

《计算机操作系统》实验报告

实验题目：进程管理及进程通信

姓名：严昕宇 学号：20121802 实验日期：2022.10.30

实验环境：

实验设备：Lenovo Thinkbook16+ 2022

操作系统：Ubuntu 22.04.1 LTS 64 位

实验目的：

1. 利用 Linux 提供的系统调用设计程序，加深对进程概念的理解
2. 体会系统进程调度的方法和效果
3. 了解进程之间的通信方式以及各种通信方式的使用

实验内容：

用 vi 编写使用系统调用的 C 语言程序

操作过程 1：

编写程序。显示进程的有关标识（进程标识、组标识、用户标识等）。经过 5 秒钟后，执行另一个程序，最后按用户指示（如：Y/N）结束操作。

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

int main() {
    int i, j;
    char ch = 'N';
    while (ch != 'y' && ch != 'Y') {
        printf("进程标识:%d\n", getpid());
        printf("组标识:%d\n", getpgrp());
        printf("用户标识:%d\n", getuid());
        i = fork();
        if (i == 0) {
            sleep(5);
            execlp("ls", "ls", (char *) 0);
            perror("exec error.\n");
            exit(0);
        } else if (i > 0) {
            j = wait(0);
```

```

        printf("是否退出 (Y/N)?\n");
        scanf("%c", &ch);
        getchar();
    } else
        printf("创建失败\n");
}
exit(0);
return 0;
}

```

结果 1:

```

yanxinyu@Thinkbook16-2022:~$ ./prog3.1.o
进程标识:5123
组标识:5123
用户标识:1000

```

操作过程 2:

参考例程 1，编写程序。实现父进程创建一个子进程。体会子进程与父进程分别获得不同返回值，进而执行不同的程序段的方法。

```

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

int main() {
    int i;
    if (fork()) {
        i = wait(0);
        printf("It is parent process.\n");
        printf("The child process,ID number %d,is finished.\n", i);
    } else {
        printf("It is child process.\n");
        sleep(10);
        exit(0);
    }
    return 0;
}

```

结果 2:

```
yanxinyu@Thinkbook16-2022:~$ ./prog3.2.o
It is child process.
It is parent process.
The child process,ID number 5451,is finished.
```

思考：子进程是如何产生的？ 又是如何结束的？子进程被创建后它的运行环境是怎样建立的？

答：子进程通过父进程调用 `fork()` 函数创建一个子进程，子进程会继承父进程剩下的程序内容，当执行到 `exit()` 时结束进程。

在 Linux 中，每个进程都用一个唯一的整型进程标识符来标识。通过系统调用 `fork()`，可创建新进程。新进程的地址空间复制了原来进程的地址空间。这种机制允许父进程与子进程轻松通信。这两个进程(父和子)都继续执行处于系统调用 `fork()` 之后的指令，但有一点不同：对于新(子)进程，系统调用 `fork()` 的返回值为 0；而对于父进程，返回值为子进程的进程标识符(非零)。子进程被创建后核心将为其分配一个进程表项和进程标识符，检查同时运行的进程数目，并且拷贝进程表项的数据，由于子进程继承父进程的所有文件。

通常，在系统调用 `fork()` 之后，有个进程使用系统调用 `exec()` 以用新程序来取代进程的内存空间。系统调用 `exec()` 加载二进制文件到内存中，并开始执行。父进程能够创建更多子进程，或者如果在子进程运行时没有什么可做，那么它采用系统调用 `wait()` 把自己移出就绪队列，直到子进程终止。

操作过程 3

参考例程 2，编写程序。父进程通过循环语句创建若干子进程。探讨进程的家族树以及子进程继承父进程的资源的关系。

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

int main() {
    int i;
    int j;
    printf("My pid is %d,my father's pid is %d\n", getpid(),
getppid());
    for (i = 0; i < 3; i++) {
        if (fork() == 0)
            printf("%dpid=%d ppid=%d\n", i, getpid(), getppid());
        else {
            j = wait(0);
            printf("%d :The child %d is finished.\n", getpid(), j);
        }
    }
}
```

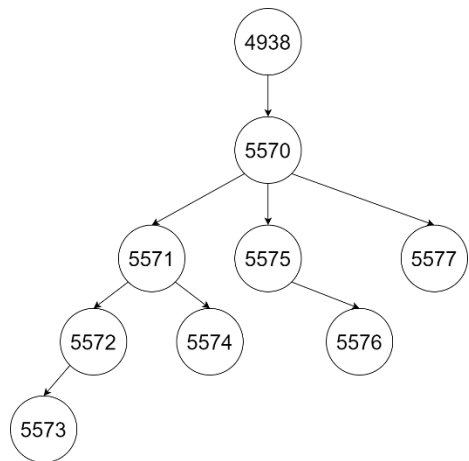
```
}  
    return 0;  
}
```

结果 3

```
yanxinyu@Thinkbook16-2022:~$ ./prog3.3.o  
My pid is 5570,my father's pid is 4938  
0pid=5571 ppid=5570  
1pid=5572 ppid=5571  
2pid=5573 ppid=5572  
5572 :The child 5573 is finished.  
5571 :The child 5572 is finished.  
2pid=5574 ppid=5571  
5571 :The child 5574 is finished.  
5570 :The child 5571 is finished.  
1pid=5575 ppid=5570  
2pid=5576 ppid=5575  
5575 :The child 5576 is finished.  
5570 :The child 5575 is finished.  
2pid=5577 ppid=5570  
5570 :The child 5577 is finished.
```

思考：① 画出进程的家族树。子进程的运行环境是怎样建立的？反复运行此程序看会有什么情况？解释一下。

答：



每一次运行返回的进程号都不相同，但是家族树的树形都是相同的，出现这样的情况是由于进程号是随机分配的。

② 修改程序，使运行结果呈单分支结构，即每个父进程只产生一个子进程。画出进程树，解释该程序。

答：

```

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

int main() {
    int i;
    int j;
    printf("My pid is %d,my father's pid is %d\n", getpid(),
getppid());
    for (i = 0; i < 3; i++) {
        if (fork() == 0)
            printf("%dpid=%d ppid=%d\n", i, getpid(), getppid());
        else {
            j = wait(0);
            printf("%d :The child %d is finished.\n", getpid(), j);
            break;
        }
    }
    return 0;
}

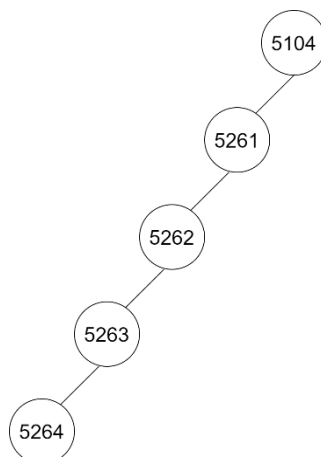
```

```

yanxinyu@Thinkbook16-2022:~$ ./prog3.4.o
My pid is 5261,my father's pid is 5104
0pid=5262 ppid=5261
1pid=5263 ppid=5262
2pid=5264 ppid=5263
5263 :The child 5264 is finished.
5262 :The child 5263 is finished.
5261 :The child 5262 is finished.

```

进程家族树如下图所示：



操作过程 4

参考例程 3 编程，使用 `fork()` 和 `exec()` 等系统调用创建三个子进程。子进程分别启动不同程序，并结束。反复执行该程序，观察运行结果，结束的先后，看是否有不同次序。

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

int main() {
    int child_pid1, child_pid2, child_pid3;
    int pid, status;
    setbuf(stdout, NULL);
    child_pid1 = fork();
    if (child_pid1 == 0) {
        execlp("echo", "echo", "child process 1", (char *) 0);
        perror("exec1 error.\n");
        exit(1);
    }
    child_pid2 = fork();
    if (child_pid2 == 0) {
        execlp("date", "date", (char *) 0);
        perror("exec2 error.\n");
        exit(2);
    }
    child_pid3 = fork();
    if (child_pid3 == 0) {
        execlp("ls", "ls", (char *) 0);
        perror("exec3 error.\n");
        exit(3);
    }
    puts("Parent process is waiting for child process return!");
    while ((pid = wait(&status)) != -1) {
        if (child_pid1 == pid)
            printf("child process 1 terminated with status %d\n",
(status >> 8));
        else {
            if (child_pid2 == pid)
                printf("child process 2 terminated with
status %d\n", (status >> 8));
            else {
                if (child_pid3 == pid)
                    /*若子进程 3
结束*/
```

```

        printf("child process 3 terminated with
status %d\n", (status >> 8));
    }
}
}
puts("All child processes terminated.");
puts("Parent process terminated.");
exit(0);
}

```

结果 4

```

yanxinyu@Thinkbook16-2022:~$ ./prog4.o
Parent process is waiting for child process return!
child process 1
2022年 11月 12日 星期六 12:27:36 CST
child process 1 terminated with status 0
child process 2 terminated with status 0
公共的  图片  音乐  f      filedir.txt  prog3.1.o  prog3.3.c  prog3.4.o  snap
模板   文档  桌面  f5     prog1.sh   prog3.2.c  prog3.3.o  prog4.c
视频   下载  cppdir  filea  prog3.1.c  prog3.2.o  prog3.4.c  prog4.o
child process 3 terminated with status 0
All child processes terminated.
Parent process terminated.
yanxinyu@Thinkbook16-2022:~$ ./prog4.o
Parent process is waiting for child process return!
child process 1
child process 1 terminated with status 0
2022年 11月 12日 星期六 12:28:06 CST
child process 2 terminated with status 0
公共的  图片  音乐  f      filedir.txt  prog3.1.o  prog3.3.c  prog3.4.o  snap
模板   文档  桌面  f5     prog1.sh   prog3.2.c  prog3.3.o  prog4.c
视频   下载  cppdir  filea  prog3.1.c  prog3.2.o  prog3.4.c  prog4.o
child process 3 terminated with status 0
All child processes terminated.
Parent process terminated.

```

思考：子进程运行其它程序后，进程运行环境怎样变化的？反复运行此程序看会有什么情况？解释一下。

答：子进程运行其他程序后，这个进程就完全被新程序代替。由于并没有产生新进程所以进程标识号不改变，除此之外的旧进程的信息、代码段、数据段、栈段等都被新程序所代替。新程序从自己的 `main()` 函数开始进行。分析可知，子进程在调用 `execlp` 后，如果正常调用，则该子进程会被 `execlp` 调用的可执行文件取代，即和 `fork()` 创建子进程不同，即使在 `execlp` 调用的可执行文件执行结束后，子进程也不会继续往下执行程序了，而是处于已被销毁的状态，同时调用的可执行文件会继承原子进程的进程号。

反复运行此程序可以发现，输出的结果顺序是会发生改变的，结束的先后次序是不可预知的。原因是当每个子进程运行其他程序时，他们的结束随着其他程序的结束而结束，所以结束的先后次序在改变。

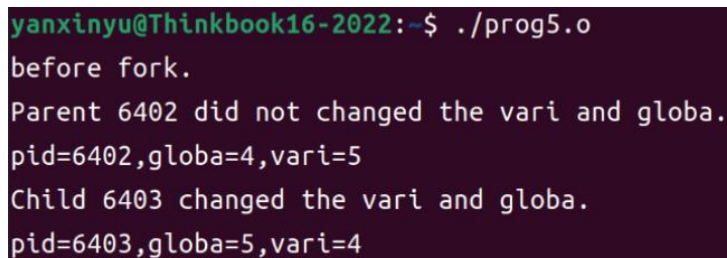
操作过程 5

参考例程 4 编程，验证子进程继承父进程的程序、数据等资源。如用父、子进程修改公共变量和私有变量的处理结果；父、子进程的程序区和数据区的位置。

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

int globa = 4;
int main() {
    pid_t pid;
    int vari = 5;
    printf("before fork.\n");
    if ((pid = fork()) < 0) {
        printf("fork error.\n");
        exit(0);
    } else if (pid == 0) {
        globa++;
        vari--;
        printf("Child %d changed the vari and globa.\n",
getpid());
    } else
        printf("Parent %d did not changed the vari and globa.\n",
getpid());
    printf("pid=%d,globa=%d,vari=%d\n", getpid(), globa, vari);
    exit(0);
}
```

结果 5



```
yanxinyu@Thinkbook16-2022:~$ ./prog5.o
before fork.
Parent 6402 did not changed the vari and globa.
pid=6402,globa=4,vari=5
Child 6403 changed the vari and globa.
pid=6403,globa=5,vari=4
```

思考：子进程被创建后，对父进程的运行环境有影响吗？解释一下。

答：无影响，这是因为子进程继承的父进程资源是以拷贝形式继承的，如果在子进程运行过程中无变动，则二者共用同一个地址的资源，若子进程或父进程在创建子进程后对某个资源产生了改动，则操作系统会拷贝一份资源作为子进程单独占用的内容，然后父进程/子进程再对自己独有的资源进行改动。

操作过程 6

参照《实验指导》第五部分中“管道操作的系统调用”。复习管道通信概念，参考例程 5，编写一个程序。父进程创建两个子进程，父子进程之间利用管道进行通信。要求能显示父进程、子进程各自的信息，体现通信效果。

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

int main() {
    int i, r, j, k, l, p1, p2, fd[2];
    char buf[50], s[50];
    pipe(fd);
    while ((p1 = fork()) == -1);
    if (p1 == 0) {
        lockf(fd[1], 1, 0);
        sprintf(buf, "Child process P1 is sending messages!\n");
        printf("Child process P1!\n");
        write(fd[1], buf, 50);
        lockf(fd[1], 0, 0);
        sleep(1);
        j = getpid();
        k = getppid();
        printf("P1 %d is wakeup. My parent process ID is %d.\n",
j, k);
        exit(0);
    } else {
        while ((p2 = fork()) == -1);
        if (p2 == 0) {
            lockf(fd[1], 1, 0);
            sprintf(buf, "Child process P2 is sending
messages!\n");
            printf("Child process P2!\n");
            write(fd[1], buf, 50);
            lockf(fd[1], 0, 0);
            sleep(1);
            j = getpid();
            k = getppid();
            printf("P2 %d is wakeup. My parent process ID
is %d.\n", j, k);
            exit(0);
        } else {
            l = getpid();
```

```

        wait(0);
        if (r = read(fd[0], s, 50) == -1)
            printf("Can't read pipe.\n");
        else
            printf("Parent %d: %s\n", l, s);
        wait(0);
        if (r = read(fd[0], s, 50) == -1)
            printf("Can't read pipe.\n");
        else
            printf("Parent %d: %s\n", l, s);
        exit(0);
    }
}

```

结果 6

```

yanxinyu@Thinkbook16-2022:~$ ./prog6.o
Child process P1!
Child process P2!
P1 6479 is wakeup. My parent process ID is 6478.
Parent 6478: Child process P1 is sending messages!

P2 6480 is wakeup. My parent process ID is 6478.
Parent 6478: Child process P2 is sending messages!

```

思考：① 什么是管道？进程如何利用它进行通信的？解释一下实现方法。

② 修改睡眠时机、睡眠长度，看看会有什么变化。请解释。

③ 加锁、解锁起什么作用？不用它行吗？

答：① 由于进程只是对一个程序的一次执行，有的任务需要多个程序协调完成，自然进程有需要协调进行的时候，协调工作自然需要信息上的通信，管道就是不同进程通信的邮差。管道是指能够连接一个写进程和一个读进程，并允许他们以生产者-消费者方式进行通信的一个共享文件，又称 `pipe` 文件。由写进程从管道的入端将数据写入管道，而读进程则从管道出端读出数据来进行通信。管道是指从一个进程到另一个进程的数据流，其特点是管道是半双工的，只能向一个方向流动，同时，它只能用于父子进程或兄弟进程间进行通信。

拿父子进程间的通信举例，在父进程创建管道的时候，会获取管道的读写端文件描述符，在创建子进程的时候子进程也会获取到管道的读写端文件描述符，此时就成立了一个半双工单向传输管道，写端将数据写入到操作系统设定的位于内存的管道缓冲区中，读端再从中读出来，这就是管道的实现方法。

② 修改睡眠时机和睡眠长度都会引起进程被唤醒的时间不一，因为在前面的时候两个进程已经分别将通信内容塞入了管道中，根据其先进先出的特性，父进程 `read` 到的内容顺序在 `sleep` 之前已经定好了，但是改变睡眠时机是有影响的，如果我们在某个进程加锁之前就 `sleep` 就保证了两个子进程谁先加锁，固定了向管道内传输数据的顺序。

③ 一是安全性考虑，保证管道在任意时间里只有一个进程对其写入。二是保证写入顺序性。`lockf(fd[1],1,0)`为加锁，`lockf(fd[1],0,0)`为解锁。加锁、解锁是为了解决临界资源的共享问题。不用它将会引起无法有效管理数据，即数据会被修改导致读错了数据。再例如有两个子进程，要保证 A 进程先写入，但是在写入前它要干别的事情，可以在 A 进程一开始就直接锁住，让 B 进程等到 A 写完解锁才能写入。

单从本实验的角度来说，即使去掉锁和解锁在大部分情况下也不会影响管道的传入，因为两个进程是并行运行的，而且都是在开头进行了相同的传入管道操作，不会影响管道通信顺序，但是如上述保证 A 进程先写入的状况是不行的。总结来说，为了安全性和规范性考虑，必须在管道通信前加锁，通信后解锁。

操作过程 7

编程验证：实现父子进程通过管道进行通信。进一步编程，验证子进程结束，由父进程执行撤消进程的操作。测试父进程先于子进程结束时，系统如何处理“孤儿进程”的。

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

int main() {
    int i, r, j, k, l, p1, p2, fd[2];
    char buf[50], s[50];
    pipe(fd);
    while ((p1 = fork()) == -1);
    if (p1 == 0) {
        lockf(fd[1], 1, 0);
        sprintf(buf, "Child process P1 is sending messages!\n");
        printf("Child process P1!\n");
        write(fd[1], buf, 50);
        lockf(fd[1], 0, 0);
        sleep(1);
        j = getpid();
        k = getppid();
        printf("P1 %d is weakup. My parent process ID is %d.\n",
j, k);
        exit(0);
    } else {
        while ((p2 = fork()) == -1);
        if (p2 == 0) {
            lockf(fd[1], 1, 0);
            sprintf(buf, "Child process P2 is sending
messages!\n");
            printf("Child process P2!\n");
```

```

        write(fd[1], buf, 50);
        lockf(fd[1], 0, 0);
        sleep(1);
        j = getpid();
        k = getppid();
        printf("P2 %d is wakeup. My parent process ID
is %d.\n", j, k);
        exit(0);
    } else {
        l = getpid();
        if (r = read(fd[0], s, 50) == -1)
            printf("Can't read pipe.\n");
        else
            printf("Parent %d: %s\n", l, s);
        if (r = read(fd[0], s, 50) == -1)
            printf("Can't read pipe.\n");
        else
            printf("Parent %d: %s\n", l, s);
        exit(0);
    }
}
}

```

结果 7

```

yanxinyu@Thinkbook16-2022:~$ ./prog7.o
Child process P1!
Parent 7015: Child process P1 is sending messages!

Child process P2!
Parent 7015: Child process P2 is sending messages!

yanxinyu@Thinkbook16-2022:~$ P1 7016 is wakeup. My parent process ID is 1607.
P2 7017 is wakeup. My parent process ID is 1607.

```

思考：对此作何感想，自己动手试一试？解释一下你的实现方法。

答：只要在父进程后加上 wait() 函数，然后打印“子进程已经结束”，一旦子进程结束，父进程撤销进程。把父进程中的 wait() 去掉，父进程先于子进程终止时，所有子进程会被领养，在字符界面中由 pid 为 1 的 init 进程领养，本实验在图形界面中，由 pid 为 1607 的 systemd 进程领养。

操作过程 8

编写两个程序一个是服务者程序，一个是客户程序。执行两个进程之间通过消息机制通信。消息标识 MSGKEY 可用常量定义，以便双方都可以利用。客户将自己的进程标识(pid)通过消息机制发送给服务者进程。服务者进程收到消息后，将自己的进程号和父进程号发送给客户，然后返回。客户收到后显示服务者的 pid 和 ppid，结束。以下例程 6 基本实现以上功能。这部分内容涉及《实验指导》第五部分中“IPC 系统调用”。先熟悉一下，再调试程序。

服务者程序：

```
/*The server receives the message from client,and answer a
message*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MSGKEY 75

struct msgform /*定义消息结构*/
{
    long mtype;
    char mtext[256];
} msg;

int msgqid;

int main() {
    int i, pid, *pint;
    msgqid = msgget(MSGKEY, 0777 | IPC_CREAT); /*建立消息队列*/
    for (;;) /*等待接受消息*/
    {
        printf("===== Before Receive=====\n");
        system("ipcs -q");
        msgrcv(msgqid, &msg, 256, 1, 0); /* 接受消息*/
        printf("===== After Receive=====\n");
        system("ipcs -q");
        pint = (int *) msg.mtext;
        pid = *pint;
        printf("Server :Receive From PID %d\n", pid);
        msg.mtype = pid; /*显示消息来源*/
        *pint = getpid(); /*加入自己的进程标识*/
        msgsnd(msgqid, &msg, sizeof(int), 0); /*发送消息*/
    }
}
```

```
}  
    return 0;  
}
```

客户程序:

```
#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/msg.h>  
  
#define MSGKEY 75  
  
struct msgform {                /*定义消息结构*/  
    long mtype;  
    char mtext[256];  
};  
  
int main() {  
    struct msgform msg;  
    int msgqid, pid, *pint;  
    msgqid = msgget(MSGKEY, 0777);  
    pid = getpid();  
    pint = (int *) msg.mtext;  
    *pint = pid;  
    msg.mtype = 1;  
    msgsnd(msgqid, &msg, sizeof(int), 0);  
    system("ipcs -q");  
    msgrcv(msgqid, &msg, 256, pid, 0);  
    printf("Client : Receive From Pid %d\n", *pint);  
    return 0;  
}
```

结果 8

```
yanxinyu@Thinkbook16-2022:~$ ./prog8.o
===== Before Receive=====

----- 消息队列 -----
键      msqid      拥有者  权限      已用字节数  消息
0x0000004b 0      yanxinyu  777      0          0

===== After Receive=====

----- 消息队列 -----
键      msqid      拥有者  权限      已用字节数  消息
0x0000004b 0      yanxinyu  777      0          0

Server :Receive From PID 11913
===== Before Receive=====

----- 消息队列 -----
键      msqid      拥有者  权限      已用字节数  消息
0x0000004b 0      yanxinyu  777      0          0

yanxinyu@Thinkbook16-2022:~$ ./prog8a.o

----- 消息队列 -----
键      msqid      拥有者  权限      已用字节数  消息
0x0000004b 0      yanxinyu  777      0          0

Client : Receive From Pid 11804
yanxinyu@Thinkbook16-2022:~$
```

思考：想一下服务者程序和客户程序的通信还有什么方法可以实现？解释一下你的设想，有兴趣试一试吗。

答：可以尝试使用信号量的方法进行通信。还可以用信号量机制来实现。信号量是一个整形计数器，用来控制多个进程对共享资源的访问。或者通过消息队列信号机制，通过向消息队列发送信息、接收信息来实现进程间的通信。它不是用于交换大批数据,而用于多线程之间的同步。它常作为一种锁机制，防止某进程在访问资源时其它进程也访问该资源。

同时，也可以尝试使用在计算机网络中学习到的 socket 套接字，建立 TCP 客户端和服务端，借助网络进行通信。

操作过程 9

这部分内容涉及《实验指导》第五部分中“有关信号处理的系统调用”。编程实现软中断信号通信。父进程设定软中断信号处理程序，向子进程发软中断信号。子进程收到信号后执行相应处理程序。

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

int main() {
    int i, j, k;
    void func();
    signal(18, &func); /*设置 18 号信号的处理程序*/
    if (i = fork()) {
        j = kill(i, 18);
        /*向子进程发送信号*/
        printf("Parent: signal 18 has been sent to
child %d,returned %d.\n", i, j);
        k = wait(0); /*父进程被唤醒*/
        printf("After wait %d,Parent %d: finished.\n", k,
getpid());
    } else {
        sleep(10);
        printf("Child %d: A signal from my parent is recived.\n",
getpid());
    } /*子进程结束，向父进程发子进程结束信号*/
}

void func() /*处理程序*/
{
    int m;
    m = getpid();
    printf("I am Process %d: It is signal 18 processing
function.\n", m);
}
```


结果 9

```
yanxinyu@Thinkbook16-2022:~$ ./prog9.o
Parent: signal 18 has been sent to child 12351,returned 0.
I am Process 12351: It is signal 18 processing function.
Child 12351: A signal from my parent is recived.
After wait 12351,Parent 12350: finished.
yanxinyu@Thinkbook16-2022:~$
```

思考：这就是软中断信号处理，有点儿明白了吧？讨论一下它与硬中断有什么区别？看来还挺管用，好好利用它。

答：软中断和硬中断最大的区别是，软中断是操作系统执行中断指令产生的，而硬中断则是硬件层次上由于外部事件引发的中断。

硬中断：

- 硬中断是由硬件产生的，比如，像磁盘，网卡，键盘，时钟等。每个设备或设备集都有它自己的 IRQ(中断请求)。基于 IRQ，CPU 可以将相应的请求分发到对应的硬件驱动上(注:硬件驱动通常是内核中的一个子程序，而不是一个独立的进程)。
- 处理中断的驱动是需要运行在 CPU 上的，因此，当中断产生的时候，CPU 会中断当前正在运行的任务，来处理中断。在有多核心的系统上，一个中断通常只能中断一颗 CPU(也有一种特殊的情况,就是在大型主机.上是有硬件通道的，它可以在没有主 CPU 的支持下，可以同时处理多个中断)。
- 硬中断可以直接中断 CPU。它会引起内核中相关的代码被触发。对于那些需要花费一些时间去处理的进程，中断代码本身也可以被其他的硬中断中断。
- 对于时钟中断，内核调度代码会将当前正在运行的进程挂起，从而让其他的进程来运行。它的存在是为了让调度代码(或称为调度器)可以调度多任务。

软中断：

- 硬中断是由硬件产生的，比如，像磁盘，网卡，键盘，时钟等。每个设备或设备集都有它自己的 IRQ(中断请求)。基于 IRQ，CPU 可以将相应的请求分发到对应的硬件驱动上(注:硬件驱动通常是内核中的一个子程序，而不是一个独立的进程)。
- 处理中断的驱动是需要运行在 CPU 上的，因此，当中断产生的时候，CPU 会中断当前正在运行的任务，来处理中断。在有多核心的系统上，一个中断通常只能中断一颗 CPU(也有一种特殊的情况,就是在大型主机.上是有硬件通道的，它可以在没有主 CPU 的支持下，可以同时处理多个中断)。
- 硬中断可以直接中断 CPU。它会引起内核中相关的代码被触发。对于那些需要花费一些时间去处理的进程，中断代码本身也可以被其他的硬中断中断。
- 对于时钟中断，内核调度代码会将当前正在运行的进程挂起，从而让其他的进程来运行。它的存在是为了让调度代码(或称为调度器)可以调度多任务。

操作过程 10

怎么样，试一下吗？用信号量机制编写一个解决生产者—消费者问题的程序，这可是受益匪浅的事。本《实验指导》第五部分有关进程通信的系统调用中介绍了信号量机制的使用。

问题：桌上有一空盘，允许存放一只水果。爸爸可向盘中放苹果，也可向盘中放桔子，儿子专等吃盘中的桔子，女儿专等吃盘中的苹果。规定当盘空时一次只能放一只水果供吃者取用，请用 P、V 原语实现爸爸、儿子、女儿三个并发进程的同步。

分析：在本题中，爸爸、儿子、女儿共用一个盘子，盘中一次只能放一个水果。当盘子为空时，爸爸可将一个水果放入果盘中。若放入果盘中的是桔子，则允许儿子吃，女儿必须等待；若放入果盘中的是苹果，则允许女儿吃，儿子必须等待。

代码实现：

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define P sem_wait
#define V sem_post
#define orange &orange_
#define apple &apple_
#define mutex &mutex_

sem_t orange_;
sem_t apple_;
sem_t mutex_;

void *father(void *arg) {
    while (1) {
        P(mutex);
        sleep(5);
        printf("父亲放橘子。\\n");
        V(orange);
    }
}

void *mother(void *arg) {
    while (1) {
        P(mutex);
        sleep(5);
        printf("母亲放苹果。\\n");
        V(apple);
    }
}
```

```

void *daughter(void *arg) {
    while (1) {
        P(orange);
        sleep(5);
        printf("女儿拿橘子。\\n");
        V(mutex);
        printf("女儿吃橘子。\\n");
    }
}

void *son(void *arg) {
    while (1) {
        P(apple);
        sleep(5);
        printf("儿子拿苹果。\\n");
        V(mutex);
        printf("儿子吃苹果。\\n");
    }
}

int main() {
    pthread_t Father, Mother, Daughter, Son;
    int arg = 1;
    sem_init(&orange, 0, 0);
    sem_init(&apple, 0, 0);
    sem_init(&mutex, 0, 1); //初始化

    pthread_create(&Father, NULL, father, &arg);
    pthread_create(&Daughter, NULL, daughter, &arg);
    pthread_create(&Mother, NULL, mother, &arg);
    pthread_create(&Son, NULL, son, &arg); //创建线程池

    pthread_exit(0);
    return 0;
}

```

结果 10

```

yanxinyu@Thinkbook16-2022:~$ ./prog10.o
父亲放橘子。
女儿拿橘子。
女儿吃橘子。
母亲放苹果。
儿子拿苹果。
儿子吃苹果。

```

讨论

1. 讨论 Linux 系统进程运行的机制和特点，系统通过什么来管理进程？

答：进程在创建时都会被分配一个数据结构，称为进程控制块（PCB）。PCB 中包含了很多重要的信息，供系统调度和进程本身执行使用。所有进程的 PCB 都存放在内核空间中。PCB 中最重要的信息就是进程 PID，内核通过这个 PID 来唯一标识一个进程。PID 可以循环使用，最大值是 32768。init 进程的 pid 为 1，其他进程都是 init 进程的后代。

Linux 操作系统使用一些系统调用(如：fork()、wait、exit 等)来实现对进程的管理。

2. C 语言中是如何使用 Linux 提供的功能的？用程序及运行结果举例说明。

答：作为 UNIX 操作系统的一种，Linux 的操作系统提供了一系列的接口，这些接口被称为系统调用(SystemCall)。在 UNIX 的理念中，系统调用"提供的是机制，而不是策略"。C 语言的库函数通过调用系统调用来实现，库函数对上层提供了 C 语言库文件的接口。在应用程序层，通过调用 C 语言库函数和系统调用来实现功能。一般来说，应用程序大多使用 C 语言库函数实现其功能，较少使用系统调用。例如实验 1 的 fork()函数就是系统调用，需添加头文件#include<stdlib.h>。

3. 什么是进程？如何产生的？举例说明。

答：进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。它是操作系统动态执行的基本单元，在传统的操作系统中，进程既是基本的分配单元，也是基本的执行单元。

进程产生的事件有用户登录、作业调度、提供服务和应用请求。一旦操作系统发现了要求创建新进程的事件后，便调用进程创建原语 Creat()按下述步骤创建一个新进程。申请空白 PCB、为新进程分配资源、初始化进程控制块、将新进程插入就绪队列

例如父进程通过调用 fork()函数会创建一个子进程。

4. 进程控制如何实现的？举例说明。

答：进程控制一般是由 OS 的内核中的原语来实现的。

例子：唤醒原语 wakeup()。唤醒原语执行的过程是：首先把被阻塞的进程从等待该事件的阻塞队列中移出，将其 PCB 中的现行状态由阻塞改为就绪，然后再将该 PCB 插入到就绪队列中。当被阻塞进程所期待的事件出现时，如 I/O 完成或其所期待的数据已经到达，则由有关进程(如用完并释放该 I/O 设备的进程)调用唤醒原语 wakeup()，将等待该事件的进程唤醒。

5. 进程通信方式各有什么特点？用程序及运行结果举例说明。

答：进程间通信可分为 4 种形式：

- **主从式：**主进程可自由地使用从进程的资源或数据；从进程的动作受主进程的控制；主进程和从进程的关系是固定的。
- **会话式：**使用进程在使用服务进程所提供的服务之前，必须得到服务进程的许可；服务进程根据使用进程的要求提供服务，但对所提供服务的控制由服务进程自身完成；使用进程和服务进程在通信时有固定连接关系。
- **消息或邮箱机制：**只要存在空缓冲区或邮箱，发送进程就可以发送消息；与会话系统不同，发送进程和接收进程之间无直接连接关系，接收进程可能在收到某个发送进程发来的消息之后，又转去接收另一个发送进程发来的消息；发送进程和接收进程之间存在缓冲区或邮箱用来存放被传送消息。
- **共享存储区方式：**诸进程可通过对共享存储区中数据的读或写来实现通信；进程在通

信前,先向系统申请获得共享存储区中的一个分区,并指定该分区的关键字;若系统已经给其他进程分配了这样的分区,则将该分区的描述符返回给申请者;由申请者把获得的共享存储分区连接到本进程上;可像读、写普通存储器一样地读、写该公用存储分区。

6. 管道通信如何实现? 该通信方式可以用在何处?

答:管道是一种最基本的进程间通信机制,由 `pipe` 函数来创建。调用 `pipe()` 函数,会在内核中开辟出一块缓冲区用来进行进程间通信,这块缓冲区称为管道,它有一个读端和一个写端,只能血缘关系的进程进行通信,面向字节流的服务,内部提供了同步机制(生产者-消费者)。父进程创建好管道之后,再调用 `fork()` 函数创建子进程,子进程就会继承那个管道,于是父子间可以约定谁来读,谁来写。因为是半双工通信,所以只能一个读,一个写,如果想要两端都能读写,那就要创建两个管道。写进程在管道写端写入数据,读进程在管道读段读出数据实现进程间的通信。

这种通信方式用于数据传输、资源共享和事件通知。

7. 什么是软中断? 软中断信号通信如何实现?

答:软中断是利用硬件中断的概念,用软件方式进行模拟,实现宏观上的异步执行效果。内核线程接受到上半部分软中断请求,就会异步的继续执行上半部未完成的请求,特点是延迟执行。软中断是软件实现的中断,也就是程序运行时其他程序对它的中断;而硬中断是硬件实现的中断,是程序运行时设备对它的中断。

利用 `signal` 和 `kill` 实现软中断通信: `kill(pid,signal)`: 向进程 `pid` 发送信号 `signal`,若 `pid` 进程在可中断的优先级(低优先级)上睡眠,则将其唤醒。`signal(sig,ps)`。

实验体会

从各个题目中，我都有不同的收获，加深了我对操作系统原理的理解与思考。

- 程序虽然具有并发性，但是在很多情况由于其前趋性存在，程序并不会并发执行。
- 进程的产生即赋予 PCB 的过程，进程产生后并不能直接运行，而是需要 os 进行分配资源和调度才能执行，`fork()`函数会给予进程分配父进程的相同的资源和运行环境，而后子进程从生成子进程的地方往后继续运行。同时进程在结束的时候（调用 `exit` 或正常结束）会被撤销进程的 PCB 从而实现结束进程。
- 类似 Linux 目录树的概念，进程产生进程抽象成了单纯的父与子的概念，这样每个进程可以是父进程也会是子进程，方便管理也方便梳理进程与进程间的关系。`Fork` 函数的返回值设计的很巧妙，在父进程中 `fork` 返回的是子进程的进程号，而子进程中则返回 0，在子进程会拷贝父进程剩下的程序的基础上，通过一个简单的 `if` 判断就可以将内容进行分隔，父进程只能访问大于 0 的部分，子进程只能访问=0 的部分。`wait` 则是体现了进程的阻塞，在调用 `wait` 的时候直接将主进程阻塞等待子进程结束父进程才能继续运行。
- 很明显本次实验反应了进程是并行执行的这一特点，之前由于进程总是呈现父子关系而父进程总是要 `wait` 到子进程结束后才继续往下运行所以之前的运行顺序总是固定的，而在本次实验中，三个子进程产生后并无固定的结束顺序规定，其运行是并行的，所以谁调用的进程先结束则结束。
- 进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。创建一个子进程，自然离不开资源的分配，子进程会继承父进程的所有资源，但这种继承并不是传统概念上的继承，在没有不对资源进行改变的时候，可以认为是继承，但在发生改变的时候，实际上是拷贝了，各自对自己占有的资源进行更改。
- 管道是操作系统中很重要的一个概念，它强化了进程与进程之间的联系关系，由于子进程只会拷贝父进程中的资源，如果是要通过子进程改变父进程中的资源，我想管道应该是很好的选择，兄弟进程也是一样的。对于进程的可操作性应该是一个很重要的角色。
- 从结果上来看，如果提前结束父进程，子进程并不会被直接杀死，而是被归为孤儿进程继续运行，此时孤儿进程的 `pid` 变为 1。在这两次尝试中我体会到了管道间通信的重要性，它是进程与进程之间沟通甚至是掌控的重要工具，加强了有血缘关系的进程间的紧密联系。
- 本题体现的是非血缘关系进程之间的通信方式，像这样非父子关系也非兄弟进程之间想要通信的话，就得使用 Linux 的消息机制了。
- 中断感觉也是非常有用的一个机制，进程可以通过中断来提升自己的运行优先级并且有时我们需要通过中断来打断例如死锁等异常情况，对于操作系统来说是一个能提升其运行安全性和操作多样性的重要机制。