操 作 系 统
实 验 报 告

实验名称：实验二 进程间通信和命令解释器

姓名：刘亚辉

学号：16340157

实验名称：进程间通信和命令解释器

一、实验目的：

1. 进程间共享内存实验，初步了解这种进程间通讯
2. 实现简单的 shell 命令解释器：了解程序运行。

二、实验要求：

1. 完成课本第三章的练习 3.10 的程序
2. 完成课本上第三章的项目：实现 shell。除此之外满足下面要求：
   a) 实现程序的后台运行

三、实验过程：

1. 利用进程间通信实现斐波那契数列的生成（子进程）与输出（父进程）

在整个实现的过程中，主要将代码分为三个部分：共享存储区的创建，子进程生成斐波那契数列，父进程输出数列并删除共享存储区。

- 共享存储区的创建：

```
scanf("%d", &fib.sequence_size);
if((key = ftok("./share", 1)) < 0) {
    printf("ftok error:%s\n", strerror(errno));
    return -1;
}
if((shmid = shmget(key, sizeof(shared_data), 0600|IPC_CREAT)) < 0){
    printf("shmget error:%s\n", strerror(errno));
    exit(-1);
}
if((shmptr = (char*)shmat(shmid, 0, 0)) == (void*)-1) {
    printf("shmat error:%s\n", strerror(errno));
    exit(-1);
}
```

- 在上面的代码中，首先利用 ftok()函数获得一个用于 IPC 通讯的 key 值，其中 “./share” 表示将当前文件夹下的子目录 share 文件夹作为共享存储区，当 key<0 时处理异常；然后利用 shmge 函数和之前获得的 key 值来创建或打开共享存储区，并返回一个共享存储区的 ID。之后连接共享存储

区，返回一个指向共享存储区的指针（在这里连接存储区的话，子进程和父进程无需单独再链接存储区）。

- 子进程生成斐波那契数列：

```
pid=fork();

if(pid==0){
    memset(fib.fib_sequence, 0, sizeof(fib.fib_sequence));
    long num1=0, num2=1;
    for(int i=0;i<fib.sequence_size; i++){
        fib.fib_sequence[i]=num2;
        num2=num1+num2;
        num1=num2-num1;
    }
    memcpy(shmptr, &fib, sizeof(fib));
    printf("child:pid is %d,share memory from %lx to %lx, content:%s\n",
        getpid(), (unsigned long)shmptr, (unsigned long)(shmptr + sizeof(fib)),
shmptr);
    printf("child process has store the first %d fibonacci numbers in the share
memory\n", fib.sequence_size);
    printf("child process exit successfully!\n");
}
```

- 首先，创建子进程，并判断 pid 以进入子进程，然后生成斐波那契数列，输出一些提示信息并退出。

- 父进程输出数列并删除共享存储区。

```
else{
    wait(NULL);
    printf("parent:pid is %d,share memory from %lx to %lx, content:%s\n",
        getpid(),(unsigned long)shmptr, (unsigned long)(shmptr + sizeof(shared_d
ata)), shmptr);
    printf("parent process read the fibonacci sequence from the share memory\n")
;

    shared_data *fib=(shared_data*)shmptr;
    for(int i=0;i<fib->sequence_size;i++){
        printf("%ld ", fib->fib_sequence[i]);
    }
    printf("\n");

```

```
        if((shmctl(shmid, IPC_RMID, 0) < 0)){
            printf("shmctl error:%s\n", strerror(errno));
            exit(-1);
        }
        printf("delete the share memory and exit successfully!\n");
    }
```

- 在父进程中，调用 wait(NULL)等待子进程运行结束。然后输出共享存储区中的数据信息，最后调用 shmctl（共享存储区控制函数），并传入参数 IPC_RMID 以删除存储区。
- 最终代码执行结果如下：

```
liuyh73@ubuntu:~/Desktop/OperatorSystem/test2$ gcc share_memory_fib.c
liuyh73@ubuntu:~/Desktop/OperatorSystem/test2$ ./a.out
Please input a number less than 10:
7
child:pid is 70476,share memory from 7f462feb1000 to 7f462feb1058, content:
child process has store the first 7 fibonacci numbers in the share memory
child process exit successfully!
parent:pid is 70475,share memory from 7f462feb1000 to 7f462feb1058, content:
parent process read the fibonacci sequence from the share memory
1 1 2 3 5 8 13
delete the share memory and exit successfully!
```

2.实现 shell 并完成程序后台运行

实现思路主要如下：通过创建子进程，利用子进程和父进程各自处理各自的函数。父进程调用 setup 函数，获取 shell 输入的命令；子进程调用 execvp 函数，执行该命令。

具体实现如下：

主体代码：

首先输出提示字符，并调用 fflush 函数刷新缓冲区，输出“$CMD->”；然后主进程调用 setup 函数，获取 shell 输入的命令字符串。之后 fork 得到子进程，子进程调用 exevcp 函数执行该命令，父进程再 background==0 的时候等待

子进程，否则进入等待下一条命令的输入。

```
while(1){
    background=0;

    printf("$CMD->");
    fflush(stdout);

    setup(inputBuffer, args, &background);

    pid_t pid=fork();
    if(pid==0){
        execvp(args[0], args);
    }
    else{
        if(background==0){
            waitpid(pid, NULL, 0);
        }
    }
}
```

setup 函数实现如下：

```
void setup(char inputBuffer[], char *args[], int *background){
    int length, start=-1, ans=0;
    length=read(STDIN_FILENO, inputBuffer, MAX_LINE);

    if(length==0) exit(0);
    if(length<0){
        perror("error reading the command");
        exit(-1);
    }
    inputBuffer[length]='\0';

    strcpy(signals[right++], inputBuffer);

    if(inputBuffer[0]=='r' && (inputBuffer[1]==' ' || inputBuffer[1]=='\t'))
{
        for(int i=right-2; i>=left; i--){
            if(signals[i][0]==inputBuffer[2]){
                strcpy(inputBuffer, signals[i]);
```

```
                length=strlen(inputBuffer);
                break;
            }
        }
    }
    if(right-left>10) left++;
    for(int i=0;i<length;i++){
        switch(inputBuffer[i]){
            case ' ':
            case '\t':
                if(start!=-1)
                    args[ans++]=&inputBuffer[start];
                inputBuffer[i]='\0';
                start=-1;
                break;
            case '\n':
                if(start!=-1)
                    args[ans++]=&inputBuffer[start];
                inputBuffer[i]='\0';
                args[ans]=NULL;
                break;
            default:
                if(start==-1)
                    start=i;
                if(inputBuffer[i]=='&'){
                    *background=1;
                    inputBuffer[i]='\0';
                }
        }
    }
    args[ans]=NULL;
}
```

在该函数中，调用 read 函数来读取命令字符串并存储在 inputBuffer 字符数组中，利用队列来存储输入的命令，并且实现 r x 命令执行最近的一条以 x 开头的命令。在后面的循环中，利用 switch 和 case 语句来将输入的命令进行切分，存储在 args 指针数组中。

最后实现 Ctrl+C 输出最近的 10 条历史记录：

先在主函数中注册 Ctrl+C 监听事件：

```
signal(SIGINT, &handle_SIGINT);
```

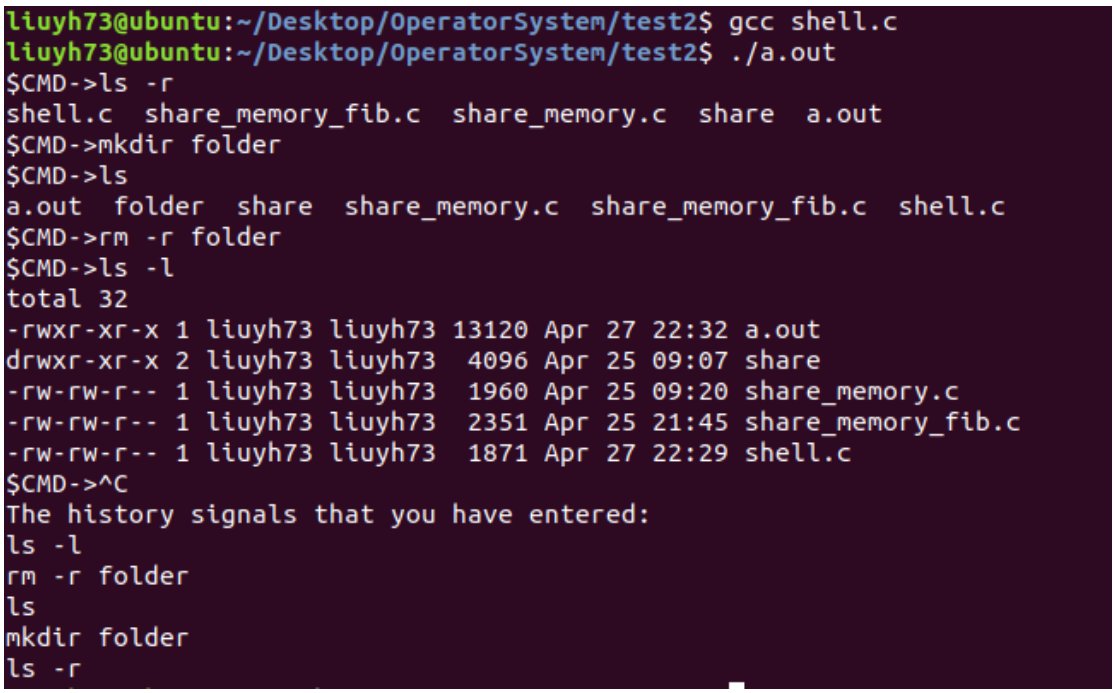
然后实现 handle_SIGINT 处理事件函数：

```
void handle_SIGINT(){
    printf("\nThe history signals that you have entered: \n");
    for(int i=right-1; i>=left; i--){
        write(STDOUT_FILENO, signals[i], strlen(signals[i]));
    }
    printf("$CMD->");
    fflush(stdout);
}

```

通过上述代码即可实现历史记录的输出。

最终实现效果如下：



r x 实现效果如下：

```
liuyh73@ubuntu:~/Desktop/OperatorSystem/test2$ gcc shell.c
liuyh73@ubuntu:~/Desktop/OperatorSystem/test2$ ./a.out
$CMD->ls
a.out  share  share_memory.c  share_memory_fib.c  shell.c
$CMD->r l
a.out  share  share_memory.c  share_memory_fib.c  shell.c
$CMD->ls -r
shell.c  share_memory_fib.c  share_memory.c  share  a.out
$CMD->ls -l
total 32
-rwxr-xr-x 1 liuyh73 liuyh73 13200 Apr 27 22:41 a.out
drwxr-xr-x 2 liuyh73 liuyh73  4096 Apr 25 09:07 share
-rw-rw-r-- 1 liuyh73 liuyh73  1960 Apr 25 09:20 share_memory.c
-rw-rw-r-- 1 liuyh73 liuyh73  2351 Apr 25 21:45 share_memory_fib.c
-rw-rw-r-- 1 liuyh73 liuyh73  1910 Apr 27 22:37 shell.c
$CMD->r l
total 32
-rwxr-xr-x 1 liuyh73 liuyh73 13200 Apr 27 22:41 a.out
drwxr-xr-x 2 liuyh73 liuyh73  4096 Apr 25 09:07 share
-rw-rw-r-- 1 liuyh73 liuyh73  1960 Apr 25 09:20 share_memory.c
-rw-rw-r-- 1 liuyh73 liuyh73  2351 Apr 25 21:45 share_memory_fib.c
-rw-rw-r-- 1 liuyh73 liuyh73  1910 Apr 27 22:37 shell.c
$CMD->
```