# 操作系统原理实验报告

# 基本信息

- 实验题目：进程控制实验
- 小组编号：无
- 完成人：罗远航
- 学号：201300301165
- 班级：软件工程2013级5班
- 报告日期：2015/04/16

# 实验内容简述

## 实验目标:

> 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。

## 实验要求:

> 根据实验中观察和记录的信息结合示例实验和独立实验程序,说明它们反映出操作系统教材中进程及处理机管理一节讲解的进程的哪些特征和功能?在真实的操作系统中它是怎样实现和反映出教材中讲解的进程的生命期、进程的实体和进程状态控制的。你对于进程概念和并发概念有哪些新的理解和认识?子进程是如何创建和执行新程序的?信号的机理是什么?怎样利用信号实现进程控制?根据实验程序、调试过程和结果分析写出实验报告。

## 实验的软硬件要求:

- 软件要求：Linux操作系统
- 硬件要求：无

# 报告主要内容

## 实验的思路:

> 参考示例程序中建立并发进程的方法,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。

# 实验模型描述

> 因为要重复建立两个子进程，因此可以把其中一个子进程的创建放于if代码段，另一个放于else代码段。使第一个创建的子进程去执行ls命令，第二个子进程去执行ps命令，然后使用waitpid()方法使得第一个创建的子进程等待第二个子进程执行完命令才执行，这样ps命令总是在ls之前执行。至于每三秒就重复，将代码置于while(true)循环中，然后sleep(3)，即可每三秒重复。

# 实验代码：

**1. pctl.h**

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

typedef void (*sighandler_t)(int);
void sigcat()
{
    printf("%d Process continue\n",getpid());
}
```

**2. pctl.c**

```
/*
* pctl.c
* by Yuanhang Luo
* 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ps命令，并控制ps命令总在ls命令之前进行。
*/

#include "pctl.h"

int main(int argc, char *argv[])
{
    int i;
    int pid_1, pid_2;//两个子进程
    int status_ls, status_ps;//两个子进程状态
    char *args_ls[] = {"/bin/ls", "-a", NULL};//ls命令
```

```
    char *args_ps[] = {"/bin/ps", "-l" ,NULL};//ps命令
    signal(SIGINT,(sighandler_t)sigcat);//键盘中断

    while(1){
        if((pid_1=fork())==0){//创建一个子进程
            pause();
            printf("CHILD 1_ps: RUNNING\n");
            printf("CHILD 1_ps: My pid is %d\n",getpid());
            printf("CHILD 1_ps: My father's pid is %d\n",getppid());
            status_ps = execve(args_ps[0],args_ps,NULL);//执行ps命令
        }
        else{
            pid_2 = fork();//创建第二个子进程
            if(pid_2 < 0){
                printf("Create ps Fail!\n");
                exit(EXIT_FAILURE);
            }
            else if(pid_2 == 0){
                pause();
                printf("CHILD 2_ls: RUNNING\n");
                printf("CHILD 2_ls: My pid is %d\n",getpid());
                printf("CHILD 2_ls: My father's pid is %d\n",getppid());
                status_ps = execve(args_ls[0],args_ls, NULL);//执行ls命令
            }
            else{
                printf("FATHER: RUNNING\n");
                printf("FATHER: My pid is: %d", getpid());

                if(kill(pid_1,SIGINT)>=0){
                    waitpid(pid_1, &status_ls,0);//等待执行ls命令
                    sleep(3);//等待3秒
                    printf("FATHER: CHILD ps finished\n");
                    if(kill(pid_2, SIGINT)>=0){
                        waitpid(pid_2, &status_ls,0);
                        printf("FATHER: CHILD ls finished.\n");
                    }
                }
            }
        }
    }
    return EXIT_SUCCESS;
}
```

### 3. Makefile

```
head = pctl.h
srcs = pctl.c
objs = pctl.o
opts = -g -c
all:    pctl
pctl:   $(objs)
    gcc $(objs) -o pctl
pctl.o: $(srcs) $(head)
    gcc $(opts) $(srcs)
clean:
    rm pctl *.o
```

# 实验过程和结果

## 实验过程Bug:

**Makefile文件编写格式错误:**

clean：后应该换行再添加rm命令，否则在执行make clean时会报错。

## 实验结果

**执行make, ./ptcl后结果:**

```
FATHER: RUNNING
6084 Process continue
CHILD 1: RUNNING
CHILD 1: My pid is 6084
CHILD ls1My father's pid is 6083
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  6083 24427  0  80   0 -  1049 wait   pts/2    00:00:00 pctl
0 R  1000  6084  6083  0  80   0 -  1784 -      pts/2    00:00:00 ps
1 S  1000  6085  6083  0  80   0 -  1048 pause  pts/2    00:00:00 pctl
0 S  1000 24427 24420  0  80   0 -  6820 wait   pts/2    00:00:00 bash
FATHER: My pid is: 6083FATHER: CHILD ps finished
6085 Process continue
CHILD ls: RUNNING
CHILD ls: My pid is 6085
CHILD ls: My father's pid is 6083
. .. Makefile  Makefile~  pctl  pctl.c  pctl.c~  pctl.h  pctl.h~  pctl.o
FATHER: CHILD ls finished.
FATHER: RUNNING
6104 Process continue
```

> 持续执行会每3秒重复以上结果

## 实验收获:

**通过实验使我对进程的创建过程有了一个初步的了解，并且学会了如何控制几个子进程之间的执行，另外使我熟悉了C语言，加深了使用Linux命令的熟悉程度。**

# 其他

> 该实验的代码以及文档由Github托管:
> https://github.com/luoyhang003/os_experiments/