《操作系统》课程实验二

实验报告

高级进程间通信问题

〈二元自然数变量函数计算问题〉

班级：无 47

姓名：刘 前[1]

学号：2014011216

日期：2016 年 11 月 25 日

操作平台：Windows 8.1

编程语言：C++

[1] 清华大学电子工程系(E-mail: liuqian14@mails.tsinghua.edu.cn)

高级进程间通信问题

实验目的：

1. 通过对进程间高级通信问题的编程实现，加深理解进程间高级通信的原理；
2. 对 Windows 或 Linux 涉及的几种高级进程间通信机制有更进一步的了解；
3. 熟悉 Windows 或 Linux 中定义的与高级进程间通信有关的函数。

实验题目：

本实验共有 2 个实验题目，任选其中之一。不同实验的难度不同，基准分也不同，请同学根据自己的情况加以选择。

实验题目	基准分
二元自然数变量函数计算问题	90
快速排序问题	100

操作系统平台可选 Windows 或 Linux，编程语言不限。

实验报告内容要求：

1. 写出设计思路和程序结构，并对主要代码进行分析；
2. 实际程序运行情况；
3. 对提出的问题进行解答；
4. 体会或者是遇到的问题。

注：本人在实验二中选择的是**二元自然数变量函数计算问题**。

二元自然数变量函数计算问题

一、 问题描述及实验要求

1. 问题描述：

设有二元自然数变量函数 $F(m,n)=f(m)+g(n)$，其中

$$f(m)=\begin{cases} f(m-1)*m & , \quad m>1 \\ 1 & , \quad m=1 \end{cases}$$

$$g(n)=\begin{cases} g(n-1)+g(n-2) & , \quad n>2 \\ 1 & , \quad n=1,2 \end{cases}$$

请编程建立 3 个并发协作进程或线程，分别完成计算 $F(m,n)$、$f(m)$ 和 $g(n)$。

2. 实验步骤

(a) 首先创建三个线程（或进程），分别执行函数 $F(m,n)$、$f(m)$ 和 $g(n)$ 计算；

(b) 线程（或进程）之间的通信可以选择下述机制之一进行：

- 管道（无名管道或命名管道）
- 消息队列
- 共享内存

(c) 通过适当的函数调用创建上述 IPC 对象，通过调用适当的函数调用实现数据的读出与写入；

(d) 需要考虑线程（或进程）间的同步；

(e) 线程（或进程）运行结束，通过适当的系统调用结束线程（或进程）。

3. 实验平台和编程语言：

自由选择 Windows 或 Linux。
编程语言不限。

4. 思考题

1. 你采用了你选择的机制而不是另外的两种机制解决该问题，请解释你做出这种选择的理由。
2. 你认为另外的两种机制是否同样可以解决该问题？如果可以请给出你的思路；如果不能，请解释理由。

二、 设计思路

首先对二元自然数变量函数计算问题进行分析。函数 f(m)的功能是计算输入自然数 m 的阶乘；函数 g(n)的功能是计算输入自然数 n 对应的斐波那契数；函数 F 将函数 f 和 g 的计算结果相加。

通过分析可以看出，如果不要求使用高级进程间通信，即使是编程初学者，也能在很短的时间内完成这一问题，但是本次实验要求使用高级进程间通信解决这一问题，因而需要 IPC 机制来实现进程(线程)之间的通信。IPC 机制要求对三个函数建立各自的线程，函数间参数或数值结果的传递采用消息队列、共享内存或管道等高级进程间通信进制来实现。本人采用匿名管道的方式实现进程间通信。

根据问题要求，本问题需要两个匿名管道，一个管道实现函数 f 和 F 之间的通信，另一个管道实现 g 和 F 之间的通信，为方便下文表述将两个管道分别称为管道 A、B。每个管道在被创建时都会生成两个句柄，分别为管道的读句柄和写句柄，分别与 WriteFile 和 ReadFile 函数协同实现向管道的写入和读取。

使用匿名管道机制解决本问题的主要思路是：

1. 主函数 main( )输入两个自然数，分别为函数 f 的参数 m 和 g 的参数 n；
2. 将 m 写入管道 A，n 写入管道 B；
3. f 函数从管道 A 中读取已经被写入的 m，经过计算得到结果 f(m)，并将结果写回管道 A；
4. g 函数从管道 B 中读取已经被写入的 n，经过计算得到结果 g(n)，并将结果写回管道 B；
5. F 函数从管道 A 和 B 中分别读取 f(m)和 g(n)，相加得到最后结果；
6. 将函数 F 的最后结果输出。

综合以上分析，可以看出，本实验的思路非常清晰，关键是如何使用 Windows 中管道机制的函数实现以上过程。

三、 程序结构

1. 基本数据结构及函数：

本次程序比较简单，没有使用特殊的数据结构，主要涉及的变量包括管道的读写句柄、线程的句柄，以及管道机制和线程中常用的函数。

程序中具体使用的变量和函数及其说明如下：

变量	说明
HANDLE HREAD_fF	f 和 F 之间的管道读句柄
HANDLE HWRITE_fF	f 和 F 之间的管道写句柄
HANDLE HREAD_gF	g 和 F 之间的管道读句柄
HANDLE HWRITE_gF	g 和 F 之间的管道写句柄
HANDLE f_thread	f 函数对应的线程
HANDLE g_thread	g 函数对应的线程
HANDLE F_thread	F 函数对应的线程

自己编写的函数及说明如下：

函数	说明
int f(int m)	函数 f(m)
int g(int n)	函数 g(n)
void WINAPI　f_PIPE_RW(PVOID pvParam)	f 函数向管道的读写操作
void WINAPI　g_PIPE_RW(PVOID pvParam)	g 函数向管道的读写操作
void WINAPI　F_PIPE_RW(PVOID pvParam)	F 函数向管道的读写操作

Windows.h 中对管道的操作函数：

与管道相关的函数	函数说明
CreatePipe( )	创建管道
ReadFile( )	读取数据
WriteFile( )	写入数据
CloseHandle( )	关闭管道

Windows.h 中对线程的操作函数：

与线程的相关函数	函数说明
CreateThread( )	创建线程
CloseHandle( )	关闭线程
WaitForSingleObject( )	等待进程结束

2. 实现方法：

首先，使用递归方法实现函数 f 和函数 g。函数 f(m)的功能是求 m 的阶乘，而函数 g(n)的功能是求第 n 个斐波那契数。然后创建三个线程，f、g 和 F 三个函数分别对应线程 f_thread，g_thread 和 F_thread。

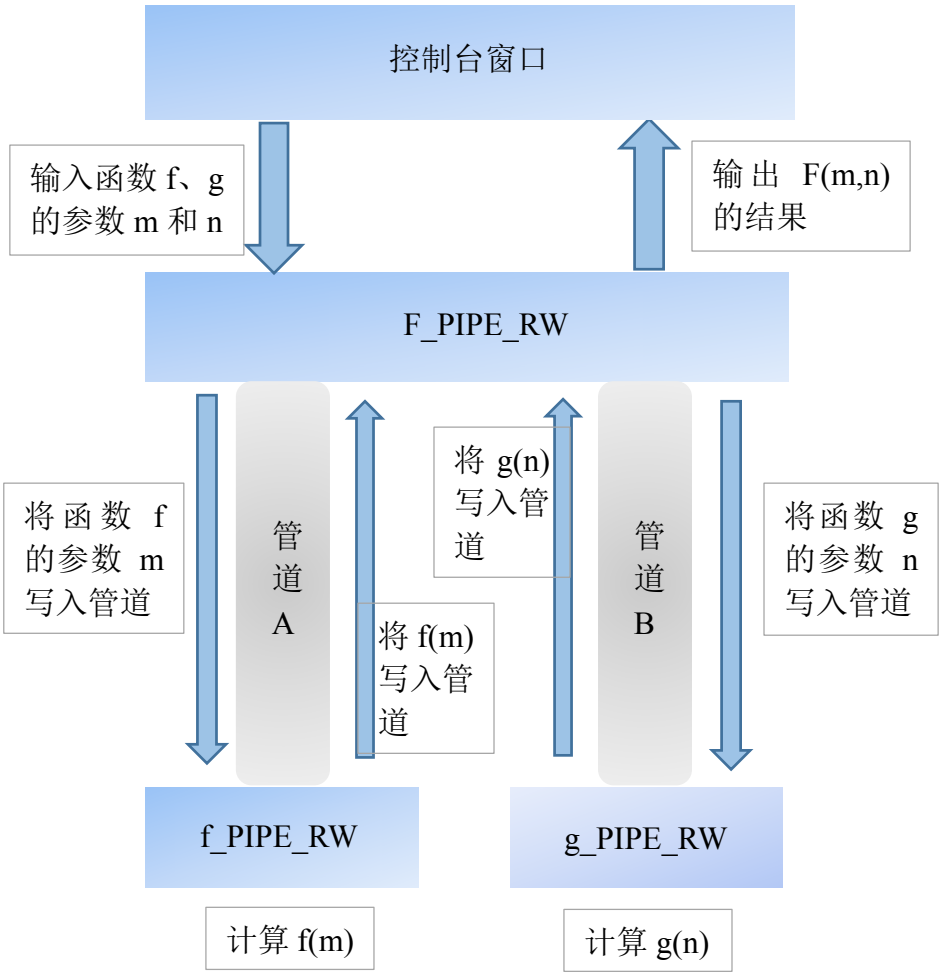
对于 f 的线程 f_thread，调用的函数为 f_PIPE_RW，该函数实现从管道 A 中读取参数 m，进行运算后将结果写回管道 A；

对于 g 的线程 g_thread，调用的函数为 g_PIPE_RW，该函数实现从管道 B 中读取参数 n，进行运算后将结果写回管道 B；

对于 F 的线程 F_thread，调用的函数为 F_PIPE_RW，该函数是本程序的核心函数，一开始向管道写入函数 f 和 g 各自的参数，最后从管道读取两个函数的返回值，相加得到最终结果，输出到控制台窗口。

因此，程序中核心的操作由函数 F_PIPE_RW 完成，通过在主函数建立调用线程 F_thread 即可实现。当线程 F_thread 结束时，函数 F(m, n)的结果已经输出。

3. 对程序结构的解释：



上图清楚展示了本程序的结构图，但没有对线程进行解说明，现补充如下：

程序中，main( )创建线程 F_thread，该线程调用了 F_PIPE_RW 函数。F_PIPE_RW 函数是程序的核心函数，问题解决的过程由该函数实现。

首先，F_PIPE_RW 从主函数 main( )中接收到输入的两个自然数，接着建立起两个管道 A 和 B，分别实现 f 与 F 之间的通信和 g 与 F 之间的通信。管道建立成功后，将 m 写管道 A，将 n 写入管道 B。再建立两个线程 f_thread 和 g_thread，分别实现 f 函数和 g 函数从管道读取数据、处理并将结果写回对应的管道。使用两个 WaitForSingleObject( )函数分别等待两个线程的结束。两线程结束表明已经将函数结果写回管道，此时再分别从管道中读取 f(m)和 g(n)，两者相加输出 F(m,n) = f(m) + g(n)结果即可。

四、 代码分析

1. 全局变量：

程序中定义的全局变量有：

```
DWORD num_of_bytes = sizeof(int);
HANDLE HREAD_fF;                    //f 和 F 之间的管道读句柄
HANDLE HWRITE_fF;                   //f 和 F 之间的管道写句柄
HANDLE HREAD_gF;                    //g 和 F 之间的管道读句柄
HANDLE HWRITE_gF;                   //g 和 F 之间的管道写句柄
```

其中 DWORD num_of_bytes 作为管道的 ReadFile 和 WriteFile 函数的参数，表示写入或读取数据的字节数。因为本程序中每次写入或读取的数均为一个 int 型变量，因而定义 num_of_bytes = sizeof(int)。两个管道的读写句柄也用全局变量，是为了减少函数之间传递参数造成不必要的麻烦。

2. 函数 f(m)和 g(n)

根据函数的表达式，使用递归的思想即可实现两个函数。

```
//f 函数(功能是计算阶乘）
int Func_f(int m)
{
    if( m == 1 )
        return 1;
    else return ( m*Func_f(m-1) );
}
//g 函数(斐波那契数列)
int Func_g(int n)
{
    if( n ==1 || n ==2 )
        return 1;
    else return (Func_g(n-1) + Func_g(n-2));
}
```

3. f_PIPE_RW 函数

f_PIPE_RW 函数实现了从管道中读取参数 m，经过运算再将结果写入管道。对 f 来说，f 向 F 传递数据相当于向管道写入数据；f 从 F 得到数据相当于从管道读入数据。

```
void WINAPI f_PIPE_RW(PVOID pvParam)        //实现 f 函数从管道读取和向管道写入数据
{
    DWORD read_dword,write_dword;          //返回实际读取或写入的字节数
    int para_m;                            //管道内的数据，即为参数 m
    while(!ReadFile(HREAD_fF,&para_m,num_of_bytes,&read_dword,NULL));
    //从管道中读取参数 m，使用 while 表示等待数据直到成功读入
    int f_result = Func_f(para_m);         //对读取的参数 m 使用 f 函数，得到结果
    while(!WriteFile(HWRITE_fF,&f_result,num_of_bytes,&write_dword,NULL));
    //将 f 函数的结果写入管道，使用 while 表示直到成功写入
}
```

4. g_PIPE_RW 函数

g_PIPE_RW 函数实现了从管道中读取参数 m，经过运算再将结果写入管道。对 g 来说，g 向 F 传递数据相当于向管道写入数据；g 从 F 得到数据相当于从管道读入数据。

```
void WINAPI g_PIPE_RW(PVOID pvParam) //实现 g 函数从管道读取和向管道写入数据
{
    DWORD read_dword,write_dword;  //返回实际读取或写入的字节数
    int para_n;                    //管道内的数据,即为参数 n
    while(!ReadFile(HREAD_gF,&para_n,num_of_bytes,&read_dword,NULL));
    //从管道中读取参数 n，使用 while 表示等待数据直到成功读入
    int g_result = Func_g(para_n);         //对读取的参数 n 使用 g 函数，得到结果
    while(!WriteFile(HWRITE_gF,&g_result,num_of_bytes,&write_dword,NULL));
    //将 g 函数的结果写入管道，使用 while 表示直到成功写入
}
```

5. F_PIPE_PW 函数

F_PIPE_RW 函数是程序的核心函数，实现了问题完整的求解过程。

a)　创建两个管道：

其中安全性参数的设置参照了课件上的代码。

```
//安全性参数设置
SECURITY_ATTRIBUTES safF,sagF;          //对应于两个管道的安全性
//安全性的参数设置,建立 f 和 F 之间的管道
safF.nLength = sizeof(SECURITY_ATTRIBUTES);
safF.lpSecurityDescriptor = NULL;
safF.bInheritHandle = TRUE;
CreatePipe(&HREAD_fF,&HWRITE_fF,&safF,0);
//安全性的参数设置,建立 g 和 F 之间的管道
sagF.nLength = sizeof(SECURITY_ATTRIBUTES);
sagF.lpSecurityDescriptor = NULL;
sagF.bInheritHandle = TRUE;
CreatePipe(&HREAD_gF,&HWRITE_gF,&sagF,0);
```

b)　将输入的 m 和 n 写入对应的管道：

核心是使用 WriteFile( )函数将 sizeof(int)字节数的数据写入管道中。

```
//将输入的两个参数写入管道
int m = para[0];
int n = para[1];
DWORD dword_fF,dword_gF;          //返回实际读入的字节数
while(!WriteFile(HWRITE_fF,&m,num_of_bytes,&dword_fF,NULL));
//F 函数向管道写入 f 函数的参数 m
while(!WriteFile(HWRITE_gF,&n,num_of_bytes,&dword_gF,NULL));
//F 函数向管道写入 g 函数的参数 n
```

c)　两个线程分别计算 f(m)和 g(n)：

使用 WaitForSingleObject( )函数等待线程结束。

```
    HANDLE f_thread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
(f_PIPE_RW),NULL,0,NULL);//建立 f 函数读写的线程
    HANDLE g_thread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
(g_PIPE_RW),NULL,0,NULL);//建立 g 函数读写的线程
    WaitForSingleObject(f_thread,INFINITE);//f_thread 线程结束时,f(m)已写入管道
    WaitForSingleObject(g_thread,INFINITE);//g_thread 线程结束时,g(n)已写入管道
```

d)　读取管道中返回的计算结果，输出最后结果：

```
    int f_result,g_result;   //分别表示 f 函数和 g 函数的返回值
    while(!ReadFile(HREAD_fF,&f_result,num_of_bytes,&dword_fF,NULL));     //F 函数
    从管道中读取 f 函数的返回值
    while(!ReadFile(HREAD_gF,&g_result,num_of_bytes,&dword_gF,NULL));     //F 函数
    从管道中读取 g 函数的返回值
    int F_result = f_result + g_result;         //F(m,n)=f(m)+g(n)
    cout<<endl<<"**************输出结果**************"<<endl
        <<"F("<<para[0]<<","<<para[1]<<")="<<F_result<<endl<<endl;
//输出最终的计算结果
```

6. 主函数建立 F_PIPE_RW 函数对应的线程：

```
    HANDLE F_thread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)
(F_PIPE_RW),&para,0,NULL);
    WaitForSingleObject(F_thread,INFINITE);
     //F 函数的线程结束时，已经输出了函数返回值
    CloseHandle(F_thread);                          //关闭线程句柄
```

以上代码通过管道实现了三个进程间的通信，最后的输出即为 F(m,n)的计算结果。

五、 程序运行结果：

以下通过若干例子测试程序的正确性。

1. m = 2, n = 3

程序运行结果：

```
********请输入两个自然数 m 和 n********
2 3

**************输出结果**************
F(2,3)=4

请按任意键继续. . .
```

验证：

f(m) = f(2) = 2! = 2；　　g(n) = g(3) = 1 + 1 = 2；

F(m,n) = F(2, 3) = 2 + 2 = 4。验证结果正确。

2. m = 10, n = 8

程序运行结果：

```
********请输入两个自然数 m 和 n********
10 8

**************输出结果**************
F(10,8)=3628821

请按任意键继续. . .
```

验证：

f(m) = f(10) = 10! = 3628800；　　g(n) = g(8) = 21；

F(m,n) = F(10, 8) = 3628800 + 21 = 3628821。验证结果正确。

同样，测试的其他例子都证明程序是正确的。

3. 另外，程序还设置了简单的输入合法性判断，当 m, n 输入为 0 0 时，程序运行结果为：

```
********请输入两个自然数 m 和 n********
0 0
Error:输入有误！请输入两个自然数!
请按任意键继续. . .
```

总之，程序的运行结果是正确的。

六、 思考题

1. 你采用了你选择的机制而不是另外的两种机制解决该问题，请解释你做出这种选择的理由。

答：

本实验提供了三种可选的高级进程间通信机制，分别为消息队列、共享内存和管道。之所以选择管道机制主要有以下原因：

a) 操作系统课程讲解了 Linux 系统下消息队列的实现方法，但是没有具体讲解 Windows 如何实现，因为我选择的是 Windows 操作系统，所以首先不选择消息队列机制；

b) 共享内存和管道的思路都比较清晰简单，但是共享内存对内存的映射、清理等感觉有些复杂，而管道的操作比较简单，很容易按照思路一步步实现；

c) 课件中只列出了共享内存常用的一些函数，但是没有给出具体的实现方法，因而还需要查阅一些课外资料；而对管道机制的具体实现则举了实例，理解了之后感觉实现起来比较简单，其中程序中关于安全性部分的参数就是参照了课件上的内容；

管道机制分为两种，包括匿名管道和命名管道。匿名管道比命名管道需要更少的开销，但是提供有限的服务，由于本题只需要来回一次数据的传递，因而对管道服务的要求不高，所以采用了容易实现、开销少的匿名管道。

2. 你认为另外的两种机制是否同样可以解决该问题？如果可以请给出你的思路；如果不能，请解释理由。

答：

除去管道之外，消息队列和共享内存机制同样可以解决进程(线程)间通信的问题。

a) **消息队列机制：**

Windows 操作系统下与 Linux 类似，也可以为当前执行的每个程序维护一个消息队列。线程将消息放入程序的消息队列中。消息队列可以看做用于存储消息的区域，不同的进程(线程)可以将消息放入消息队列或者从消息队列中取出消息。

对于本题来说，有三个并行的线程，可以将函数 f、g、F 对应的线程分别记作 f_thread、g_thread 和 F_thread，那么解决该问题的思路为：

1. F_thread 将控制台输入参数放入消息队列；
2. g_thread 和 f_thread 分别从消息队列中读取消息，获得对应的输入参数，景经过计算之后再放入消息队列；
3. F_thread 从消息队列中读取消息，即获得 f(m)和 g(n)，二者相加之后输出。

按照以上思路，通过使用针对线程句柄的 WaitForSingleObject( )函数，即可实现各进程间的同步，保证对消息队列的读写操作不会发生冲突。

b) **共享内存机制：**

共享内存使用文件映射机制，将程序中需要用到的参数数据映射到到各进程间共享的内存区域，主要的思路为：

1. 使用 CreateFile( )函数创建一个文件；
2. 针对该文件句柄使用 CreateFileMapping( )函数创建文件映射对象；
3. MapViewOfFile( )函数将文件视图映射到进程(线程)的地址空间，实现对共享内存区域的读写；
4. F 进程先将参数 m，n 写入共享内存区域，f 和 g 从共享内存区域读取 m 和 n，计算之后再将结果写入共享内存区域，F 进程再从共享内存区域读取两个计算结果，相加后输出即可；
5. 使用 UnmapViewOfFile( )函数解除文件与地址空间的映射关系，并用 CloseHandle 关闭文件映射对象。

按照以上思路，使用共享内存机制实现进程间通信的方法也比较直观，也能够解决本实验的问题。

七. 实验总结及体会

本实验主要实现了高级进程间通信的管道机制，刚看到题目是感觉很简单，不使用高级进程间通信时只需要几行代码即可解决。但是，在使用进程间通信解决问题时与实验一相同，还是只有思路却始终难以下手。在选定了使用匿名管道作为 IPC 机制之后，我翻阅了课件、教科书，同时在网上学习了一些技术博客中对管道的解释和举例，才逐渐明白了管道机制的实现方式，以及 CreatePipe()、ReadFile()、WriteFile()和 CloseHandle()等函数的使用方法。

在学习完各函数的基本使用方法之后，便按照之间的设计思路进行了代码实现。程序中一共创建了三个线程和两个管道，管道机制实现了线程间数据的传递，在创建管道时，参照了课件上提供的代码对管道相关的读写句柄、与安全性有关的参数(SECURITY_ATTRIBUTES)等进行了设置。因为在第一次实验中已经对多线程问题有了比较清晰的了解，因而实现过程相对比较顺利，进行了基本的语法上的 Debug 之后即可正常运行，这也令自己比较激动。

本次实验让我体会到 Windows 操作系统开发者在 IPC 机制上所体现出的智慧，将生活中的管道思想用到进程间通信中，不得不说还是很有创新性的。总之，通过本次实验，我对 Windows 操作系统下 IPC 机制尤其是匿名管道机制的理解又更加深了，对操作系统也有了更多的了解。