**操 作 系 统**

**实 验 报 告**

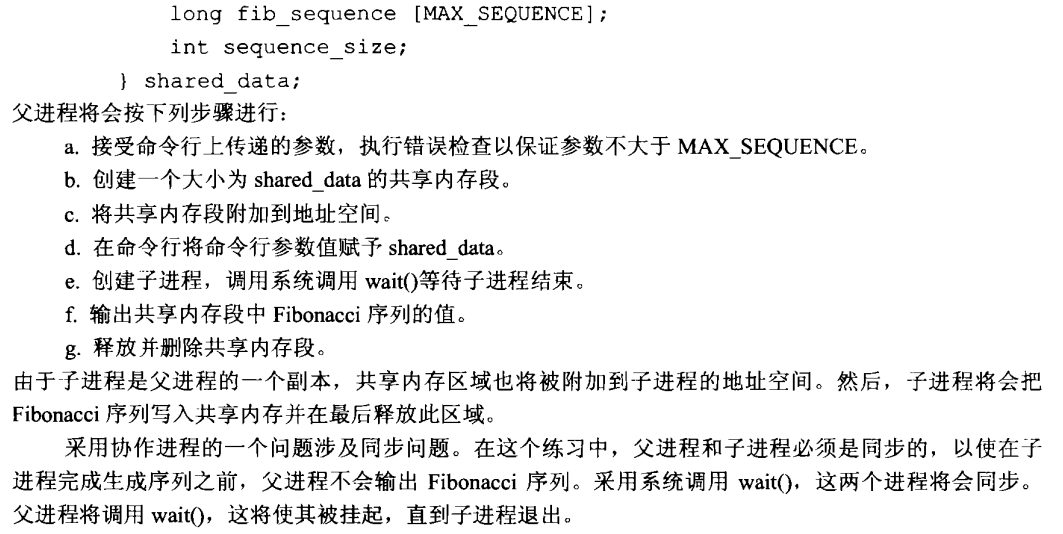
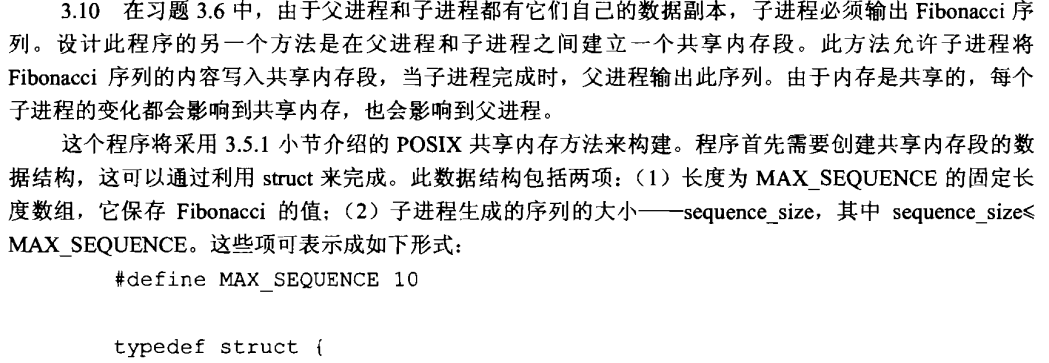
**实验名称：实验二 进程间通信和命令解释器**

**姓名： 王迎旭**

**学号： 16340226**

实验名称：**进程间通信和命令解释器**

1. 实验目的：
2. 进程间共享内存实验，初步了解这种进程间通讯
3. 实现简单的shell命令解释器：了解程序运行。
4. 实验要求：
5. 进程间共享内存实验：完成课本第三章的练习3.10的程序
6. shell的要求：参考书上第三章项目实现shell。除此之外还需要实现程序的后台运行
7. 实验过程：
8. **进程间共享内存实验**



**1**.明确目标程序的需求：

①使用共享内存来为解决父进程与子进程中数据存放的问题

②使用子进程完成斐波拉契数列各项的计算，使用父进程输出斐波拉契数列各项的具体数值

**2**.搜索题目所涉及相关资料：

Ⅰ、**了解int main(int argc,char\* argv[ ])函数中参数argc与\*argv[]的用法 ：**

① argc：命令行总的参数的个数,即argv中元素的格式。

② \* argv[ ]: 字符串数组,用来存放指向你的字符串参数的指针数组, 每一个元素指向一个参数

**Ⅱ、了解共享内存模块相关资料**

① 创建或打开共享存储区(**shmget**)：依据用户给出的整数值key，创建新区或打开现有区，返回一个共享存储区ID。

② 连接共享存储区(**shmat**)：连接共享存储区到本进程的地址空间，返回共享存储区首地址。父进程已连接的共享存储区可被fork创建的子进程继承。

③ 拆除共享存储区连接(**shmdt**)：拆除共享存储区与本进程地址空间的连接。

④ 共享存储区控制(**shmctl**)：对共享存储区进行控制。如：共享存储区的删除需要显式调用shmctl(shmid, IPC\_RMID, 0)；

⑤ 用系统调用**ftok**给出IPC键值key：保证同一个用户的两个进程获得相同的IPC键值key

**Ⅲ、了解wait()函数以及waitpid()函数的用法**

**① wait()**

A、函数介绍：

wait()会暂时停止目前进程的执行，直到有信号来到或子进程结束。如果在调用wait()时子进程已经结束，则wait()会立即返回子进程结束状态值。子进程的结束状态值会由参数status 返回，而子进程的进程识别码也会一快返回。如果不在意结束状态值，则参数status 可以设成NULL。子进程的结束状态值请参考waitpid()。

B、返回值类型

如果执行成功则返回子进程识别码（PID），如果有错误发生则返回-1。失败原因存于errno 中。

**② waitpid()**

A、函数介绍：

waitpid()会暂时停止目前进程的执行，直到有信号来到或子进程结束。如果在调用waitpid()时子进程已经结束，则waitpid()会立即返回子进程结束状态值。子进程的结束状态值会由参数status 返回，而子进程的进程识别码也会一快返回。如果不在意结束状态值，则参数status 可以设成NULL。

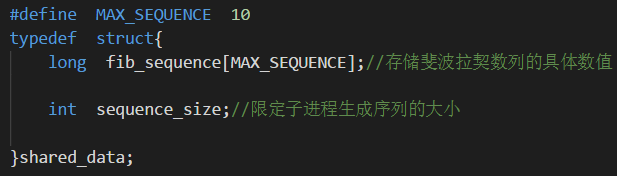
1. 返回值类型

如果执行成功则返回子进程识别码（PID），如果有错误发生则返回-1。失败原因存于errno 中。

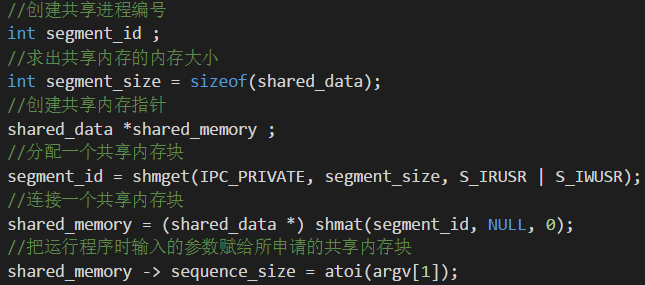
**3**.斐波拉契数列的程序设计与运行

**Ⅰ、针对斐波拉契数列数列理清设计思路**

① 使用结构体构建共享内存模块



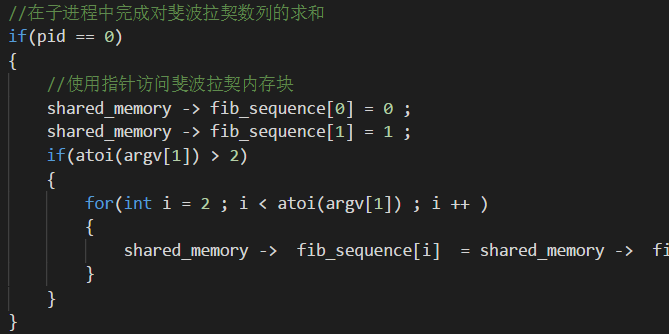
② 创建共享内存块并开辟指针空间进行连接

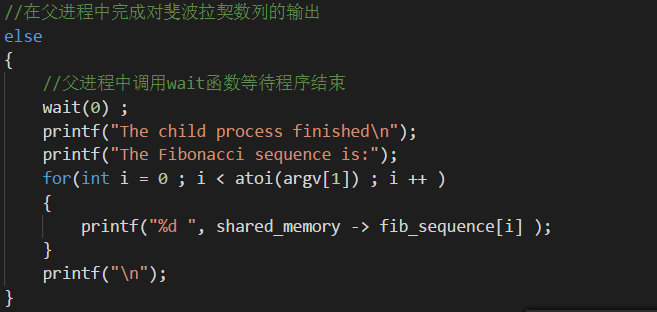


③ 创建子进程

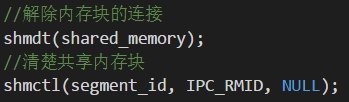


④ 父进程中设置wait()函数等待子进程结束输出求和序列，子进 程中完成对斐波拉契数列求和





⑤ 共享内存块的解除连接与释放空间

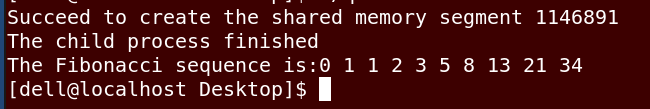


**Ⅱ、编译运行并输出结果**

① 编译运行

IMG_256

② 结果显示



分析：

1. 在运行时候使用./question1 10指令,这里是为了使main函数中的参数argc = 2 以及 argv[1] = 10 ，这么做也是在程序运行之前直接给其传入运行参数，而非程序运行之后传入参数。
2. 创建共享内存块id为1146891的共享内存块；随后父进程进入wait()函数中，等待子进程完成；当子进程中赋值过程完成之后，子进程会printf一句话“The child process finished”，并结束；父进程完成对斐波拉契数列的输出；最后整个程序取消共享内存块的连接，释放共享内存。
3. **Shell的实现**

**1、明确目标程序的需求：**

① 通过C程序实现一个具备基本功能的shell

② 实现程序的后台运行

1. **搜索题目所涉及相关资料：**

Ⅰ、**execvp()函数**

1. 函数介绍

execvp()会从PATH 环境变量所指的目录中查找符合参数file 的文件名，找到后便执行该文件，然后将第二个参数argv传给该欲执行的文件。

B、返回值

如果执行成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno中。

Ⅱ、**setup()函数**

1. 函数介绍

setup() 用于读入下一行输入的命令，并将它分成没有空格的命令和参数存于数组args[]中，用NULL作为数组结束的标志。

1. 返回值类型

无返回值

Ⅲ、**信号捕捉函数signal(int signum, sighandler\_t handler)**

1. 函数介绍

设置对某一信号的捕捉。接收到一个类型为sig的信号时，就执行handler 所指定的函数。（int）signum是传递给它的唯一参数。执行了signal()调用后，进程只要接收到类型为sig的信号，不管其正在执行程序的哪一部分，就立即执行handler函数。当handler函数执行结束后，控制权返回进程被中断的那一点继续执行。

 函数第一个参数signum：指明了所要处理的信号类型，它可以取除了SIGKILL和SIGSTOP外的任何一种信号。

函数第二个参数handler：描述了与信号关联的动作，它可以取以下三种值：

1)SIG\_IGN:表示忽略该信号。

2)SIG\_DFL:表示恢复对信号的系统默认处理,不写此处理函数默认也是执行系统默认操作。

3)sighandler\_t类型的函数指针，接收到sig信号之后进行函数调用

B、返回值：

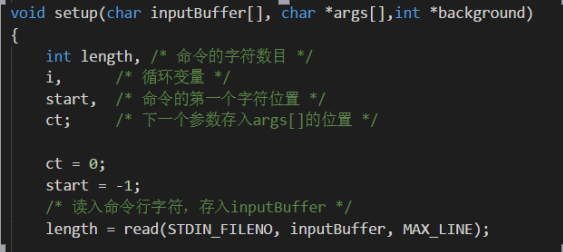
返回先前的信号处理函数指针，如果有错误则返回SIG\_ERR(-1)。

**Ⅲ、编写程序实现shell**

① 首先限定命令行最多能读入的字符个数



② 参考PPT，实现setup()函数（部分截图）

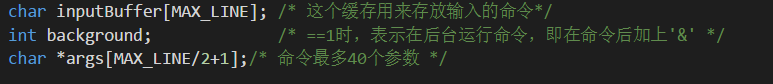


③ 实现main主函数

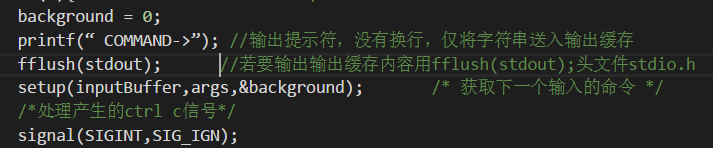
**注：由于一般程序使用ctrl + c 就能强制程序退出，但是shell中需要使用ctrl + d指令完成程序退出功能，所以这里需要使用信号捕捉函数对ctrl + c 指令进行处理，让程序读到ctrl + c指令之后什么都不做。**

具体步骤:

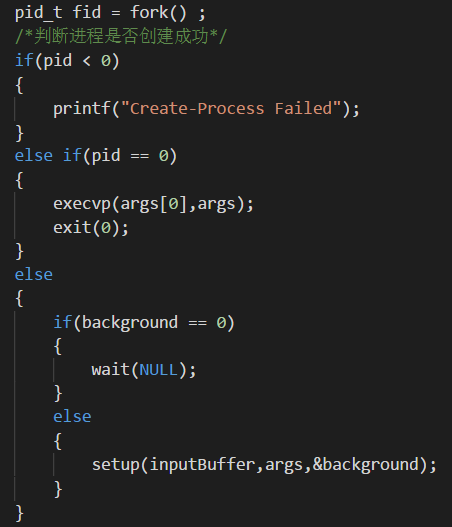
Ⅰ、设置main函数运行参数



Ⅱ、完成指令输入部分，并设置信号捕捉函数，处理ctrl + c导致进程结束的问题

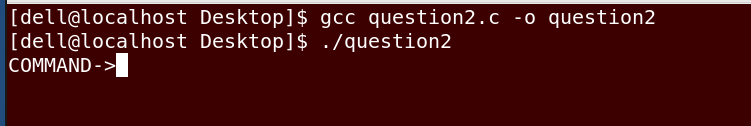


Ⅲ、创建子进程，完成程序



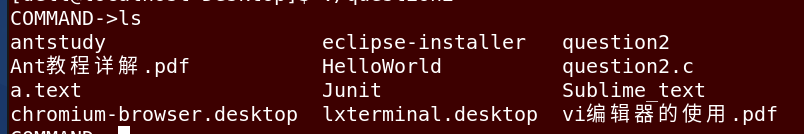
Ⅳ、shell程序的编译及运行

A、编译运行



B、运行结果显示

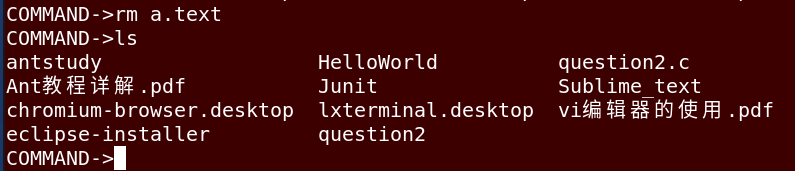
1. ls指令：



1. ctrl + c 指令



1. rm 指令



分析:

① 通过使用ls指令，成功显示出Desktop上保留的文件夹

② 同时ctrl + c信号量也被成功捕捉，并且并没有造成程序退出

③ 使用rm指令之后，发现a.text被成功删除，证明shell程序的完整性

1. 实验心得：

1.关于共享内存块的创建于使用部分，在参考了PPT所给的资料之后，完成起来并没有遇到太大的阻力，但是在使用指针的时候，还是有点不太熟练，就导致报错很多，但是语法上的错误还是慢慢的更正了过来。

2.关于创建shell并完成相应功能部分，这个部分应该是问题比较大的，因为初次使用PPT中所给的函数，遇到了一定的困难，比如setup函数的运行机制以及execvp函数在shell程序中的用途；同时在参考书本设置信号捕捉量也是遇到了一定的问题，比如如何设计指令让程序可以避免读到ctrl + c这个中断；虽然说问题比较多，但是还是花了时间把这些问题一一解决了，这次的实验也是让我对程序的进程运行有了更深的了解。