使用系统调用对进程进行控制

```
root@instance-20221004-1949:~/class# vim test1.c
root@instance-20221004-1949:~/class# gcc test1.c -o test1
root@instance-20221004-1949:~/class# ls
test1  test1.c
root@instance-20221004-1949:~/class# ./test1
parent context.
parent is waiting the child1 terminate.
child2  context.
child1 's context.
child2 terminate.
parent is waiting the child2 terminate.
child1 terminate.
parent terminate.
root@instance-20221004-1949:~/class#
```

test1.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<wait.h>
int main() {
        int pid1,pid2;
        while((pid1=fork())==-1);
        if(pid1>0) {     //创建子进程成功，处于父进程下
                while((pid2=fork())==-1);
                if(pid2>0) {    //创建子进程成功，处于父进程下
                        printf("parent context.\n");
                        printf("parent is waiting the child1 terminate.\n");
                        wait(0);
                        printf("parent is waiting the child2 terminate.\n");
                        wait(0);
                        printf("parent terminate.\n");
                        exit(0);
                } else {        //处于子进程2下
                        printf("child2  context.\n");
                        sleep(5);
                        printf("child2 terminate.\n");
                }
        } else {        //处于子进程1下
                printf("child1 's context.\n");
                sleep(5);
                printf("child1 terminate.\n");
        }
        return 0;
}
```

使用管道极致进行通信

```
root@instance-20221004-1949:~/class# vim test2.c
root@instance-20221004-1949:~/class# gcc test2.c -o test2
root@instance-20221004-1949:~/class# ls
test1  test1.c  test2  test2.c
root@instance-20221004-1949:~/class# ./test2
parent process sends a message to child.
parent waits the child to terminate.
The message from parent is: A message to pipe'communication.

child process terminates.
parent process terminates.
root@instance-20221004-1949:~/class#
```

test2.c

```c
#include<unistd.h>
#include<wait.h>
#include<stdio.h>
#include<stdlib.h>
char parent[]="A message to pipe'communication.\n";
int main() {
        int pid,chanl[2];       //chanl[2]存放打开文件描述符。chanl[0]管道读端，chanl[1]管道写端。
        char buf[100];
        pipe(chanl);    //创建管道
        while((pid=fork())==-1);
        if(pid>0) {
                close(chanl[0]);        //关闭读通道
                printf("parent process sends a message to child.\n");
                write(chanl[1],parent,sizeof(parent));  //向管道写信息
                close(chanl[1]);        //关闭写通道
                printf("parent waits the child to terminate.\n");
                wait(0);
                printf("parent process terminates.\n");
                exit(0);
        } else {
                close(chanl[1]);        //关闭写通道
                read(chanl[0],buf,100);         //从管道中读取信息
                close(chanl[0]);        //关闭读通道
                printf("The message from parent is: %s\n",buf);
                sleep(5);
                printf("child process terminates.\n");
        }
        return 0;
}
```

test2.c

使用信号机制进行进程通信

```
root@instance-20221004-1949:~/class# vim test3.c
root@instance-20221004-1949:~/class# gcc test3.c -o test3
root@instance-20221004-1949:~/class# ls
test1  test1.c  test2  test2.c  test3  test3.c
root@instance-20221004-1949:~/class# ./test3
parent process killed signal to child process!
child process 19998 were end,return status=0
root@instance-20221004-1949:~/class#
```

test3.c

test3.c

```c
# include<stdio.h>
#include<sys/types.h>
# include<signal.h>
# include<unistd.h>
# include<stdlib.h>
# include<wait.h>
int  wait_mark=1;
void waiting(),catch();
int main() {
    int  p1,status,cpid;
    while((p1=fork())==-1);
    if(p1>0) {
        kill(p1,SIGUSR1);
        printf("parent process killed signal to child process!\n");
        cpid=wait(&status);
        printf("child process %d were end,return status=%d\n",cpid ,WEXITSTATUS(status));
        exit(0);
    } else if(p1==0) {
        signal(SIGUSR1,catch);
        waiting();
        printf("child process catch a signal from parent and ready to end!\n");
        exit(5);
    }
    return 0;
}
void waiting() {
    while(wait_mark!=0);
}
void catch() {
    wait_mark=0;
}
```

总结

本次实验很顺利没有遇到问题，通过本次实验加深了对 Linux 编程的学习，学会了用系统调用和库函数惊醒编程、实现进程控制、进程通信以及文件操作。