

# 华中科技大学

## 实 验 报 告

课 程： 操作系统原理

实验序号： 第 2 次实验

实验名称： 进程通信程序

院 系： 软件学院

专业班级：

学 号：

姓 名：

2023 年 03 月 28 日

## 一、实验目的

1. 理解进程/线程的概念，会对进程/线程进行基本控制；
2. 理解进程/线程的典型通信机制和应用编程；
3. 掌握和推广国产操作系统（推荐银河麒麟或优麒麟）

## 二、实验内容

在 Linux 下利用信号机制(signal)实现进程间通信。父进程创建(fork)子进程，并让子进程进入死循环。子进程每隔 2 秒输出 "I am Child Process, alive !\n"。父进程询问用户 "To terminate Child Process. Yes or No? \n"，要求用户从键盘回答 Y 或 N。若用户回答 N，延迟 2 秒后再提问。若用户回答 Y，向子进程发送用户信号，让子进程结束。子进程结束之前打印字符串："Bye,Wolrd !\n"。

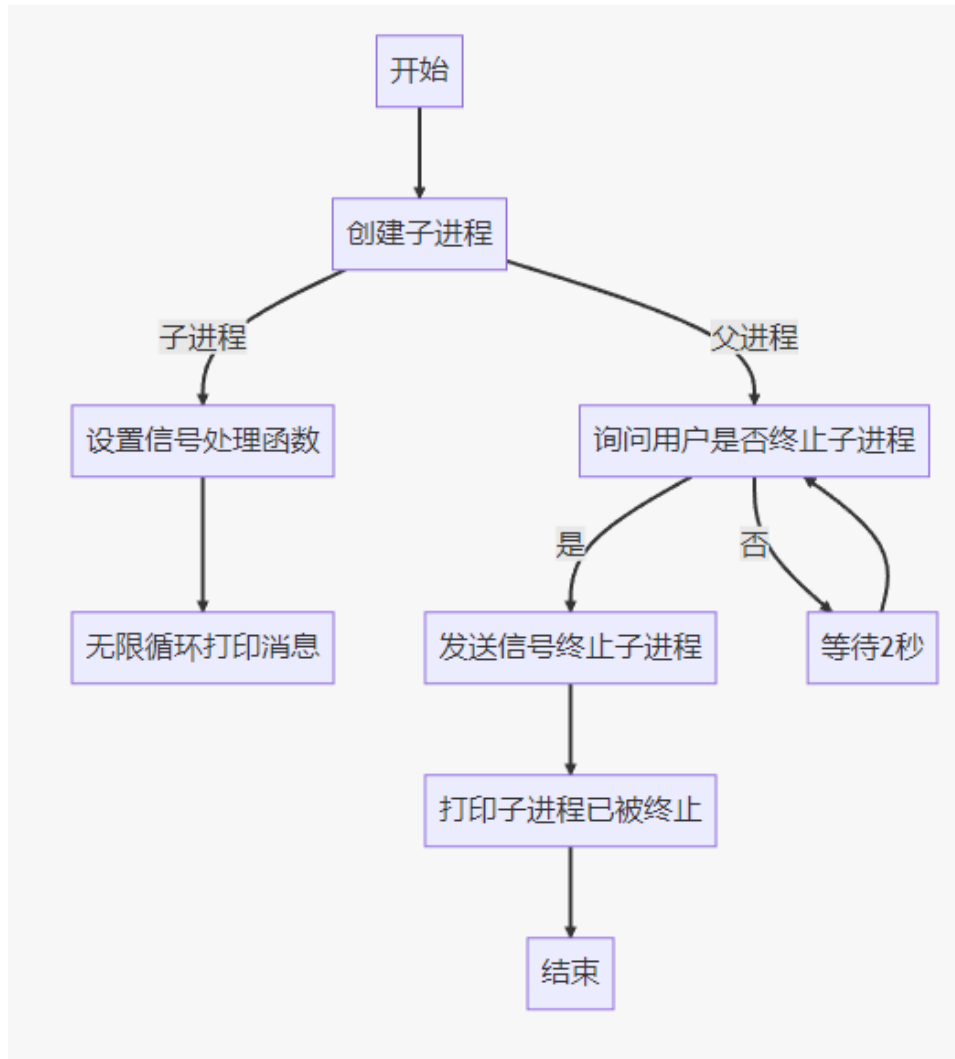
## 三、实验环境

麒麟 Kylin(V10) + gcc/g++ + vi/vim/vscode/notepad++

## 四、程序的设计思路

### 4.1 程序结构（或程序流程、或程序原理，任选一个或多个组合）

1. 如果是子进程：
  - 设置信号处理函数、无限循环打印消息
2. 如果是父进程：
  - 询问用户是否终止子进程：如果用户选择是，发送信号终止子进程，并打印"子进程已被终止"，结束程序；如果用户选择否，等待 2 秒，然后再次询问用户。



## 4.2 关键函数或参数或机制

### 1. signal()

程序可用使用 signal 函数来处理指定的信号，主要通过忽略和恢复其默认行为来工作。signal 函数的原型如下：

```
void (*signal(int sig, void (*func)(int)))(int);
```

这是一个相当复杂的声明，signal()是一个带有 sig 和 func 两个参数的函数，func 是一个类型为 void (\*)(int)的函数指针。准备捕获的信号参数由 sig 给出，接收到的指定信号后要调用的函数由参数 func 给出。该函数返回一个类型为 void (\*)(int)的函数指

针，指向先前指定信号处理函数的函数指针。

注意信号处理函数的原型必须为 `void func (int)`，或者是下面的特殊值：

✧ `SIG_IGN`：忽略信号

✧ `SIG_DFL`：恢复信号的默认行为

```
void child_process_over(int signum){
    printf("Bye,world!\n");
    kill(getpid(),SIGKILL);
}

signal(SIGTERM,child_process_over);
```

2. `kill()` - `int kill(pid_t pid, int sig);`

参数 `pid`(进程标识符)规定把信号发送到何处，`pid` 不同值的含义如下：

✧ `pid>0` 表示把信号 `sig` 发送给进程标识符为 `pid` 的进程。

✧ `pid=0` 表示把信号 `sig` 发送给同一个进程组的进程。

✧ `pid=-1` 表示把信号 `sig` 发送给除了调用者自身以外所有进程标识符 `pid` 大于 1 的进程。

✧ `pid<-1` 表示把信号 `sig` 发送给进程组中的所有进程。

参数 `sig` 规定发送哪个信号，为 0 时，不发送任何信号(即空信号)，但照常进行错误检查。因此，它可用于检查目标进程是否存在，以及当前进程是否具有向目标发送信号的权限。`kill()`最常见的用法是，用于 `pid>0` 时的信号发送，调用成功返回 0；否则返回 -1。

## 五、关键代码分析

## 5.1 程序关键片段一：引入库和定义函数

这部分代码主要是对程序所需的库进行引用，并定义了一个函数

terminate\_child\_process，该函数在接收到 SIGTERM 信号时被调用，用于终止子进程。

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <signal.h>

#include <unistd.h>

void terminate_child_process(int signum){

    printf("Goodbye, world!\n");

    kill(getpid(),SIGKILL);

}
```

## 5.2 程序关键片段二：主函数和进程创建

主函数中，首先定义了一个进程 ID 变量 child\_pid，然后调用 fork()函数创建了一个子进程。

```
int main()

{

    pid_t child_pid;

    child_pid = fork();
```

## 5.3 程序关键片段三：子进程的行为定义

在子进程中，首先设置了一个信号处理函数，当接收到 SIGTERM 信号时，调用 terminate\_child\_process 函数。然后进入一个无限循环，每 2 秒打印一次"I am the child process and I am still alive"。

```
if(child_pid == 0)
{
    signal(SIGTERM,terminate_child_process);

    for(;;){

        printf("I am the child process and I am still alive \n");

        sleep(2);

    }

}
```

#### 5.4 程序关键片段四：父进程的行为定义

在父进程中，进入一个循环，每次询问用户是否要终止子进程，如果用户输入'Y'或'y'，则发送 SIGTERM 信号给子进程，然后打印"Child process has been terminated."并跳出循环；否则，父进程休眠 2 秒后继续询问。

```
else
{
    for(int i=0;i<100;i++){

        printf("Do you want to terminate the child process? Yes or No?\n");

        char user_input=getchar();

        getchar();

        if(user_input=='Y' || user_input=='y'){

            kill(child_pid,SIGTERM);

            printf("Child process has been terminated.\n");

            break;

        }else sleep(2);

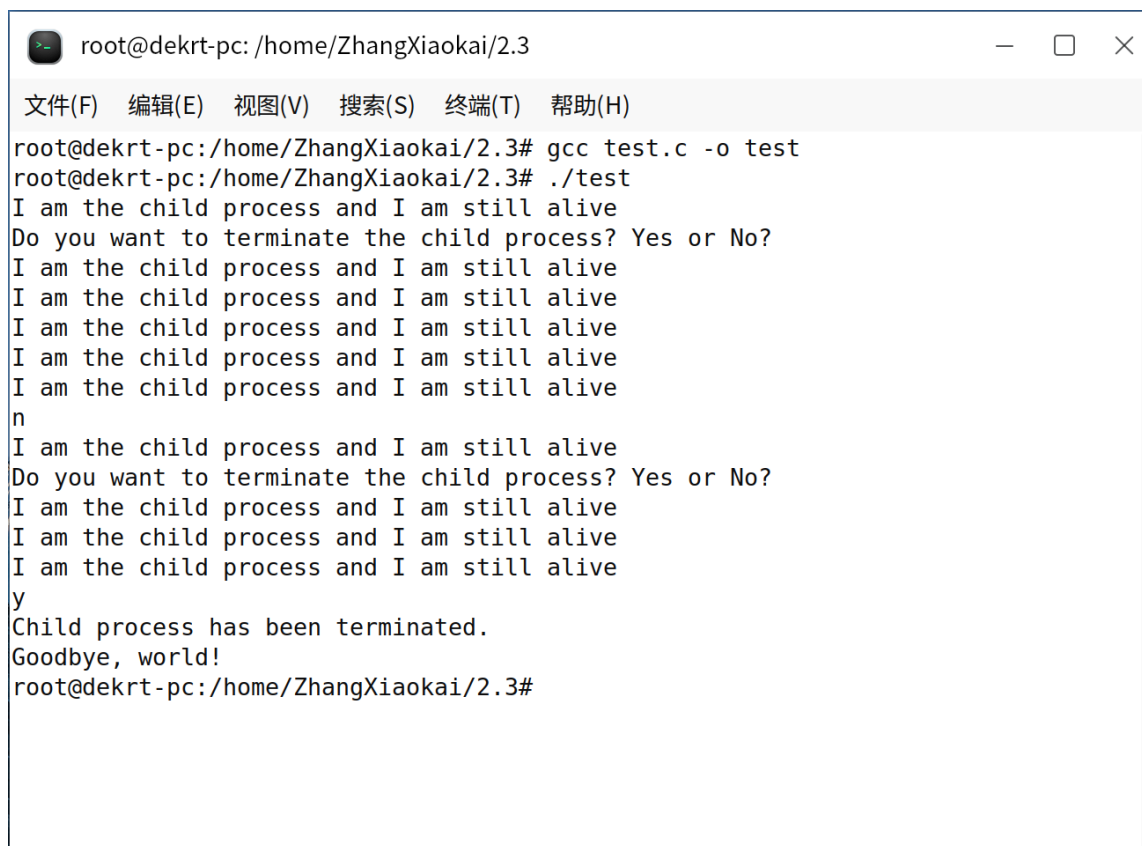
    }

}

return 0;

}
```

## 六、程序运行结果和分析

A terminal window titled 'root@dekrt-pc: /home/ZhangXiaokai/2.3' with standard window controls. The terminal shows the execution of a C program. The user runs 'gcc test.c -o test' and then './test'. The program prints 'I am the child process and I am still alive' multiple times, asks 'Do you want to terminate the child process? Yes or No?', and responds to 'n' and 'y' inputs. Finally, it prints 'Child process has been terminated.' and 'Goodbye, world!' before returning to the shell prompt.

```
root@dekrt-pc: /home/ZhangXiaokai/2.3
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
root@dekrt-pc:/home/ZhangXiaokai/2.3# gcc test.c -o test
root@dekrt-pc:/home/ZhangXiaokai/2.3# ./test
I am the child process and I am still alive
Do you want to terminate the child process? Yes or No?
I am the child process and I am still alive
I am the child process and I am still alive
I am the child process and I am still alive
I am the child process and I am still alive
I am the child process and I am still alive
I am the child process and I am still alive
n
I am the child process and I am still alive
Do you want to terminate the child process? Yes or No?
I am the child process and I am still alive
I am the child process and I am still alive
I am the child process and I am still alive
y
Child process has been terminated.
Goodbye, world!
root@dekrt-pc:/home/ZhangXiaokai/2.3#
```

如图所示，子进程被正确创建并在死循环中打印字符串。父进程负责与用户交互，当用户输入“y”时，父进程通过发送信号成功地将子进程终止。

## 七、实验错误排查和解决方法

### 7.1 父进程 kill 子进程后，子进程无法在退出前打印 goodbye

不由父进程直接 kill 子进程，父进程只是给子进程发送 SIGTERM 信号，由子进程接收到该进程后自己结束。

### 7.2 父进程在收到 n 的输入后，会循环两次，打印两次问题

在终端输入 n 并且回车之后，会有两个字符输入，程序第一次接收了字符 n，然后进入了下一个循环，又接收了回车的字符，导致每次输入后程序多进行了一次循环。

应当在使用 getchar 接收字符后，再调用一次 getchar，将缓冲区回车字符清除掉。

### 7.3 提示 signal 或 kill 函数未定义

需要引入 signal.h 头文件

## 六、实验参考资料和网址

(1) 教学课件

(2) [https://www.tutorialspoint.com/c\\_standard\\_library/c\\_function\\_signal.htm](https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm)