# 计算机学院 操作系统 课程实验报告

| 实验题目： 进程控制 | | 学号：202200130048 |
|---|---|---|
| 日期：10.9 | 班级： 6 | 姓名： 陈静雯 |
| Email：1205037094@qq.com | | |

实验步骤与现象：

1. 示例实验

（1）默认命令

```
orange@orange-VirtualBox:~/czsystem$ gcc -g -c pctl.c
orange@orange-VirtualBox:~/czsystem$ gcc pctl.o -o pctl
orange@orange-VirtualBox:~/czsystem$ ./pctl
SIGINT: Success

I am Child process 3195
I am Parent process 3194
My father is 3194
3194 Wakeup 3195 child.
3194 don't Wait for child done.

orange@orange-VirtualBox:~/czsystem$ 3195 Process continue
3195 child will Running:
/bin/ls -a
.   .vscode   exec.c  getpid.c  pctl.c  pctl.o
..  Makefile  fork.c  pctl      pctl.h  wait.c
^C
```

父进程 3194 创建了一个子进程 3195，子进程执行被暂停。

父进程向子进程发出键盘中断信号唤醒子进程并与子进程并发执行。 父进程并没有等待子进程的结束继续执行先行结束了（此时的子进程成为了孤儿进程，不会有父进程为它清理退出状态了）。而子进程继续执行，它变成了列出当前目录所有文件名的命令 ls -a。在完成了列出文件名命令之后，子进程的执行也结束了。此时子进程的退出状态将有初始化进程为它清理。

（2）有指定命令

```
orange@orange-VirtualBox:~/czsystem$ ./pctl /bin/ls -l
SIGINT: Success

I am Parent process 3214
3214 Waiting for child done.

I am Child process 3215
My father is 3214
^Z
[1]+  Stopped                 ./pctl /bin/ls -l
```

子进程仍然被挂起，而父进程则在等待子进程的完成。为了检测父子进程是否都在并发

执行，输入 ctrl+z 将当前进程放入后台并输入 ps 命令查看当前系统进程信息，显示如下:

```
orange@orange-VirtualBox:~/czsystem$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    2766    2527  0  80   0 -  4923 do_wai pts/2    00:00:00 bash
0 T  1000    3214    2766  0  80   0 -   670 do_sig pts/2    00:00:00 pctl
1 T  1000    3215    3214  0  80   0 -   670 do_sig pts/2    00:00:00 pctl
0 R  1000    3219    2766  0  80   0 -  5579 -      pts/2    00:00:00 ps
```

可以看到当前系统中同时有两个叫 pctl 的进程，它们的进程号分别是 3214 和 3215。它们的状态都为一T‖,说明当前都被挂起。3215 的父进程是 3214， 而 3214 的父进程是 2766，也就是 bash-shell。为了让 pctl 父子进程继续执行，输入 fg 命令让 pctl 再次返回前台，现在 pctl 父子进程重新返回前台。通过键盘发键盘中断信号来唤醒 pctl 父子进程继续执行，输入 ctrl+c，将会显示：

```
orange@orange-VirtualBox:~/czsystem$ fg
./pctl /bin/ls -l
^C3214 Process continue
3215 Process continue
3215 child will Running:
/bin/ls -l
total 35
-rwxrwxrwx 1 root root   169 Oct  9 22:59 Makefile
-rwxrwxrwx 1 root root   118 Oct  9 20:03 exec.c
-rwxrwxrwx 1 root root   382 Oct  9 20:03 fork.c
-rwxrwxrwx 1 root root    86 Oct  9 20:05 getpid.c
-rwxrwxrwx 1 root root 18936 Oct  9 23:24 pctl
-rwxrwxrwx 1 root root  2500 Oct  9 22:37 pctl.c
-rwxrwxrwx 1 root root   277 Oct  9 19:59 pctl.h
-rwxrwxrwx 1 root root  8152 Oct  9 23:24 pctl.o
-rwxrwxrwx 1 root root   372 Oct  9 20:04 wait.c

My child exit! status = 0
```

以上输出说明了子进程在捕捉到键盘中断信号后继续执行了指定的命令，按我们要求的长格式列出了当前目录中的文件名，父进程在接收到子进程执行结束的信号后将清理子进程的退出状态并继续执行，它报告了子进程的退出编码（0 表示子进程正常结束)最后父进程也结束执行。

2. **独立实验**

**（1）实验结果**

```
orange@orange-VirtualBox:~/czsystem$ gcc -g -c test.c
orange@orange-VirtualBox:~/czsystem$ gcc test.o -o test
orange@orange-VirtualBox:~/czsystem$ ./test
SIGINT: Success
I am Child process1 3945. My father is 3944
waiting child2.
I am Child process2 3946. My father is 3944
/bin/ps -a
    PID TTY          TIME CMD
   1708 tty2     00:00:00 gnome-session-b
   3944 pts/0    00:00:00 test
   3945 pts/0    00:00:00 test
   3946 pts/0    00:00:00 ps
^C3945 Process continue
child2 done, 3945 child1 will Running: /bin/ls -a
3944 Process continue
.   .vscode   exec.c  getpid.c  pctl.c  pctl.o  test.c  test.o
..  Makefile  fork.c  pctl      pctl.h  test    test.h  wait.c
I am Child process1 3947. My father is 3944
waiting child2.
I am Child process2 3948. My father is 3944
/bin/ps -a
    PID TTY          TIME CMD
   1708 tty2     00:00:00 gnome-session-b
   3944 pts/0    00:00:00 test
   3947 pts/0    00:00:00 test
   3948 pts/0    00:00:00 ps
^C3947 Process continue
child2 done, 3947 child1 will Running: /bin/ls -a
3944 Process continue
```

```
.   .vscode   exec.c  getpid.c  pctl.c  pctl.o  test.c  test.o
..  Makefile  fork.c  pctl      pctl.h  test    test.h  wait.c
I am Child process1 3949. My father is 3944
waiting child2.
I am Child process2 3950. My father is 3944
/bin/ps -a
    PID TTY          TIME CMD
   1708 tty2     00:00:00 gnome-session-b
   3944 pts/0    00:00:00 test
   3949 pts/0    00:00:00 test
   3950 pts/0    00:00:00 ps
```

一开始父进程 3944 创建两个子进程 1 和 2，子进程 1 暂停等待子进程 2 完成 ps 操作，直到接收到键盘信号 ctrl+c 才继续执行 ls 操作，父进程等子进程 12 都完成后，睡眠三秒，进入下一次循环，即重复执行上面的过程。

结论分析：

1. **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？**

（1） 进程创建：fork（），子进程是父进程的一个副本，它继承了父进程的大部分属性，但子进程有自己的独立进程 ID（PID）和父进程 ID（PPID），子进程会从被创建的下一行代码开始执行

```
pid=fork();//建立子进程
if(pid<0)//建立子进程失败？
{
```

```
pid1=fork();//建立子进程1
if(pid1>0) pid2=fork();//建立子进程2
```

（2） 进程控制块：操作系统使用 PCB 来管理和调度进程。其中包含进程的所有必要信息，如进程状态、进程标识符、内存管理信息、处理器状态等。在 fork() 调用后，子进程的 PCB 会被初始化，以确保子进程可以独立运行。

（3） 进程等待：waitpid（），父进程可以使用 waitpid()来等待子进程结束，这有助于避免僵尸进程的产生。

```
    printf("%d Waiting for child done.\n\n",getpid())
    waitpid(pid,&status,0); //等待子进程结束
    printf("\nMy child exit! status = %d\n\n",status)
}
```

```
    waitpid(pid2,&status2,0); //等待子进程2结束
    waitpid(pid1,&status1,0); //等待子进程1结束
    sleep(3);
```

（4） 进程执行新程序：execve()，系统调用用于替换当前进程的内存空间，加载并执行新的程序。

```
for(i=0; args[i] != NULL; i++)
    printf("%s ",args[i]); printf("\n");
//装入并执行新的程序
status1 = execve(args[0],args,NULL);
```

（5） 进程调度：通过 fork()创建的多个进程会被操作系统调度器管理，确保它们公平地获得 CPU 时间。

（6） 进程标识符：PID，每个进程都有一个唯一的进程标识符。

2. **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、进程的实体和进程控制的。**

（1） 生命期：从进程被 fork 开始，直到 exit 正常终止或异常终止为止

```
signal(SIGINT,&sigcat);//注册一个本进程处理键盘中断的函数
perror("SIGINT");//如果系统调用signal成功执行，输出“SIGINT”，否则，
                 //输出“SIGINT”及出错原因

while(1){
  pid1=fork();//建立子进程1
```

```
  }
  return EXIT_SUCCESS;
}
```

```
                 //输出“SIGINT”及出错原因
pid=fork();//建立子进程
if(pid<0)//建立子进程失败?
{
  printf("Create Process fail!\n");
  exit(EXIT_FAILURE);
```

```
  sleep(5); //思考：如果去掉这条语句，可能会出现什么现象
  if(kill(pid,SIGINT) >= 0)
     printf("%d Wakeup %d child.\n",getpid(),pid) ;
  printf("%d don't Wait for child done.\n\n",getpid());
}

eturn EXIT_SUCCESS;
```

```
My child exit! status = 0
```

（2）进程的实体：通过进程控制块体现，PCB 是操作系统用于管理进程的重要数据结构，它包含了进程的所有必要信息，如进程状态、进程标识符（PID）、内存管理信息、处理器状态、文件描述符表等。每个进程都有一个唯一的 PCB，操作系统通过 PCB 来管理和调度进程。

（3）进程控制：通过进程创建、进程调度、进程终止等体现

```
printf(" I am child process %d\nMy father is %d\n",g
pause();//暂停，等待键盘中断信号唤醒
//子进程被键盘中断信号唤醒继续执行
printf("%d child will Running: \n",getpid());
```

```
    sleep(5); //思考：如果去掉这条语句，可能会出现什么现象
    if(kill(pid,SIGINT) >= 0)
        printf("%d Wakeup %d child.\n",getpid(),pid) ;
    printf("%d don't Wait for child done.\n\n",getpid());
```

```
I am Child process 3195
I am Parent process 3194
My father is 3194
3194 Wakeup 3195 child.
3194 don't Wait for child done.

orange@orange-VirtualBox:~/czsystem$ 3195 Process cont
3195 child will Running:
/bin/ls -a
.   .vscode   exec.c  getpid.c  pctl.c  pctl.o
..  Makefile  fork.c  pctl      pctl.h  wait.c
```

3. **对于进程概念和并发概念有哪些新的理解和认识？**

（1） 进程：每个进程都有自己独立的地址空间、文件描述符表、环境变量等资源，这使得进程之间互不影响，提高了系统的稳定性和安全性。每个进程都有自己的进程控制块（PCB），记录了进程的状态信息，如进程标识符（PID）、父进程标识符（PPID）、内存管理信息等。

（2） 并发：多个任务在同一时间段内交错执行，但不一定同时执行。操作系统通过快速切换 CPU 上的任务来实现并发。并发可以提高资源利用率和系统响应性，但也带来了同步和调试的挑战。

4. **子进程是如何创建和执行新程序的？**

（1） 创建：fork（）

```
                                    //输出 SIGINT 发出
    pid=fork();//建立子进程
    if(pid<0)//建立子进程失败?
    {
        printf("Create Process fail!\n");
```

（2） 执行：exec（）

```
    for(i=0; args[i] != NULL; i++)
        printf("%s ",args[i]); printf("\n");
    //装入并执行新的程序
    status1 = execve(args[0],args,NULL);
```

5. **信号的机理是什么？**

（1）信号生成：用户按 Ctrl+C 发送 SIGINT 信号。内核接收到信号，将其添加到目标

进程的信号集合中。

（2）信号传递：内核在适当的时候检查进程的信号集合。如果进程当前没有被阻塞且信号未被屏蔽，则内核将信号传递给进程。

（3）信号处理：

立即处理：如果进程当前正在运行，内核会立即中断当前的执行，调用信号处理函数。

延迟处理：如果进程当前不在运行状态（如被阻塞或等待 I/O），内核会在进程恢复运行时调用信号处理函数。

（3） 信号处理函数：信号处理函数执行完毕后，内核恢复进程的正常执行。如果有多个信号等待处理，内核会依次调用相应的信号处理函数

**6. 怎样利用信号实现进程控制？**

通过发送特定的信号，可以控制目标进程的行为，如终止进程、暂停进程、恢复进程等。使用 kill() 系统调用发送信号，通过 signal() 或 sigaction() 注册信号处理函数，可以在进程接收到特定信号时执行自定义的操作。

```
signal(SIGINT,&sigcat);//注册一个本进程处理键盘中断的函数
perror("SIGINT");//如果系统调用signal成功执行，输出“SIGINT”，否则，
                 //输出“SIGINT”及出错原因
```

```
 sleep(5); //思考：如果去掉这条语句，可
if(kill(pid,SIGINT) >= 0)
    printf("%d Wakeup %d child.\n",get
```