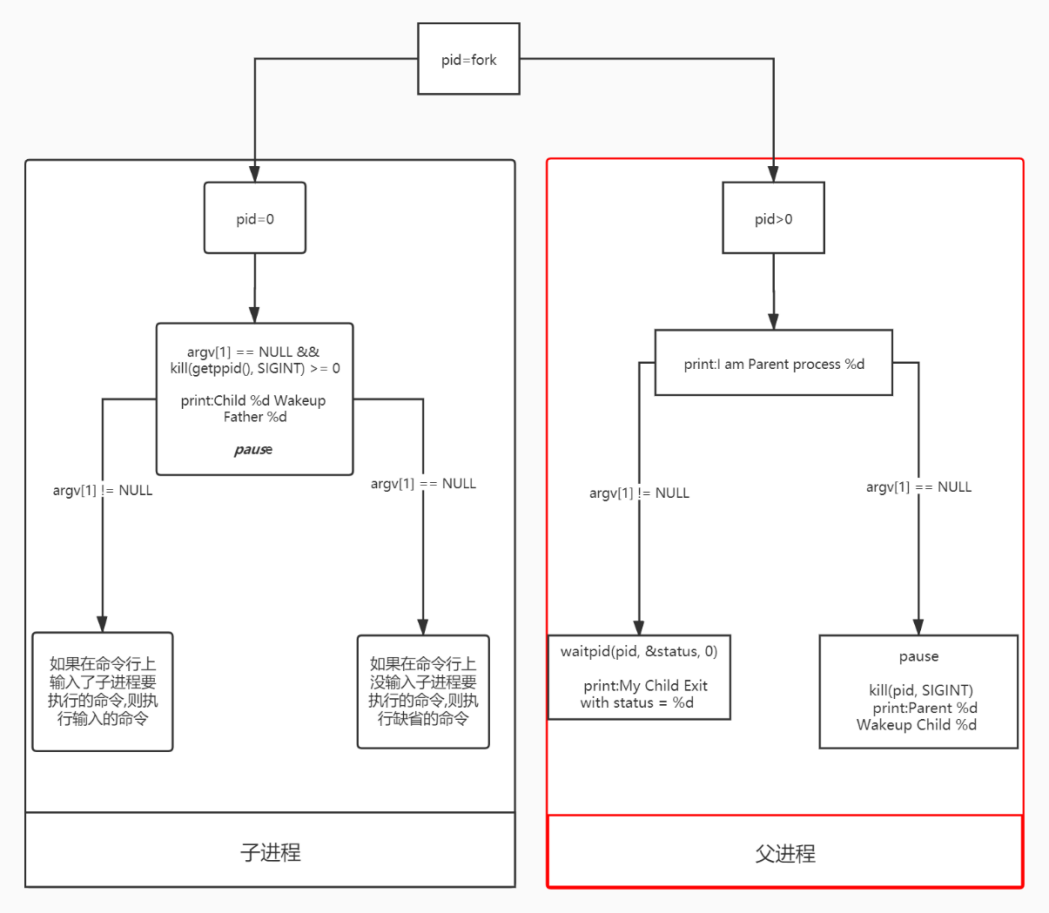
Signal

进程都能使用 signal 函数定义自己的信号处理函数,捕捉并自行处理接收的除 SIGSTOP 和 SIGKILL 之外的信号。

函数原型：sighandler_t signal(int signum, sighandler_t handler);

2、运行给出的示例代码，并学习父子进程的相关函数方法

经过阅读示例代码，可以将代码的逻辑结果整理为如下图所示：



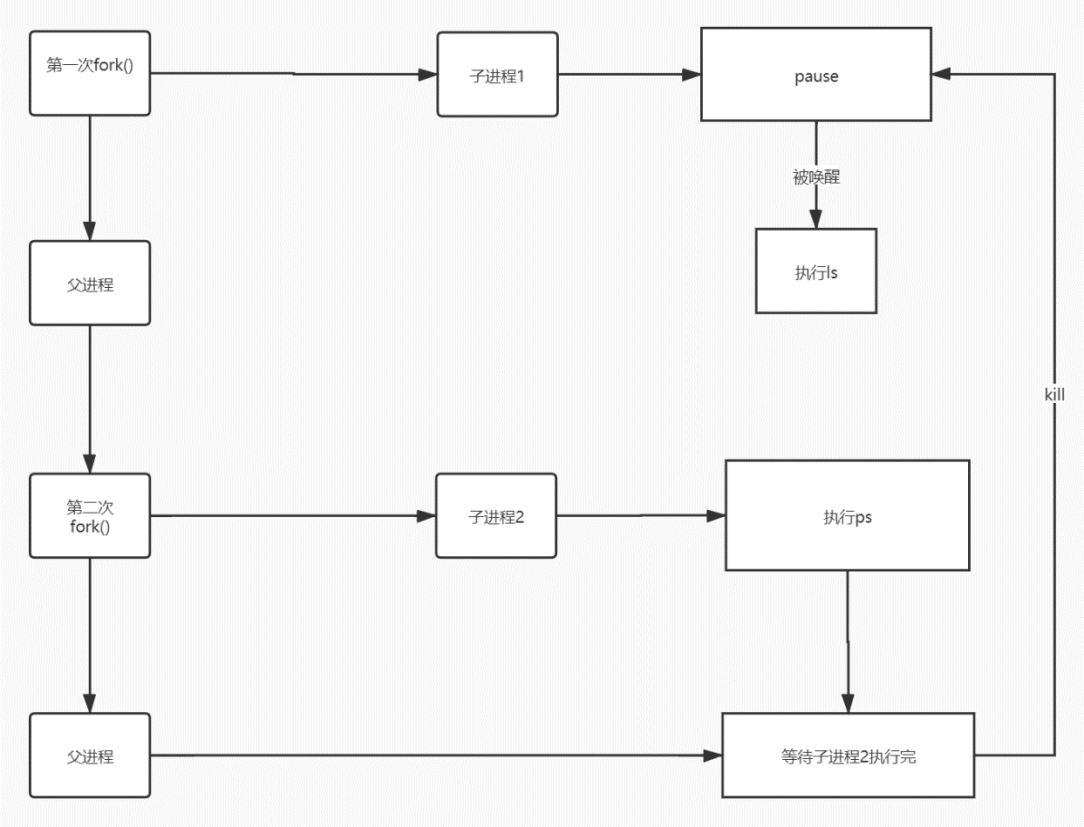
了解了代码的逻辑结果，并且按照代码的逻辑结构对父进程与子进程的样

例程序进行了模拟和分析，至此对父子进程的程序设计基础有了一定的了解，开始设计实验程序。

3、设计程序的逻辑结构

实验的要求是编写一个多进程并发执行程序。父进程每隔 3 秒重复建立两个子进程，首先创建的让其执行 ls 命令，之后创建执行让其执行 ps 命令，并控制 ps 命令总在 ls 命令之前执行。

经过思考，将本实验的程序设计思路使用逻辑图表示如下所示



4、实现程序并测试

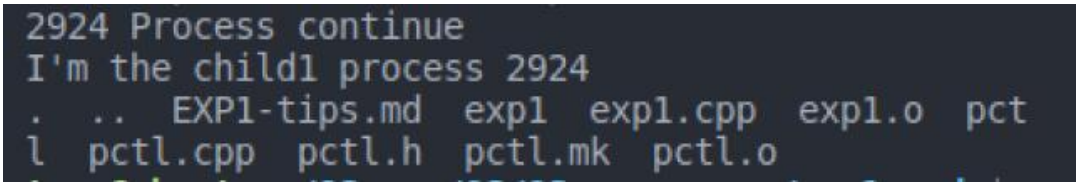
经过对上述的逻辑结构进行模拟和调试，验证了程序设计的正确性和思路的严谨性，进行了程序的实现和测试，测试结果如下所示：

程序的运行结果：

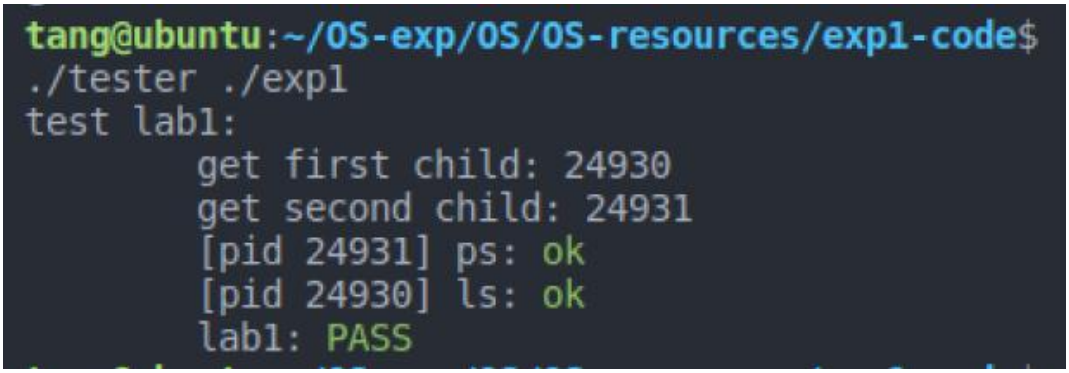
PS 操作结果

```
tang@ubuntu:~/OS-exp/OS/OS-resources/exp1-code$
./exp1
I'm the child2 process 2925
   PID TTY          TIME CMD
   992 tty1     00:00:00 gnome-session-b
  1044 tty1     00:00:09 gnome-shell
  1211 tty1     00:00:01 Xwayland
  1274 tty1     00:00:00 ibus-daemon
  1277 tty1     00:00:00 ibus-dconf
  1280 tty1     00:00:00 ibus-x11
  1304 tty1     00:00:00 gsd-xsettings
  1307 tty1     00:00:00 gsd-a11y-settin
  1308 tty1     00:00:00 gsd-clipboard
  1313 tty1     00:00:00 gsd-color
  1315 tty1     00:00:00 gsd-datetime
```

LS 操作结果



使用 tester.c 测试结果：



实验总结：

经过父子进程的程序设计实验，对进程的调度和并发执行有了更加深刻的认识：

并发的实质是一个处理器在几个进程之间的多路复用，是对有限的物理资源强制行使多用户共享，消除计算机部件之间的互等现象，以提高系统资源利用率。在宏观上，并发性反映一个时间段中几个进程都在同一处理器上处于运行还未运行结束的状态；在微观上，它表现为任一时刻仅有一个进程在处理器上运行。

并发进程之间的关系分为两类：无关的和交互的。无关的并发进程：一个并发进程不会改变另一个并发进程的变量值；交互的并发进程：一组并发进程共享某些变量，进程的执行可能影响其他并发进程的执行结果。

附录：程序源代码

```
#include "pctl.h"

int main(int argc, char *argv[]) {
    int i;
    int pid1, pid2;
    int status1, status2;
```

```
    signal(SIGINT,(sighandler_t)sigcat);
    char *arg1[] = {"/bin/ls","-a", NULL};
    char *arg2[] = {"/bin/ps","-a", NULL};

    for (i = 0; i < 1; i++)
    {
        pid1 = fork();
        if (pid1 < 0)
        {
            printf("Create Process child1 fail!\n");
            exit(EXIT_FAILURE);
        }
        if (pid1 == 0)//child1
        {
            pause();
            printf("I'm the child1 process %d \n", getpid());
              status1 = execve(arg1[0], &arg1[0], NULL);
        }
        else//father
        {
            pid2 = fork();
            if (pid2 < 0)
            {
                printf("Create Process child2 fail!\n");
                exit(EXIT_FAILURE);
            }
            if (pid2 == 0)//child2
            {
                printf("I'm the child2 process %d \n", getpid());
                  status2 = execve(arg2[0], &arg2[0], NULL);
            }
            else//father
              {
                  waitpid(pid2, &status2, 0);
                  if (kill(pid1, SIGINT) >= 0)
                        waitpid(pid1, &status1, 0);
              }
        }
        sleep(3);
    }
    return EXIT_SUCCESS;
}
```