

# 《操作系统课程设计》

## 实验报告

课程名称：操作系统课程设计

课程号：304009010 课序号：0

学 期：2009 ~ 2010 学年 春季学期

任课教师：段磊

姓 名：朱名发

学 号：0843041348

实验列表			
实验名称		完成状态	PAGE
Windows 环境下	读者与写者问题	√	2-13
	内存管理问题	√	14-26
	文件系统模拟实验	√	27-47
Linux 环境下	shell 程序	√	48-72
	作业调度系统	√	73-96

## (一) 读者写者问题

实验 题目	读者-写者问题		
姓名:朱名发	学号: 0843041348	专业:计算机科学与技术	完成日期: 2010-4-28
实验内容简 要描述(实验 内容、目的、 环境、主要方 法)	<p><b>一、实验目地</b></p> <ol style="list-style-type: none"> <li>1、熟悉多线程编程;</li> <li>2、熟悉使用信号量机制解决同步问题。</li> </ol> <p><b>二、实验内容</b></p> <p>创建一个控制台进程。此进程包含 n 个线程。用这 n 个线程来表示 n 个读者或写者。每个线程按相应测试数据文件(后面有介绍)的要求进行读写操作。用信号量机制分别实现读者优先和写者优先的读者-写者问题。</p> <p>读者-写者问题的读写操作限制(包括读者优先和写者优先):</p> <ol style="list-style-type: none"> <li>1)写-写互斥,即不能有两个写者同时进行写操作。</li> <li>2)读-写互斥,即不能同时有一个线程在读,而另一个线程在写。</li> <li>3)读-读允许,即可以有一个或多个读者在读。</li> </ol> <p>读者优先的附加限制:如果一个读者申请进行读操作时已有另一个读者正在进行读操作,则该读者可直接开始读操作。</p> <p>写者优先的附加限制:如果一个读者申请进行读操作时已有另一写者在等待访问共享资源,则该读者必须等到没有写者处于等待状态后才能开始读操作。</p> <p>运行结果显示要求:要求在每个线程创建、发出读写操作申请、开始读写操作和结束读写操作时分别显示一行提示信息,以确定所有处理都遵守相应的读写操作限制。</p> <p><b>三、实验环境</b></p> <ol style="list-style-type: none"> <li>1、Windows XP 操作系统的 PC 机。</li> <li>2、Visual C++ 6.0</li> </ol> <p><b>四、实验原理</b></p> <p>可以将所有读者和所有写者分别存于一个读者等待队列和一个写者等待队列中,每当读允许时,就从读者队列中释放一个或多个读者线程进行读操作;每当写允许时,就从写者队列中释放一个写者进行写操作。</p> <p><b>1、读者优先</b></p> <p>读者优先指的是除非有写者在写文件,否则读者不需要等待。所以可以用一个整型变量 read_count 记录当前的读者数目,用于确定是否需要释放正在等待的写者线程(当 read_count=0 时,表明所有的读者读完,需要释放写者等待队列中的一个写者)。每一个读者开始读文件时,必须修改 read_count 变量。因此需要一个互斥对象 mutex 来实现对全局变量 read_count 修改时的互斥。</p> <p>另外,为了实现写-写互斥,需要增加一个临界区对象 write。当写者发出写请求时,必须申请临界区对象的所有权。通过这种方法,也可以实现读-写互斥,当 read_count=1 时(即第一个读者到来时),读者线程也必须申请临界区对象的所有权。</p> <p>当读者拥有临界区的所有权时,写者阻塞在临界区对象 write 上。当写者拥有临界区的所有权时,第一个读者判断完“read_count==1”后阻塞在 write</p>		

	<p>上，其余的读者由于等待对 read_count 的判断，阻塞在 mutex 上。</p> <p>2、写者优先</p> <p>写者优先与读者优先类似。不同之处在于一旦一个写者到来，它应该尽快对文件进行写操作，如果有一个写者在等待，则新到来的读者不允许进行读操作。为此应当添加一个整型变量 write_count，用于记录正在等待的写者的数目，当 write_count=0 时，才可以释放等待的读者线程队列。</p> <p>为了对全局变量 write_count 实现互斥，必须增加一个互斥对象 mutex3。</p> <p>为了实现写者优先，应当添加一个临界区对象 read，当有写者在写文件或等待时，读者必须阻塞在 read 上。</p> <p>读者线程除了要对全局变量 read_count 实现操作上的互斥外，还必须有一个互斥对象对阻塞 read 这一过程实现互斥。这两个互斥对象分别命名为 mutex1 和 mutex2。</p> <p><b>五、主要方法（函数）</b></p> <ul style="list-style-type: none"> <li>➤ void ReaderPriority(char *fileName); //读者优先处理函数</li> <li>➤ void RP_ReaderThread(void *p); //读者优先中的读者线程 //操作函数</li> <li>➤ void RP_WriterThread(void *p); //读者优先中的写者线程 //操作函数</li> <li>➤ void WriterPriority(char * fileName); //写者优先处理函数</li> <li>➤ void WP_ReaderThread(void *p); //写者优先中的读者线程 //处理函数</li> <li>➤ void WP_WriterThread(void *p); //写者优先中的写者线程 //处理函数</li> </ul>
解决问题所用的主要代码及描述（附注释）	<pre> #include &lt;iostream&gt; #include &lt;string.h&gt;    //字符串处理 #include &lt;windows.h&gt; #include &lt;conio.h&gt; /*conio 是 Console Input/Output（控制台输入输出）的简写，其中定义了通过控制台进行数据输入和数据输出的函数，主要是一些用户通过按键盘产生的对应操作*/ #include &lt;stdlib.h&gt;    //stdlib 头文件即 standard library 标准库头文件 #include &lt;fstream.h&gt;  //文件输入 / 输出  #define READER 'R'    //读者 #define WRITER 'W'    //写者 #define MAX_THREAD_NUM 64    //最大线程数 #define INTE_PER_SEC 1000    //每秒时钟中断的数目 #define MAX_FILE_NUM 32    //最大文件数目数 #define MAX_STR_LEN 32    //字符串的长度 int readcount=0;    //读者数目 int writecount=0;    //写者数目 CRITICAL_SECTION RP_Write;    //临界资源 CRITICAL_SECTION cs_Write; CRITICAL_SECTION cs_Read; </pre>

	<pre> struct ThreadInfo {     int serial;                                //线程序号     char entity;                               //线程类别(判断是读者还是写者 线程)     double delay;                             //线程延迟时间     double persist;                           //线程读写操作时间 }; /// ///读者优先&lt;---&gt;读者线程/// /// void RP_ReaderThread(void *p) //P:读者线程信息 {     //互斥变量     HANDLE h_Mutex;      h_Mutex=OpenMutex(MUTEX_ALL_ACCESS,FALSE,"mutex_for_readcount");     DWORD wait_for_mutex;                    //等待互斥变量所有权     DWORD m_delay;                           //延迟时间     DWORD m_persist;                         //读文件持续时间     int m_serial;                             //线程序号     // 从参数中获得信息     m_serial=((ThreadInfo*)(p))-&gt;serial ;     m_delay=(DWORD)(((ThreadInfo*)(p))-&gt;delay *INTE_PER_SEC);     m_persist=(DWORD)((((ThreadInfo*)(p))-&gt;persist *INTE_PER_SEC);     Sleep(m_delay);                          //延迟等待     cout&lt;&lt;m_serial&lt;&lt;"号读线程发送对目标文件的读取请求..."&lt;&lt;endl;     //等待互斥信号,保证对 ReadCount 的访问,修改互斥     wait_for_mutex=WaitForSingleObject(h_Mutex,-1);     //读者数目增加     readcount++;     if(readcount==1)     {         //第一个读者,等待资源         EnterCriticalSection(&amp;RP_Write);     }     ReleaseMutex(h_Mutex);                   //释放互斥信     //读文件     cout&lt;&lt;m_serial&lt;&lt;"号读线程开始读文件..."&lt;&lt;endl;     Sleep(m_persist);     //退出线程     cout&lt;&lt;m_serial&lt;&lt;"号读线程已完成读文件的操作..."&lt;&lt;endl;     //等待互斥信号,保证对 ReadCount 的访问,修改互斥     wait_for_mutex=WaitForSingleObject(h_Mutex,-1); </pre>
--	--

```
//读者数目减少
readcount--;
if(readcount==0)
{
    //如果所有的读者读完,唤醒写者
    LeaveCriticalSection(&RP_Write);
}
ReleaseMutex(h_Mutex);           //释放互斥信号
}

////////////////////////////////////
//P:写者线程信息
void RP_WriterThread(void *p)
{
    DWORD m_delay;           //延迟时间
    DWORD m_persist;         //写文件持续时间
    int m_serial;            //线程序号
    // 从参数中获得信息
    m_serial=((ThreadInfo*)(p))->serial ;
    m_delay=(DWORD)(((ThreadInfo*)(p))->delay *INTE_PER_SEC);
    m_persist=(DWORD)(((ThreadInfo*)(p))->persist *INTE_PER_SEC);
    Sleep(m_delay);
    cout<<m_serial<<"号写线程发送写请求..."<<endl;
    //等待资源
    EnterCriticalSection(&RP_Write);
    //写文件
    cout<<m_serial<<"号写线程开始写操作..."<<endl;
    Sleep(m_persist);
    //退出线程
    cout<<m_serial<<"号写线程已经完成对文件的写操作..."<<endl;
    //释放资源
    LeaveCriticalSection(&RP_Write);
}

////////////////////////////////////
////////////////////////////////////读者优先处理函数////////////////////////////////////
////////////////////////////////////
//fileName:文件名
void ReaderPriority(char *fileName)
{
    DWORD n_thread=0;        //线程数目
    DWORD thread_ID;         //线程 ID
    DWORD wait_for_all;       //等待所有线程结束
    //互斥对象
    HANDLE h_Mutex;
```

	<pre> h_Mutex=CreateMutex(NULL,FALSE,"mutex_for_readcount"); //线程对象的数组 HANDLE h_Thread[MAX_THREAD_NUM]; ThreadInfo thread_info[MAX_THREAD_NUM]; readcount=0;                //初始化 readcount InitializeCriticalSection(&amp;RP_Write);        //初始化临界区 ifstream inFile; inFile.open (fileName); if(!inFile) {     cout&lt;&lt;"文件打开失败!"&lt;&lt;endl; } cout&lt;&lt;"读者优先:"&lt;&lt;endl; while(inFile) {     //读入每一个读者,写者的信息     inFile&gt;&gt;thread_info[n_thread].serial;     inFile&gt;&gt;thread_info[n_thread].entity;     inFile&gt;&gt;thread_info[n_thread].delay;     inFile&gt;&gt;thread_info[n_thread++].persist;     inFile.get(); } for(int i=0;i&lt;(int)(n_thread);i++) {     if(thread_info[i].entity==READER    thread_info[i].entity == 'r')     {         //创建读者进程  h_Thread[i]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)(RP_ReaderThread), &amp;thread_info[i],0,&amp;thread_ID);     }     else     {         //创建写线程  h_Thread[i]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)(RP_WriterThread), &amp;thread_info[i],0,&amp;thread_ID);     } }  //等待所有的线程结束 wait_for_all=WaitForMultipleObjects(n_thread,h_Thread,TRUE,-1); </pre>
--	--

	<pre> cout&lt;&lt;"已经完成了所有的读写操作!"&lt;&lt;endl; }  //////////////////////////////////// //写者优先---读者线程 //P:读者线程信息 void WP_ReaderThread(void *p) {     //互斥变量     HANDLE h_Mutex1;     h_Mutex1=OpenMutex(MUTEX_ALL_ACCESS,FALSE,"mutex1");     HANDLE h_Mutex2;     h_Mutex2=OpenMutex(MUTEX_ALL_ACCESS,FALSE,"mutex2");     DWORD wait_for_mutex1;           //等待互斥变量所有权     DWORD wait_for_mutex2;     DWORD m_delay;                   //延迟时间     DWORD m_persist;                 //读文件持续时间     int m_serial;                     //线程的序号     //从参数中得到信息     m_serial=((ThreadInfo*)(p))-&gt;serial ;     m_delay=(DWORD)((((ThreadInfo*)(p))-&gt;delay *INTE_PER_SEC);     m_persist=(DWORD)((((ThreadInfo*)(p))-&gt;persist *INTE_PER_SEC);     Sleep(m_delay);                  //延迟等待     cout&lt;&lt;m_serial&lt;&lt;"号读线程发送读请求..."&lt;&lt;endl;     wait_for_mutex1=WaitForSingleObject(h_Mutex1,-1);     //读者进去临界区     EnterCriticalSection(&amp;cs_Read);     //阻塞互斥对象 Mutex2,保证对 readCount 的访问和修改互斥     wait_for_mutex2=WaitForSingleObject(h_Mutex2,-1);     //修改读者的数目     readcount++;     if(readcount==1)     {         // 如果是第 1 个读者,等待写者写完         EnterCriticalSection(&amp;cs_Write);     }     ReleaseMutex(h_Mutex2);// 释放互斥信号 Mutex2     //让其他读者进去临界区     LeaveCriticalSection(&amp;cs_Read);     ReleaseMutex(h_Mutex1);     //读文件     cout&lt;&lt;m_serial&lt;&lt;"号读线程开始读文件..."&lt;&lt;endl;     Sleep(m_persist);     //退出线程 </pre>
--	---

```

        cout<<m_serial<<"号读线程已经完成了读操作..."<<endl;
        //阻塞互斥对象 Mutex2,保证对 readcount 的访问,修改互斥
        wait_for_mutex2=WaitForSingleObject(h_Mutex2,-1);
        readcount--;
        if(readcount==0)
        {
            //最后一个读者,唤醒写者
            LeaveCriticalSection(&cs_Write);
        }
        ReleaseMutex(h_Mutex2); //释放互斥信号
    }

    ////////////////////////////////////////
    //写者优先---写者线程
    //P:写者线程信息
    void WP_WriterThread(void *p)
    {
        DWORD wait_for_mutex3;           //互斥变量
        DWORD m_delay;                    //延迟时间
        DWORD m_persist;                  //读文件持续时间
        int m_serial;                      //线程序号
        HANDLE h_Mutex3;
        h_Mutex3=OpenMutex(MUTEX_ALL_ACCESS,FALSE,"mutex3");
        //从参数中获得信息
        m_serial=((ThreadInfo*)(p))->serial ;
        m_delay=(DWORD)(((ThreadInfo*)(p))->delay *INTE_PER_SEC);
        m_persist=(DWORD)(((ThreadInfo*)(p))->persist *INTE_PER_SEC);
        Sleep(m_delay);                    //延迟等待
        cout<<m_serial<<"号写线程发送写请求..."<<endl;
        wait_for_mutex3=WaitForSingleObject(h_Mutex3,-1);
        writecount++;                      //修改写者数目
        if(writecount==1)
        {
            EnterCriticalSection(&cs_Read);
        }
        ReleaseMutex(h_Mutex3);
        EnterCriticalSection(&cs_Write);
        cout<<m_serial<<"号写线程开始对文件进行写操作..."<<endl;
        Sleep(m_persist);
        cout<<m_serial<<"号写线程已完成对文件的写操作..."<<endl;
        LeaveCriticalSection(&cs_Write);
        wait_for_mutex3=WaitForSingleObject(h_Mutex3,-1);
        writecount--;
        if(writecount==0)
    
```



```

        {
            LeaveCriticalSection(&cs_Read);
        }
        ReleaseMutex(h_Mutex3);
    }

    //////////////////////////////////////
    //写者优先处理函数
    // fileName:文件名
    void WriterPriority(char * fileName)
    {
        DWORD n_thread=0;
        DWORD thread_ID;
        DWORD wait_for_all;
        HANDLE h_Mutex1;
        h_Mutex1=CreateMutex(NULL,FALSE,"mutex1");
        HANDLE h_Mutex2;
        h_Mutex2=CreateMutex(NULL,FALSE,"mutex2");
        HANDLE h_Mutex3;
        h_Mutex3=CreateMutex(NULL,FALSE,"mutex3");
        HANDLE h_Thread[MAX_THREAD_NUM];
        ThreadInfo thread_info[MAX_THREAD_NUM];
        readcount=0;
        writecount=0;
        InitializeCriticalSection(&cs_Write);
        InitializeCriticalSection(&cs_Read);
        ifstream inFile;
        inFile.open (fileName);
        cout<<"写者优先:"<<endl;
        while(inFile)
        {
            inFile>>thread_info[n_thread].serial;
            inFile>>thread_info[n_thread].entity;
            inFile>>thread_info[n_thread].delay;
            inFile>>thread_info[n_thread++].persist;
            inFile.get();
        }
        for(int i=0;i<(int)(n_thread);i++)
        {
            if(thread_info[i].entity==READER || thread_info[i].entity == 'r')
            {
                //创建读者进程

                h_Thread[i]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)(WP_ReaderThr
    
```

```

ead),
&thread_info[i],0,&thread_ID);
    }
    else
    {
        //创建写线程

h_Thread[i]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)(WP_WriterThre
ad),
&thread_info[i],0,&thread_ID);
    }
}

//等待所有的线程结束
wait_for_all=WaitForMultipleObjects(n_thread,h_Thread,TRUE,-1);
cout<<"已完成所有的读写操作。。" <<endl;
}
////////////////////////////////////
//主函数
int main(void)
{
    char ch;
    while(1)
    {
        cout<<"-----读者与写者
问题-----" <<endl;

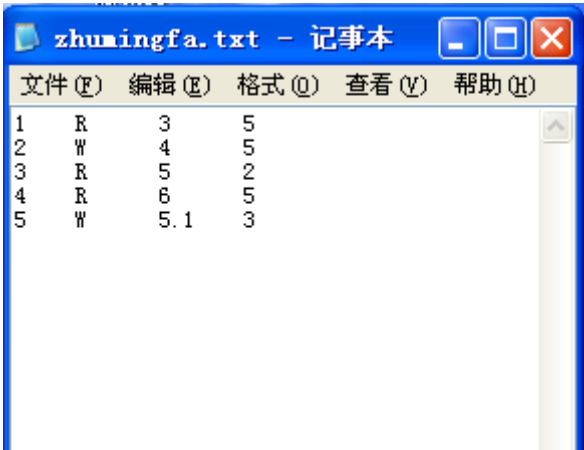
cout<<"*****
*****" <<endl;

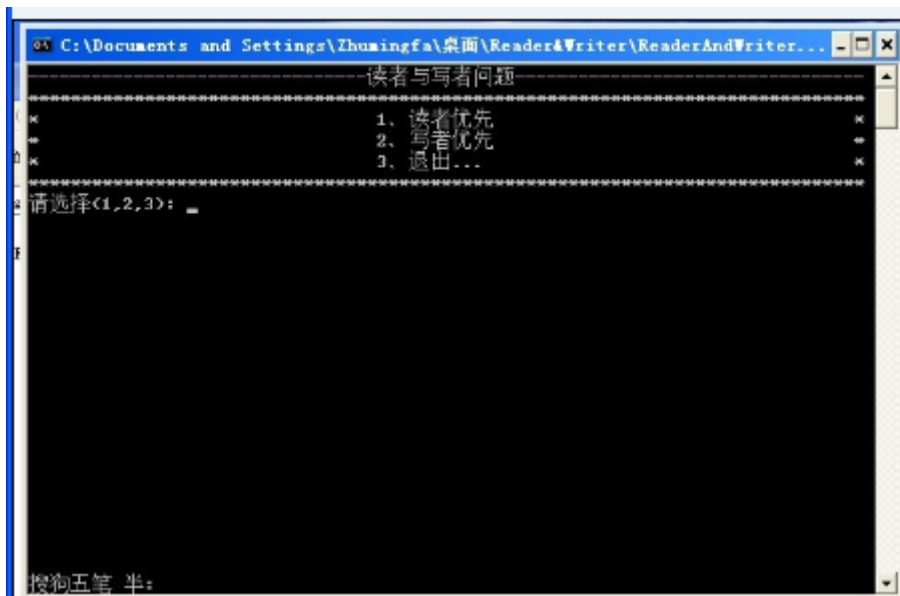
        cout<<"*
1、读者优先                                *" <<endl;
        cout<<"*
2、写者优先                                *" <<endl;
        cout<<"*
3、退出...                                *" <<endl;

cout<<"*****
*
*****" <<endl;

        printf("请选择(1,2,3): ");
        do{
            ch=(char)getch();
        }while(ch!='1'&&ch!='2'&&ch!='3');
        system("cls"); //执行清屏命令
    }
}

```

	<pre> switch(ch) { case '1':     ReaderPriority("zhumingfa.txt");     break; case '2':     WriterPriority("zhumingfa.txt");     break; case '3':     return 0; } cout&lt;&lt;endl&lt;&lt;"按任意键继续..."&lt;&lt;endl; getch(); system("cls"); } return 0; } </pre>
<p>实验结果分析(或错误原因分析)</p>	<p>一、测试数据</p>  <p>二、程序界面</p>



### 三、读者优先



	<p>四、写者优先</p>  <pre> C:\Documents and Settings\Zhumingfa\桌面\F 写者优先: 1号读线程发送读请求... 1号读线程开始读文件... 2号写线程发送写请求... 3号读线程发送读请求... 5号写线程发送写请求... 4号读线程发送读请求... 1号读线程已经完成了读操作... 2号写线程开始对文件进行写操作... 2号写线程已完成对文件的写操作... 5号写线程开始对文件进行写操作... 5号写线程已完成对文件的写操作... 3号读线程开始读文件... 4号读线程开始读文件... 3号读线程已经完成了读操作... 4号读线程已经完成了读操作... 已完成所有的读写操作。。。  按任意键继续... </pre>
小结	<p>本实验测试结果与理论相符，经过多组测试数据的检验尚未发现逻辑错误。通过本实验加深了对多线程编程思想及信号量机制解决同步问题方法的理解。</p>
指导老师意见	

## (二) 内存管理问题

实验 题目	内存管理问题		
姓名:朱名发	学号: 0843041348	专业:计算机科学与技术	完成日期: 2010-5-20
实验内容简要描述(实验内容、目的、环境、主要方法)	<p><b>一、实验目地</b></p> <ol style="list-style-type: none"> <li>1、从不同侧面了解 windows2000/xp 对用户进程的虚拟内存空间的管理与分配的方法;</li> <li>2、了解跟踪程序的编写方法 (与被跟踪程序保持同步, 使用 windows 提供的信号量机制)。</li> <li>3、对 windows 分配虚拟内存、改变内存状态以及对物理内存各页面文件状态查询的 API 函数的功能、参数限制及其使用规则等。</li> </ol> <p><b>二、实验内容</b></p> <p>使用 Windows 2000 / XP 的 API 函数, 编写一个包含两个线程的进程, 一个线程用于模拟内存分配活动, 一个线程用于跟踪第一个线程的内存行为, 而且要求两个线程之间通过信号量实现同步。模拟内存活动的线程可以从一个文件中读出要进行的内存操作。内存操作包括保留(reserve)一个区域、提交(commit)一个区域、释放(release)一个区域、回收(decommit)一个区域和加锁(lock)与解锁(unlock)一个区域, 可以将这些操作编号存放于文件。</p> <p><b>三、实验环境</b></p> <ol style="list-style-type: none"> <li>1、Windows XP 操作系统的 PC 机。</li> <li>2、Visual C++ 6.0</li> </ol> <p><b>四、实验要求</b></p> <p>本实验应包含以下源程序:</p> <ul style="list-style-type: none"> <li>➤ <b>Makefile.cpp:</b> 主要实现将操作写入文件, 采用了 c 语言的 fwrite 函数直接以结构(struct)为单位写入文件。采用两层循环, 外层循环控制对内存的操作(保留、提交、锁、解锁、回收、释放), 内层循环控制对内存操作的权限, 用随机数生成等待执行的时间和分配的粒度</li> <li>➤ <b>Memory-op.cpp :</b> <ol style="list-style-type: none"> <li>(1)主函数: 创建两个线程, 并将返回的句柄存入数组中。 创建两个信号量(allo, trac)分别用于通知跟踪线程和记录线程。 用函数 WaitForMultipleObjects 来等待两个线程的结束。</li> <li>(2)Tracker 线程(记录内存的状况)</li> <li>(3)Allocator 线程(模拟内存分配活动)</li> </ol> </li> </ul> <p><b>四、实验原理</b></p> <ul style="list-style-type: none"> <li>➤ 使用 windows xp 的 API 函数, 编写一个包含两个线程的进程, 一个线程用于模拟内存分配活动, 一个线程用于跟踪第一个线程的内存行为 (要求两线程之间通过信号量实现线程同步)。</li> <li>➤ 模拟内存活动的线程可以从一个文件中读出要进行的内存操作, 其</li> </ul>		

中，每个内存操作均包括如下内容：

时间：操作等待时间。

块数：分配内存的粒度。

操作：包括保留（reserve）一个区域、提交（commit）一个区域、释放（release）一个区域、回收（decommit）一个区域和加锁（lock）与解锁（unlock）一个区域，可以将这些操作编号存放于文件中。

- 保留：是指保留进程的虚拟地址空间，而不是分配物理存储空间。
- 提交：是指在内存中分配物理存储空间。
- 回收：是指释放物理内存空间，但在虚拟地址空间仍然保留，它与提交相对应，即可以回收已经提交的内存块。
- 释放：是指将物理存储和虚拟地址空间全部释放，它与保留相对应，即可以释放已经保留的内存块。
- 大小：块的大小。
- 访问权限：

PAGE\_READONLY-----使被提交页面可读

PAGE\_READWRITE-----使被提交页面可读写

PAGE\_EXECUTE-----使被提交页面可执行

PAGE\_EXECUTE\_READ-----使被提交页面可执行且可读

PAGE\_EXECUTE\_READWRITE-使被提交页面可执行可读写

- 默认情况下，32 位 windows xp 上每个用户进程可以占有 2GB 的私有地址空间，操作系统占有剩下的 2GB.
- Windows xp 在 x86 体系结构上利用二级页表结构来实现虚拟地址向物理地址的变换，一个 32 位虚拟地址被解释为三个独立的分量——页目录索引、页表索引和字节索引——它们用于找出描述页面映射结构的索引。页面大小及页表项的宽度决定了页目录和页表索引的宽度。
- Windows 进程的虚拟地址空间中也有三种状态的页面：空闲页面、保留页面以及提交页面。
 

**空闲（Free）页面：**指那些可以保留或提交的可用页面。

**保留（Reserved）页面：**指逻辑页面已分配但没有分配物理存储的页面。

目的：可以保留一部分虚拟地址，这样，如果不预先释放这些地址，就不能被其它应用程序（如：Malloc, LocalAlloc 等）的操作所使用。试图读或写空闲页面或保留页面将导致页面出错异常，保留页面可被释放或提交。

**提交（Committed）页面：**物理存储（内存或磁盘上）已被分配的页面。

## 五、主要方法（函数）

### 1) GetSystemInfo

- 函数功能：该函数返回当前系统的信息。
- 函数原型：void GetSystemInfo (LPSYSTEM\_INFO lpSystemInfo);  
参数：lpSystemInfo：指向 SYSTEM\_INFO 结构的指针，SYSTEM\_INFO 结构由这个函数填充。 返回值：该函数没有返回值。

	<p><b>2) GlobalMemoryStatus</b></p> <ul style="list-style-type: none"> <li>➤ <b>函数功能：</b>该函数可以获得计算机系统中当前使用的物理内存和虚拟内存的信息。</li> <li>➤ <b>函数原型：</b><code>void GlobalMemoryStatus(LPMEMORYSTATUS lpBuffer)</code> 参数: <code>lpBuffer</code>: 指向 <code>MEMORYSTATUS</code> 数据结构。<code>GlobalMemoryStatus</code> 函数将内存的当前信息存储在该结构中。</li> </ul> <p><b>3) VirtualAlloc</b></p> <ul style="list-style-type: none"> <li>➤ <b>函数功能：</b>该函数可以在调用进程的虚拟地址中保留或提交页面。除非设置了 <code>MEM—RESET</code> 标志，否则被这个函数分配的内存单元被自动初始化为 0。</li> <li>➤ <b>函数原型：</b><code>LPVOID VirtualAlloc(LPVOID lpAddress, DWORD dwSize, DWORD flAllocationType, DWORD flProtect)</code>; <code>lpAddress</code>: 待分配空间的起始地址。如果指定的内存被保留, 指定的地址将四舍五入到下一个 64K 边界。如果指定的内存已经被保留并且被提交, 该地址将四舍五入到下一个页面边界 <code>dwSize</code>: 定义分配空间的大小(以字节为单位) <code>flAllocationType</code>: 定义分配类型, 可以使用以下标志的任何组合定义分配类型: <code>MEM_COMMIT</code>: 在内存或磁盘页面文件中分配物理存储空间。 <code>MEM_RESERVE</code>: 保留进程的虚拟地址空间, 而不分配物理空间。 <code>flProtect</code>: 指定存取保护位的类型。 <code>PAGE_READONLY</code> : <code>PAGE_READWRITE</code> : <code>PAGE_EXECUTE</code> : <code>PAGE_EXECUTE_READ</code> : <code>PAGE_EXECUTE_READWRITE</code> : <code>PAGE_GUARD</code>: 保护标志, 可以建立保护页面 返回值: 如果函数成功, 返回值为所分配的页面的基址; 如果函数失败, 返回值为 <code>NULL</code>。</li> </ul> <p><b>4) VirtualFree</b></p> <ul style="list-style-type: none"> <li>➤ <b>函数功能：</b>可以释放或注销调用进程虚拟空间中的页面。</li> <li>➤ <b>函数原型：</b><code>BOOL VirtualFree(LPVOID lpAddress, DWORD dwSize, DWORD dwFreeType)</code>; <code>lpAddress</code>: 指向释放页面的基址 <code>dwSize</code>: 定义释放区域的大小(以字节位单位) <code>dwFreeType</code>: 定义释放类型, 只能设置以下标志中的一个: <code>MEM_DECOMMIT</code>: 注销提交页面, 即释放物理内存空间, 但在虚拟地址空间仍然保留 <code>MEM_RELEASE</code>: 释放备用页面, 即将物理存储和虚拟地址空间全部释放 为了释放一片页面区域, 要保证该范围内的所有页面处于相同状态(或者都是备用页面或者都是提交页面); 如果原来保留的页面只有一部分被提交过, 你必须首先调用 <code>VirtualFree</code> 函数注销这些提交的页面, 然后再调用 <code>VirtualFree</code> 函数释放它们</li> </ul>
--	---



	<p>5) VirtualLock</p> <ul style="list-style-type: none"> <li>➤ <b>函数功能:</b> 该函数可以将进程虚拟空间中的内存加锁。以确保后面的对该区域的存取操作不会产生页面失败。</li> <li>➤ <b>函数原型:</b> BOOL VirtualLock(LPVOID lpAddress, DWORD dwSize); 参数: lpAddress: 指向加锁页面区域基址的指针。 dwSize: 定义加锁区域的大小(以字节为单位)。有效页面的范围为从 lpAddress 到 lpAddress+dwsize。这意味着, 当一个 2 字节区域跨过一个页面边界时, 将两个页面加锁。 返回值: 如果函数成功, 返回一个非零值; 如果函数失败, 返回值为 0。</li> </ul> <p>6) VirtualUnlock</p> <ul style="list-style-type: none"> <li>➤ <b>函数功能:</b> 该函数可以将进程虚拟空间指定范围内的页面解锁, 从而系统在必要时可以将这些页面换出到页面文件中。</li> <li>➤ <b>函数原型:</b> BOOL VirtualUnlock(LPVOID lpAddress, DWORD dwSize); 参数: lpAddress: 指向解锁页面区域基址的指针。 dwSize: 定义解锁区域的大小(以字节为单位)。有效页面的范围为 lpAddress 到 lpAddress+dwSize。这意味着, 当一个 2 字节区域跨过一个页面边界时, 将相邻的两个页面解锁。 返回值: 如果函数成功, 返回一个非零值; 如果函数失败, 返回值为 0。</li> </ul>
<p>解决问题所用的主要代码及描述 (附注释)</p>	<p>(一) Makfile.cpp</p> <pre> //////////////////////////////////// //文件名: Makefile.cpp //程序功能: 文件生成程序 //作者: 朱名发 //当前版本: V1.0 //时间: 2010-5-13 //////////////////////////////////// #include&lt;stdio&gt; #include&lt;stdlib&gt; #include&lt;fstream.h&gt; #include&lt;ctime&gt;  //定义内存操作结构体 struct operation {     int time; //起始时间     int block; //内存页数     int oper; //用来记录内存操作类型     int protection; //操作权限标志 };  int main(void) </pre>

```
{
    FILE *file;
    if(NULL==(file=fopen("opfile","wb"))){/"opfile"为二进制文件用以确定内存
操作
    {
        printf("\n 打开文件 opfile 出错!");
        getchar();
        exit(1);
    }
    operation op;
    for(int j=0;j<6;j++) //0-保留;1-提交;2-锁;3-解锁;4-回收;5-释放
    {
        for(int i=0;i<5;i++)
            //0-PAGE_READONLY;
            //1-PAGE_READWRITE;
            //2-PAGE_EXECUTE;
            //3-PAGE_EXECUTE_READ;
            //4-PAGE_EXECUTE_READWRITE;
        {
            op.time=rand()%1000;//调用随机函数, 生成等待执行的时间
            op.block=rand()%5+1;//调用随机函数, 生成分配的粒度
            op.oper=j;
            op.protection=i;
            fwrite(&op,sizeof(operation),1,file);//将生成的结构写入文件中
        }
    }
    return 0;
}
```

## (二) Memory-op.cpp

```
////////////////////////////////////
//文件名: Memory-op.cpp
//主要功能: 实现内存管理
//作者: 朱名发
//当前版本: V1.0
//时间: 2010-5-13
////////////////////////////////////
#include<windows.h>
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<fstream.h>
```

```
//定义内存操作结构体
struct operation
{
    int time; //起始时间
    int block; //内存页数
    int oper; //用来记录内存操作类型
    int protection; //操作权限标志
};

//构建结构体, 用来跟踪每一次分配活动
struct trace
{
    LPVOID start; //起始地址
    long size; //分配的大小
};

HANDLE allo, trac; //信号量句柄
DWORD Tracker(LPDWORD lpdwparm) //跟踪 allocator 线程的内存行为, 并输出必要信息
{
    ofstream outfile; //输出文件
    outfile.open("out.txt");
    for(int i=0; i<30; i++)
    {
        WaitForSingleObject(trac, INFINITE); //等待 allocator 一次内存分配活动结束

        //将内存和系统状况打印出来
        outfile<<i<<<endl;

        //下面一段显示系统信息, 每次执行操作后系统信息不变
        //查看系统信息
        SYSTEM_INFO info; //系统信息
        GetSystemInfo(&info);
        outfile<<"-----系统信息-----"<<endl;

        outfile<<"dwActiveProcessorMask"<<"\t"<<info.dwActiveProcessorMask<<endl;

        outfile<<"dwAllocationGranularity"<<"\t"<<info.dwAllocationGranularity<<endl;

        outfile<<"dwNumberOfProcessors"<<"\t"<<info.dwNumberOfProcessors<<endl;

        outfile<<"dwOemId"<<"\t"<<info.dwOemId<<endl;
    }
}
```

	<pre> outfile&lt;&lt;"dwPageSize"&lt;&lt;"\t"&lt;&lt;info.dwPageSize&lt;&lt;endl; outfile&lt;&lt;"dwProcessorType"&lt;&lt;"\t"&lt;&lt;info.dwProcessorType&lt;&lt;endl;  outfile&lt;&lt;"lpMaximumApplicationAddress"&lt;&lt;"\t"&lt;&lt;info.lpMaximumApplicati onAddress&lt;&lt;endl;  outfile&lt;&lt;"lpMinimumApplicationAddress"&lt;&lt;"\t"&lt;&lt;info.lpMinimumApplicati onAddress&lt;&lt;endl;  outfile&lt;&lt;"wProcessorArchitecture"&lt;&lt;"\t"&lt;&lt;info.wProcessorArchitecture&lt;&lt;e ndl;  outfile&lt;&lt;"wProcessorLevel"&lt;&lt;"\t"&lt;&lt;info.wProcessorLevel&lt;&lt;endl; outfile&lt;&lt;"wProcessorRevision"&lt;&lt;"\t"&lt;&lt;info.wProcessorRevision&lt;&lt;endl; outfile&lt;&lt;"wReserved"&lt;&lt;"\t"&lt;&lt;info.wReserved&lt;&lt;endl; outfile&lt;&lt;"-----"&lt;&lt;endl;  //内存状况 MEMORYSTATUS status; GlobalMemoryStatus(&amp;status); outfile&lt;&lt;"-----内存状况-----"&lt;&lt;endl; outfile&lt;&lt;"dwAvailPageFile"&lt;&lt;"\t"&lt;&lt;status.dwAvailPageFile&lt;&lt;endl; outfile&lt;&lt;"dwAvailPhys"&lt;&lt;"\t"&lt;&lt;status.dwAvailPhys&lt;&lt;endl; outfile&lt;&lt;"dwAvailVirtual"&lt;&lt;"\t"&lt;&lt;status.dwAvailVirtual&lt;&lt;endl; outfile&lt;&lt;"dwLength"&lt;&lt;"\t"&lt;&lt;status.dwLength&lt;&lt;endl; outfile&lt;&lt;"dwMemoryLoad"&lt;&lt;"\t"&lt;&lt;status.dwMemoryLoad&lt;&lt;endl; outfile&lt;&lt;"dwTotalPageFile"&lt;&lt;"\t"&lt;&lt;status.dwTotalPageFile&lt;&lt;endl; outfile&lt;&lt;"dwTotalPhy"&lt;&lt;"\t"&lt;&lt;status.dwTotalPhys&lt;&lt;endl; outfile&lt;&lt;"dwTotalVirtual"&lt;&lt;"\t"&lt;&lt;status.dwTotalVirtual&lt;&lt;endl; outfile&lt;&lt;"-----"&lt;&lt;endl; //以下一段显示内存基本信息，每次操作后内存基本信息不变 // MEMORY_BASIC_INFORMATION mem;//内存基本信息  VirtualQuery(info.lpMinimumApplicationAddress,&amp;mem,sizeof(MEMORY_B ASIC_INFORMATION)); outfile&lt;&lt;"-----内存基本信息-----"&lt;&lt;endl; outfile&lt;&lt;"AllocationBase"&lt;&lt;"\t"&lt;&lt;mem.AllocationBase&lt;&lt;endl; outfile&lt;&lt;"AllocationProtect"&lt;&lt;"\t"&lt;&lt;mem.AllocationProtect&lt;&lt;endl; outfile&lt;&lt;"BaseAddress"&lt;&lt;"\t"&lt;&lt;mem.BaseAddress&lt;&lt;endl; outfile&lt;&lt;"Protect"&lt;&lt;"\t"&lt;&lt;mem.Protect&lt;&lt;endl; outfile&lt;&lt;"RegionSize"&lt;&lt;"\t"&lt;&lt;mem.RegionSize&lt;&lt;endl; outfile&lt;&lt;"State"&lt;&lt;"\t"&lt;&lt;mem.State&lt;&lt;endl; outfile&lt;&lt;"Type"&lt;&lt;"\t"&lt;&lt;mem.Type&lt;&lt;endl; outfile&lt;&lt;"-----"&lt;&lt;endl; </pre>
--	---

	<pre> //释放信号量, 并通知 allocator 可以执行下一次内存分配活动 ReleaseSemaphore(allo,1,NULL);  }  return 0;  }  //模拟内存分配活动的线程 void Allocator() {     trace traceArray[5];     int index=0;     FILE *file;     if(NULL==(file=fopen("opfile","rb")))//读入文件     {         printf("文件 opfile 打开失败! \n");         getchar();         exit(1);     }     operation op;     SYSTEM_INFO info;     DWORD temp;     GetSystemInfo(&amp;info);     for(int i=0;i&lt;30;i++)     {         WaitForSingleObject(allo,INFINITE);//等待 tracker 打印结束的信号量         cout&lt;&lt;i&lt;&lt;" ";         fread(&amp;op,sizeof(operation),1,file);         Sleep(op.time);//执行时间, 如果想在指定时间执行可以取消注释         GetSystemInfo(&amp;info);         switch(op.protection)//设置操作权限         {             case 0:             {                 index=0;                 temp=PAGE_READONLY;                 break;             }             case 1:                 temp=PAGE_READWRITE;                 break;             case 2:                 temp=PAGE_EXECUTE;                 break;             case 3: </pre>
--	--

	<pre> temp=PAGE_EXECUTE_READ; break; case 4: temp=PAGE_EXECUTE_READWRITE; break; default: temp=PAGE_READONLY; } switch(op.oper)//操作一个区域 { case 0://保留一个区域 { cout&lt;&lt;"保留区域"&lt;&lt;endl; traceArray[index].start=VirtualAlloc(NULL,op.block *info.dwPageSize,MEM_RESERVE,PAGE_NOACCESS); traceArray[index++].size=op.block *info.dwPageSize; cout&lt;&lt;"起始地址:"&lt;&lt;traceArray[index-1].start&lt;&lt;"\t" &lt;&lt;"长度:"&lt;&lt;traceArray[index-1].size&lt;&lt;endl; break; } case 1://提交一个区域 { cout&lt;&lt;"提交区域"&lt;&lt;endl;  traceArray[index].start=VirtualAlloc(traceArray[index].start,traceArray[index ].size,MEM_COMMIT,temp); index++; cout&lt;&lt;" 起始 地 址 : "&lt;&lt;traceArray[index-1].start&lt;&lt;"\t"&lt;&lt;" 长 度:" &lt;&lt;traceArray[index-1].size&lt;&lt;endl; break; } case 2://锁一个区域 { cout&lt;&lt;"锁住区域"&lt;&lt;endl; cout&lt;&lt;" 起 始 地 址 : "&lt;&lt;traceArray[index].start&lt;&lt;"\t"&lt;&lt;" 长 度:"&lt;&lt;traceArray[index].size&lt;&lt;endl;  if(!VirtualLock(traceArray[index].start,traceArray[index++].size)) { cout&lt;&lt;GetLastError()&lt;&lt;endl;//GetLastError() 函数 返 回 错误号 } break; </pre>
--	--

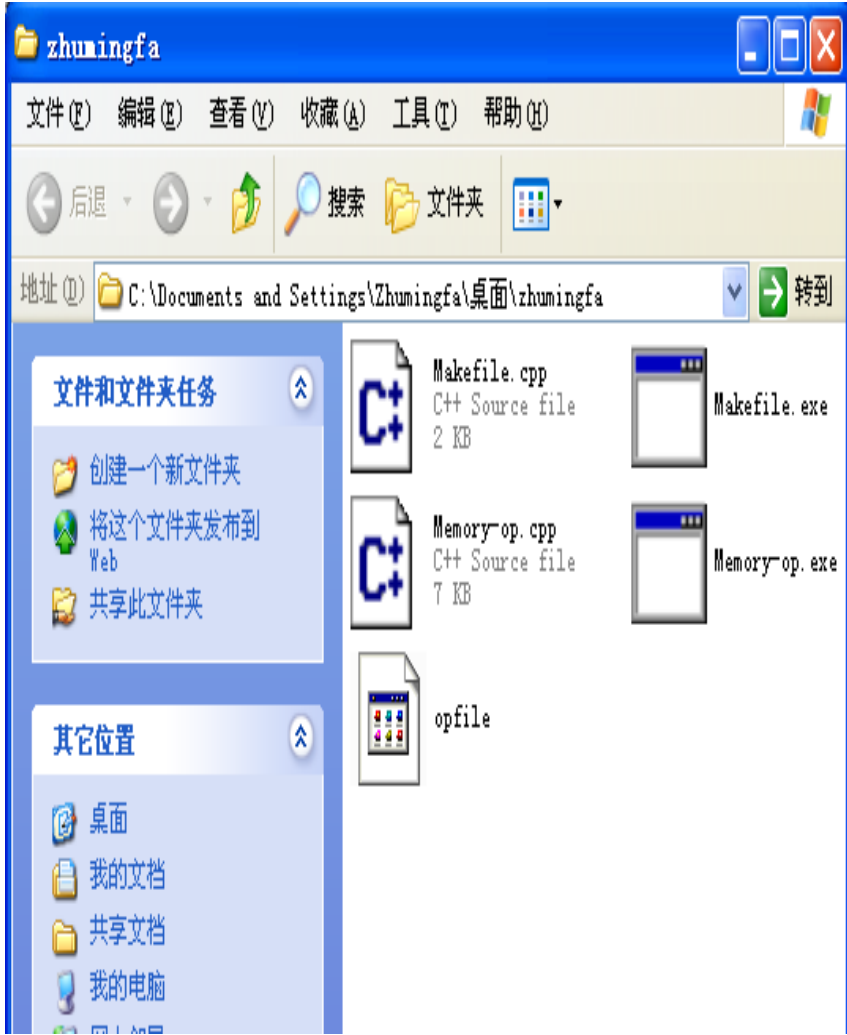
```

    }
    case 3://解锁一个区域
    {
        cout<<"解锁区域"<<endl;
        cout<<" 起 始 地 址 : "<<traceArray[index].start<<"\t"<<" 长
度:"<<traceArray[index].size<<endl;

        if(!VirtualUnlock(traceArray[index].start,traceArray[index++].size))
        {
            cout<<GetLastError()<<endl;
        }
        break;
    }
    case 4://回收一个区域
    {
        cout<<"回收区域"<<endl;
        cout<<" 起 始 地 址 : "<<traceArray[index].start<<"\t"<<" 长
度:"<<traceArray[index].size<<endl;

        if(!VirtualFree(traceArray[index].start,traceArray[index++].size,MEM_DECO
MMIT))
        {
            cout<<GetLastError()<<endl;
        }
        break;
    }
    case 5://释放一个区域
    {
        cout<<"释放区域"<<endl;
        cout<<" 起 始 地 址 : "<<traceArray[index].start<<"\t"<<" 长
度:"<<traceArray[index].size<<endl;
        if(!VirtualFree(traceArray[index++].start,0,MEM_RELEASE))
        {
            cout<<GetLastError()<<endl;
        }
        break;
    }
    default:
        cout<<"出错!"<<endl;
    }
    ReleaseSemaphore(trac,1,NULL);//释放信号量,并通知 tracker 可以打
印信息
    }
}

```

	<pre> int main(void) {     DWORD dwThread;     HANDLE handle[2];     //生成两个线程     handle[0]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)Tracker,NULL, 0,&amp;dwThread);     handle[1]=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)Allocator,NUL L,0,&amp;dwThread);     //生成两个信号量     allo=CreateSemaphore(NULL,0,1,"allo");     trac=CreateSemaphore(NULL,1,1,"trac");     //等待线程执行的执行结束后，现退出     WaitForMultipleObjects(2,handle,TRUE,INFINITE);     return 0; }         </pre>
<p>实验结果分析(或错误原因分析)</p>	<p>一、运行 Makefile.exe 产生二进制文件 opfile</p> 

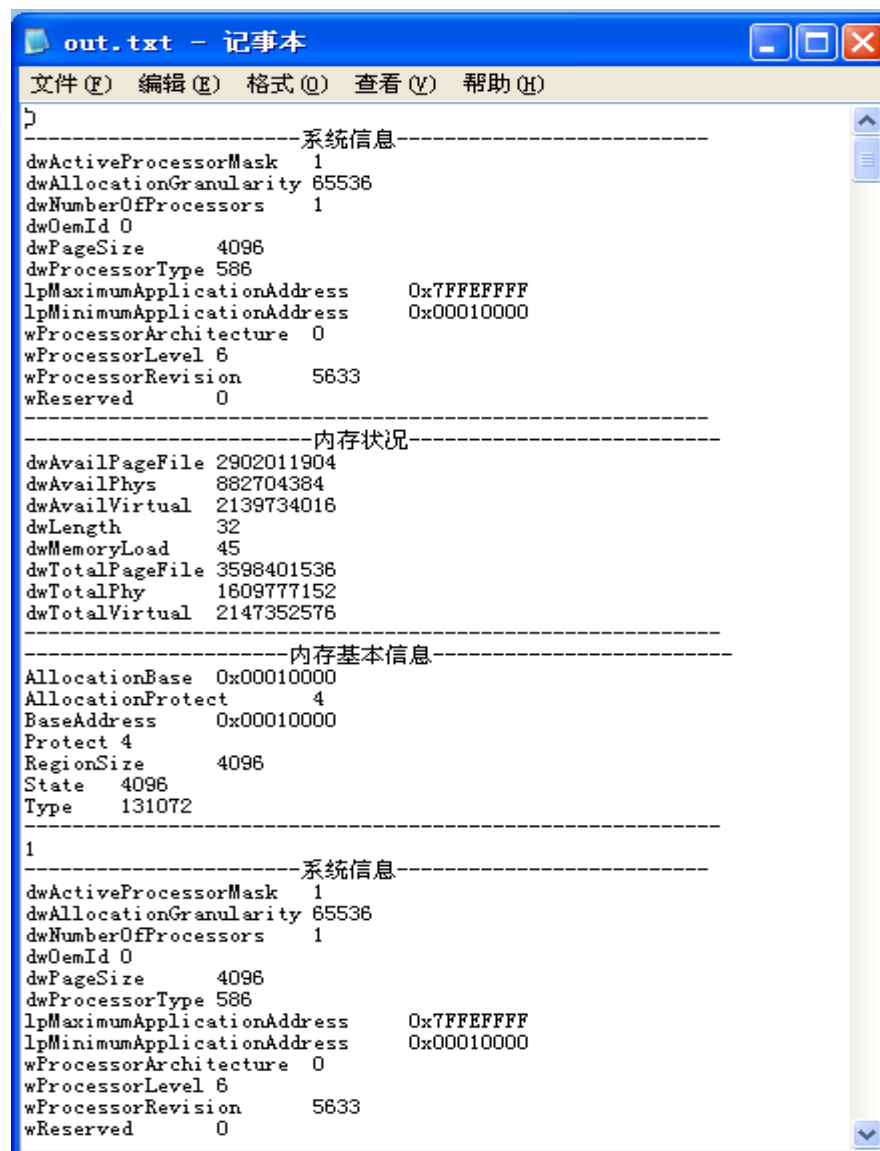


## 二、运行 Memory-op.exe 输出内存操作情况

```

C:\E:\study_soft_files\VC6.0\Microsoft Visual Stu...
0:保留区域
起始地址:0x003A0000 长度:12288
1:保留区域
起始地址:0x003B0000 长度:4096
2:保留区域
起始地址:0x003C0000 长度:20480
3:保留区域
起始地址:0x003D0000 长度:16384
4:保留区域
起始地址:0x003E0000 长度:20480
5:提交区域
起始地址:0x003A0000 长度:12288
6:提交区域
起始地址:0x003B0000 长度:4096
7:提交区域
起始地址:0x003C0000 长度:20480
8:提交区域
起始地址:0x003D0000 长度:16384
9:提交区域
起始地址:0x003E0000 长度:20480
10:锁住区域
起始地址:0x003A0000 长度:12288
11:锁住区域
起始地址:0x003B0000 长度:4096
12:锁住区域
起始地址:0x003C0000 长度:20480
13:锁住区域
起始地址:0x003D0000 长度:16384
14:锁住区域
起始地址:0x003E0000 长度:20480
15:解锁区域
起始地址:0x003A0000 长度:12288
16:解锁区域
起始地址:0x003B0000 长度:4096
17:解锁区域
起始地址:0x003C0000 长度:20480
18:解锁区域
起始地址:0x003D0000 长度:16384
19:解锁区域
起始地址:0x003E0000 长度:20480
20:回收区域
起始地址:0x003A0000 长度:12288
21:回收区域
起始地址:0x003B0000 长度:4096
22:回收区域
起始地址:0x003C0000 长度:20480
23:回收区域
起始地址:0x003D0000 长度:16384
24:回收区域
起始地址:0x003E0000 长度:20480
25:释放区域
起始地址:0x003A0000 长度:12288
26:释放区域
起始地址:0x003B0000 长度:4096
27:释放区域
起始地址:0x003C0000 长度:20480
28:释放区域
起始地址:0x003D0000 长度:16384
29:释放区域
起始地址:0x003E0000 长度:20480
Press any key to continue_
  
```

### 三、二中产生的 out.txt 文件



```

out.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

-----系统信息-----
dwActiveProcessorMask 1
dwAllocationGranularity 65536
dwNumberOfProcessors 1
dwOemId 0
dwPageSize 4096
dwProcessorType 586
lpMaximumApplicationAddress 0x7FFFFFFF
lpMinimumApplicationAddress 0x00010000
wProcessorArchitecture 0
wProcessorLevel 6
wProcessorRevision 5633
wReserved 0
-----内存状况-----
dwAvailPageFile 2902011904
dwAvailPhys 882704384
dwAvailVirtual 2139734016
dwLength 32
dwMemoryLoad 45
dwTotalPageFile 3598401536
dwTotalPhys 1609777152
dwTotalVirtual 2147352576
-----内存基本信息-----
AllocationBase 0x00010000
AllocationProtect 4
BaseAddress 0x00010000
Protect 4
RegionSize 4096
State 4096
Type 131072

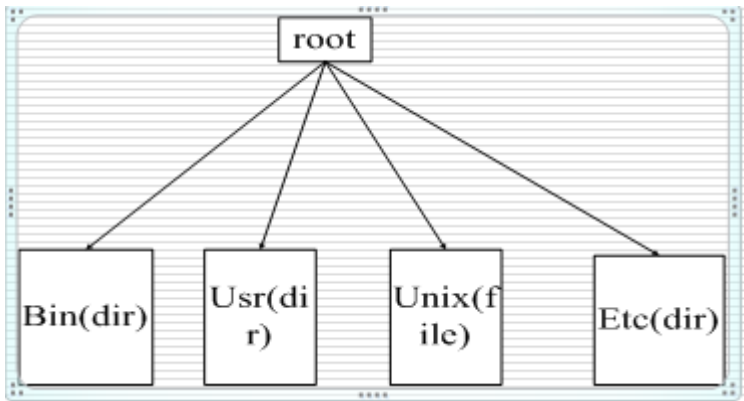
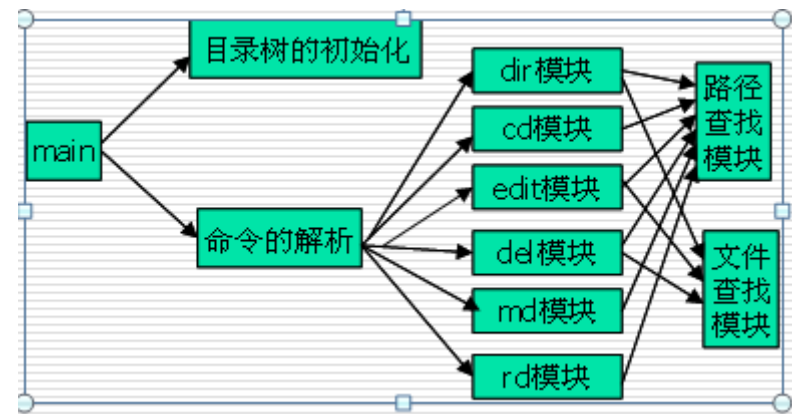
1
-----系统信息-----
dwActiveProcessorMask 1
dwAllocationGranularity 65536
dwNumberOfProcessors 1
dwOemId 0
dwPageSize 4096
dwProcessorType 586
lpMaximumApplicationAddress 0x7FFFFFFF
lpMinimumApplicationAddress 0x00010000
wProcessorArchitecture 0
wProcessorLevel 6
wProcessorRevision 5633
wReserved 0
    
```

#### 小结

本次实验，使我对 windows xp 管理与分配用户进程的虚拟内存空间的方法有了一定的了解，对跟踪程序的编写方法也有了一定的认识，通过对 API 函数的分析与调用，进一步熟悉了 windows 分配虚拟内存、改变内存状态及对物理内存各页面文件状态查询的 API 函数的功能、参数限制与使用规则等。

#### 指导老师意见

### (三) 文件模拟系统

实验 题目	文件系统模拟实验		
姓名:朱名发	学号: 0843041348	专业:计算机科学与技术	完成日期: 2010-6-10
实验内容简要描述(实验内容、目的、环境、主要方法)	<p>一、实验目地</p> <ol style="list-style-type: none"> <li>1、熟悉文件系统的基本功能和结构;</li> <li>2、实现一个简单的类似 dos 的文件系统;</li> </ol> <p>二、实验内容</p> <p>➤ 初始目录的树形结构</p>  <pre> graph TD     root[root] --&gt; Bin["Bin(dir)"]     root --&gt; Uusr["Uusr(dir)"]     root --&gt; Unix["Unix(file)"]     root --&gt; Etc["Etc(dir)"]     </pre> <p>➤ 程序模块图</p>  <pre> graph LR     main[main] --&gt; Init[目录树的初始化]     main --&gt; Parse[命令的解析]     Parse --&gt; Dir[dir模块]     Parse --&gt; Cd[cd模块]     Parse --&gt; Edit[edit模块]     Parse --&gt; Del[del模块]     Parse --&gt; Md[md模块]     Parse --&gt; Rd[rd模块]     Dir --&gt; Path[path查找模块]     Cd --&gt; Path     Edit --&gt; Path     Del --&gt; File[file查找模块]     Md --&gt; File     Rd --&gt; File     </pre> <p>三、实验环境</p> <ol style="list-style-type: none"> <li>1、Windows XP 操作系统的 PC 机。</li> <li>2、本实验程序利用 Windows SDK 编制，所以实验需要在 windows 下 VC 集成开发环境中进行。</li> </ol> <p>四、实验要求</p> <p>➤ 构建一棵树形结构的文件系统</p> <p>➤ 要求程序模拟简单的 dos 文件系统的命令功能:</p> <p>Cd: 类似 dos 的 cd 命令</p> <p>Edit: 创建文件</p> <p>Del: 删除文件</p> <p>Rd:删除目录</p> <p>Dir:显示目录</p> <p>Md:创建目录</p>		

	<p><b>四、实验原理及方法（函数）</b></p> <p>➤ 树型结构的文件系统结构中，节点的结构：</p> <p>//结点结构</p> <pre> struct FileNode {     char filename[FILENAME_LEN]; //文件名/目录名     int isdir; //目录文件识别标志     int i_nlink; //文件的链接数     int adr; //文件的地址     struct FileNode *parent, *child; //指向父亲的指针和指向左孩子的指针     struct FileNode *sibling_prev, *sibling_next; //指向前一个兄弟的指针和指向后一个兄弟的指针. }; </pre> <p>➤ 涉及到的函数</p> <pre> void Forearm(); //创建文件树并初始化 int ParseCommand(); //接受输入的命令并把它分解成操作名和路径文件名 void ExecuteCommand(); //执行命令  int cdComd(); //处理 cd 命令 int editComd(); //处理 edit 命令 int delComd(); //处理 del 命令 int rdComd(); //处理 rd 命令 int dirComd(); //处理 dir 命令 int mdComd(); //处理 md 命令 int FindPath(char *ph); //寻找参数 ph 所指向的路径 //从参数 Para2 中找到要建立或删除的文件、目录名，并把指针指向其父亲结点 int FindfileName(char Para2[]); struct FileNode* CreateFileNode(char fileName[],int isDir,int fileLinkCount); //创建结点 int GetInput(char* buffer,unsigned int buffer_len); //获取输入 int CheckCommand(); //命令检查 int GetDir(int begin,char* path,char* curDir); //获取路径 void Trim(char* str); </pre>
解决问题所用的主要代码及描述（附注释）	<p>////////////////////////////////////</p> <p>//文件名:FileSystem.cpp</p> <p>//程序功能:模拟文件系统</p> <p>//作者:朱名发</p>

```
//日期:2010-6-8
////////////////////////////////////
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<ctype.h>
#include<iostream>
using namespace std;

#define INPUT_LEN 100
#define fileName_LEN 30
#define COMMAND_LEN 20

//文件结点
struct FileNode
{
    char fileName[fileName_LEN];           //文件名或目录名
    int isDir;                             //目录文件标志
    int fileLinkCount;                     //文件的链接数
    int address;                           //文件的路径
    struct FileNode *parent, *leftChild;   //指向父亲和指向左孩子
    的指针
    struct FileNode *siblingPrev, *siblingNext; //指向前一个兄弟的指针
    和指向后一个兄弟的指针.
};

void Forearm();                          //创建文件树并初始化
int ParseCommand();                      //接受输入的命令并把它分解成操作名和路径文
件名
void ExecuteCommand();                  //执行命令

int cdComd();                           //处理 cd 命令
int editComd();                          //处理 edit 命令
int delComd();                           //处理 del 命令
int rdComd();                            //处理 rd 命令
int dirComd();                           //处理 dir 命令
int mdComd();                            //处理 md 命令

int FindPath(char *ph);//寻找参数 ph 所指向的路径
//从参数 Para2 中找到要建立或删除的文件、目录名, 并把指针指向其父亲
结点
int FindfileName(char Para2[]);
```

```

struct FileNode* CreateFileNode(char fileName[],int isDir,int fileLinkCount);//创建结点
int GetInput(char* buffer,unsigned int buffer_len);    //获取输入
int CheckCommand();                                  //命令检查
int GetDir(int begin,char* path,char* curDir);         //获取路径
void Trim(char* str);

struct FileNode * cp, *tp, *root,*upper;
char path[INPUT_LEN-COMMAND_LEN];                    //记录当前走过的路径
char curpath[INPUT_LEN-COMMAND_LEN],Para1[COMMAND_LEN],
Para2[INPUT_LEN-COMMAND_LEN],tmppath[INPUT_LEN-COMMAND_LEN];
char fileName[fileName_LEN],dirname[fileName_LEN],tmp;
int i,j;

//主函数
int main(void)
{
    Forearm();                                        //初始化文件树
    while(1)
    {
        if(ParseCommand())                          //分解命令
            ExecuteCommand();                        //执行命令
    }
    return 0;
}

//执行命令子函数
void ExecuteCommand()
{
    int sign;
    //根据参数 Para1 调用相应的功能处理模块
    if(strcmp(Para1,"CD")==0)
        sign=cdComd();                              //CD 命令
    else if(strcmp(Para1,"EDIT")==0)
        sign=editComd();                            //EDIT 命令
    else if(strcmp(Para1,"DEL")==0)
        sign=delComd();                              //DEL 命令
    else if(strcmp(Para1,"DIR")==0)
        sign=dirComd();                              //DIR 命令
    else if(strcmp(Para1,"MD")==0)
        sign=mdComd();                              //MD 命令
}

```

```

else if(strcmp(Para1,"RD")==0)
    sign=rdComd();                //RD 命令
else if(strcmp(Para1,"Q")==0)
    exit(0);                      //Q 命令
else
    cout<<"命令错误,请重试"<<endl; //命令输入不正确,报错
}

//创建结点
struct FileNode* CreateFileNode(char fileName[],int isDir,int fileLinkCount)
{
    //申请结点空间
    struct FileNode* node=(struct FileNode*)malloc(sizeof(struct FileNode));
    //相应内容赋初值
    strcpy(node->fileName,fileName);
    node->isDir=isDir;
    node->fileLinkCount=fileLinkCount;
    node->parent=NULL;
    node->leftChild=NULL;
    node->siblingPrev=NULL;
    node->siblingNext=NULL;
    return node;
}

//创建文件树并初始化
void Forearm()
{
    struct FileNode *bin,*usr,*unix,*etc;
    strcpy(path,"/");//根目录为当前目录

    //创建初始目录结点
    bin = CreateFileNode("bin",1,0);
    usr = CreateFileNode("usr",1,0);
    unix = CreateFileNode("unix",0,0);
    etc = CreateFileNode("etc",1,0);
    root = cp = tp = CreateFileNode("/",1,0);
    //创建初始目录结点结束
    //初始目录赋值
    root -> parent = NULL;
    root -> leftChild = bin;
    root -> siblingPrev = root -> siblingNext = NULL;

    bin -> parent = root;
    bin -> leftChild = NULL;

```

```

bin -> siblingPrev = NULL;
bin -> siblingNext = usr;

usr -> parent = NULL;
usr -> leftChild = NULL;
usr -> siblingPrev = bin;
usr -> siblingNext = unix;

unix -> parent = NULL;
unix -> leftChild = NULL;
unix -> siblingPrev = usr;
unix -> siblingNext = etc;

etc -> parent = NULL;
etc -> leftChild = NULL;
etc -> siblingPrev = unix;
etc -> siblingNext = NULL;
//初始目录结点赋值结束
cout<<">===== 文 件 系 统 模 拟 实 验
=====<<"<<endl;
cout<<"----- 朱 ----- 名 ----- 发
-----"<<endl;

cout<<"CD|cd----->类似 DOS 的 cd 命令;"<<endl;
cout<<"EDIT|edit----->创建文件;"<<endl;
cout<<"DEL|del----->删除文件;"<<endl;
cout<<"RD|rd----->删除目录;"<<endl;
cout<<"DIR|dir----->显示目录;"<<endl;
cout<<"MD|md----->创建目录;"<<endl;
cout<<"Q|q----->退出系统."<<endl;
cout<<"----- 请 选 相 关 命 令 进 程 文 件 管 理
-----"<<endl;

cout<<"-----"<<endl;

}

//获取文件或目录名，并把指针指向其父亲结点
int FindfileName(char Para2[])
{
    i=strlen(Para2)-1;
    j=0;

    while(Para2[i]!='/'&& i>=0)
    {

```



```

        fileName[j]=Para2[i];
        i--; j++;
    }
    fileName[j]='\0';//获得逆序的文件或目录名, 存入 fileName 中
    if(i<0) Para2[i+1]='\0';
    else Para2[i]='\0';
    j--;

    //fileName 逆转, 获得正确的文件或目录名
    for(i=0;i<strlen(fileName)/2;i++,j--)
    {
        tmp=fileName[i];
        fileName[i]=fileName[j];
        fileName[j]=tmp;
    }

    //查找路径
    if(strlen(Para2)>0)
    {
        int sign=FindPath(Para2);
        if(sign==0) return 0;
    }
    return 1;
}

//缓冲区安全输入子函数
//如果输入超过 buffer_len, 则截取前 buffer_len-1 长度的输入,
//buffer_len 处字符用'\0'代替
int GetInput(char* buffer,unsigned int buffer_len)
{
    int count=0;
    while(count<buffer_len)
    {
        if((buffer[count]=getchar())!=10)
        {
            buffer[count]='\0';
            return count;
        }
        count++;
    }
    while(getchar()!=10);
    buffer[buffer_len-1]='\0';
    return -1;
}

```

```
//分解命令子函数
int ParseCommand()
{
    char Inputs[INPUT_LEN];
    int i=0,j=0,k=0,ch;

    printf("%s>",path);
        //获取输入
    if(GetInput(Inputs,INPUT_LEN)==-1)
    {
        printf("输入行太长。 \n");
        return 0;
    }

    Para1[0]=Para2[0]='\0';

    //获取参数 Para1, 即操作名
    while(Inputs[i]!=' ' && Inputs[i]!='\0' && i<COMMAND_LEN-1)
    {
        Para1[i]=Inputs[i];
        i++;
    }//while
    Para1[i]='\0';

    //输入命令太长
    if(i==(COMMAND_LEN-1))return 1;

    //获取参数 2, 即路径文件名
    if(Inputs[i]!='\0')
    {
        while(Inputs[i]==' ' && i<INPUT_LEN-1) i++;
        j=0;
        while(Inputs[i]!='\0' && i<INPUT_LEN-1)
        {
            Para2[j]=Inputs[i];
            i++; j++;
        }
        Para2[j]='\0';
    }
    Trim(Para1);
    Trim(Para2);

    //将命令全部变成大写字母
```

```

        for(k=0;k<strlen(Para1);k++)
        {
            ch=toupper((int)Para1[k]);
            Para1[k]=ch;
        }
        return 1;
    }

//cd 功能处理子函数
int cdComd()
{
    if(!CheckCommand())//命令检查
        return 0;

    if(strcmp(Para2,"..")==0)
    {        //对 cd ..命令的处理
        int i;

        while(cp->siblingPrev)
            cp=cp->siblingPrev;//找到这一层最左边的结点
        if(cp->parent)
        {
            cp=cp->parent;//找到父亲结点
        }
        else
        {
            return 0;
        }
        //对当前路径进行相应处理
        i=strlen(path);
        while(path[i]!='/'&& i>0) i--;

        if(i!=0)
            path[i]='\0';
        else
            path[i+1]='\0';

    }
    else
    {
        FindPath(Para2);//查找路径
    }
    return 1;
}

```

```
//命令格式处理子函数
void Trim(char* str)
{
    int begin,end;
    char* tmp;
    begin=0;
    end=strlen(str);

    //找到字符串第一个非空格的位置
    while(str[begin]==' ' && str[begin]!='\0')begin++;
    //去除字符串尾部空格
    while(str[--end]==' ');
    str[end+1]='\0';

    //除去空格
    if(begin<end)
    {
        tmp=(char*)malloc((sizeof(char))*(end-begin+2));
        strcpy(tmp,&str[begin]);
        strcpy(str,tmp);
        free(tmp);
    }
}

//获取当前目录名子函数
int GetDir(int begin,char* path,char* curDir)
{
    int i=0;
    int len=strlen(path);
    while(!((path[begin]=='\\') || (path[begin]=='/'))&&begin<len)
    {
        curDir[i++]=path[begin++];
    }
    curDir[i]='\0';
    Trim(curDir);
    return begin+1;
}

//查找路径子函数
int FindPath(char *ph)
{
    struct FileNode *tp,*temp;
```

```

char oldpath[INPUT_LEN-COMMAND_LEN];
int i=0;
int sign=1;
if(strcmp(ph,"")==0)
{
    //ph 是根目录
    cp=root;
    strcpy(path,"");
    return 1;
}

temp=cp;
strcpy(oldpath,path); //保存原路径和指针
if(ph[0]=='/')
{
    //指针指向根目录的左孩子
    cp=root->leftChild;
    i++; //滤过 '/'
    strcpy(path,"");
}
else
{
    if(cp!=NULL && cp!=root )
        strcat(path,"/");

    if(cp && cp->leftChild)
    {
        if(cp->isDir)
            cp=cp->leftChild; //指针指向当前目录的左孩子
        else
        {
            printf("路径错误!\n");
            return 0;
        }
    }
}

while(i<=strlen(ph)&& cp) //继续查找指定路径, 如遇到文件则报错
{
    int j=0;
    if(ph[i]=='/' && cp->leftChild)
    {
        i++; //略过 '/'
        if(cp->isDir)
            cp=cp->leftChild; //继续查找下级目录
        else
        {

```

	<pre>                 printf("路径错误!\n");                 return 0;             }             strcat(path, "/");         }          //curpath 记录当前要找的路径名         while(ph[i]!='/' &amp;&amp; i&lt;=strlen(ph))         {             curpath[j]=ph[i];             i++; j++;         }          curpath[j]='\0';          while((strcmp(cp-&gt;fileName, curpath)!=0    (cp-&gt;isDir!=1)) &amp;&amp; cp-&gt;siblingNext!=NULL)         {             cp=cp-&gt;siblingNext;         }          if(strcmp(cp-&gt;fileName, curpath)==0)         {             if(cp-&gt;isDir==0)             {                 strcpy(path, oldpath);                 cp=temp;                 printf("是文件不是目录.\n");                 return 0;             }              strcat(path, cp-&gt;fileName);         }          if(strcmp(cp-&gt;fileName, curpath)!=0    cp==NULL)         {             strcpy(path, oldpath);             cp=temp;             printf("输入路径错误\n");             return 0;         }     }     return 1; </pre>
--	--

```

}

//创建文件子函数
int editComd()
{
    char tmp;
    struct FileNode * temp=CreateFileNode("",0,0);
    int sign;
    struct FileNode *tp;

    //路径不能为空
    if(strlen(Para2)==0)
    {
        printf("\n 命令格式有错误.\n");
        return 0;
    }
    //长度检查
    if(strlen(Para2)>50)
    {
        printf("\n 文件名过长\n");
        return 0;
    }

    //格式检查
    if (!(isalpha(Para2[0]) || Para2[0]=='_' || Para2[0]=='\0' || Para2[0]=='/'))
    {
        printf("文件名格式有错!\n");/* 文件首字母可以为'字母'或'数字'或
        '_'或'/'或'回车'*/
        return 0;
    }

    //获取文件名
    sign=FindfileName(Para2);
    if(sign==0)
        return 0;

    if(cp->isDir!=1)//如当前指针指向的是文件, 则报错
    {
        printf("you cannot edit a file in under a file!\n");
        return 0;
    }

    //创建文件结点, 并插入到指定目录下
    tp=CreateFileNode("",1,0);
    strcpy(tp->fileName,fileName);

```

```

tp->isDir=0;
tp->fileLinkCount=0;
if(cp->leftChild==NULL)
{
    tp->parent=cp;
    tp->leftChild=NULL;
    cp->leftChild=tp;
    tp->siblingPrev=NULL;
    tp->siblingNext=NULL;
}
else
{
    temp=cp;
    //用 temp 找到新结点插入处
    temp=temp->leftChild;
    while(temp->siblingNext) //find the last sibling node
    {
        temp=temp->siblingNext;
        if(strcmp(temp->fileName,fileName)==0&&temp->isDir==0)
        {
            printf("此文件名已存在\n");//重名报错
            return 0;
        }
    }
    //找到了最后一个结点

    temp->siblingNext=tp;
    tp->parent=NULL;
    tp->leftChild=NULL;
    tp->siblingPrev=temp;
    tp->siblingNext=NULL;

}

return 1;
}

//创建目录子函数
int mdComd()
{
    char tmp;
    int sign;
    struct FileNode * temp, *tp;
    temp=CreateFileNode("",1,0);
    //参数不能为空
    if(strlen(Para2)==0)

```



```

{
    printf("\n 命令格式有错误.\n");
    return 0;
}
//长度检查
if(strlen(Para2)>50)
{
    printf("\n 目录名过长\n");
    return 0;
}

//格式检查
if (!(isalpha(Para2[0]) || Para2[0]=='_' || Para2[0]=='\0' || Para2[0]=='/'))
{
    printf("目录名格式有错!\n");/* 目录首字母可以为'字母'或'数字'或
'_'或'/'或'回车'*/
    return 0;
}

//获取目录名
sign=FindfileName(Para2);
if(sign==0) return 0;

//若当前指针指向的是文件, 则报错
if(cp->isDir!=1)
{
    printf("you cannot make a directory in under a file!\n");
    return 0;
}

//创建目录结点, 并插入到指定目录下
tp=CreateFileNode(fileName,1,0);
if(cp->leftChild==NULL)
{
    tp->parent=cp;
    tp->leftChild=NULL;
    cp->leftChild=tp;
    tp->siblingPrev=NULL;
    tp->siblingNext=NULL;
}
else
{
    temp=cp;
    //用 temp 找到新结点插入处
    temp=temp->leftChild;
    while(temp->siblingNext)//find the last sibling node
    {
        temp=temp->siblingNext;
    }
}

```

```

        if(strcmp(temp->fileName,fileName)==0&&temp->isDir==1)
        {
            printf("此文件名已存在\n");//重名报错
            return 0;
        }
    }//找到了最后一个结点

    temp->siblingNext=tp;
    tp->parent=NULL;
    tp->leftChild=NULL;
    tp->siblingPrev=temp;
    tp->siblingNext=NULL;

}

return 1;
}

//删除文件子函数
int delComd()
{
    char tmp;
    int sign;
    struct FileNode *temp;
    //参数不能为空
    if(strlen(Para2)==0)
    {
        printf("\n 命令格式有错误.\n");
        return 0;
    }

    //获取文件名
    sign=FindfileName(Para2);
    if(sign==0) return 0;

    //用 temp 指向要删除的结点
    if(cp->leftChild)
    {
        temp=cp->leftChild;
        while(temp->siblingNext && (strcmp(temp->fileName,fileName)!=0 ||
temp->isDir!=0))
            temp=temp->siblingNext;
        if(strcmp(temp->fileName,fileName)!=0)
        {

```

```

        printf("不存在该文件!\n");
        return 0;
    }
}
else
{
    printf("不存在该文件!\n");
    return 0;
}

//要删除的不能是目录
if(temp->isDir!=0)
{
    printf("ERROR!该命令只能删除文件,不可删除目录!\n");
    return 0;
}

//如仍有用户使用该文件，则不能删除
if(temp->fileLinkCount!=0)
{
    printf("还有用户共享了该文件,不能删除!\n");
    return 0;
}

//删除工作
if(temp->parent==NULL)//不是第一个孩子
{
    temp->siblingPrev->siblingNext=temp->siblingNext;
    if(temp->siblingNext)//处理是最后一个兄弟的情况
        temp->siblingNext->siblingPrev=temp->siblingPrev;
    temp->siblingPrev=temp->siblingNext=NULL;
}
else//第一个孩子
{
    if(temp->siblingNext)//处理是最后一个兄弟的情况
        temp->siblingNext->parent=temp->parent;
    temp->parent->leftChild= temp->siblingNext;
}
else
{
    free(temp);

    return 1;
}
}
//命令检查子函数
int CheckCommand()
{
    if(strlen(Para2)==0)

```

```

    {
        printf("命令语法不正确。 \n");
        return 0;
    }
    return 1;
}

//删除目录子函数
int rdComd()
{
    char tmp;
    struct FileNode *temp;
    char cmd[2];

    //命令检查
    if(!CheckCommand())
        return 0;
        //获取目录名
    if(!FindfileName(Para2))
        return 0;

    //用 temp 指向要删除的结点
    if(cp->leftChild)
    {
        temp=cp->leftChild;
        while(temp->siblingNext && (strcmp(temp->fileName,fileName)!=0||
temp->isDir!=1))
            temp=temp->siblingNext;
        if(strcmp(temp->fileName,fileName)!=0)
        {
            printf("不存在该目录!\n");
            return 0;
        }
    }
    else
    {
        printf("不存在该目录!\n");
        return 0;
    }

    //要删除的不能是文件
    if(temp->isDir!=1)
    {
        printf("ERROR!该命令只能删除目录,不可删除文件!\n");
    }
}

```

```

        return 0;
    }

    //若要删除的目录不为空, 则提示用户是否删除
    if(temp->leftChild)
    {
        printf("\n 该目录不为空,您确定要删除吗? Y/N \n");

        GetInput(cmd,2);
        if(strcmp(cmd,"n")==0 || strcmp(cmd,"N")==0)
            return 0;
    }

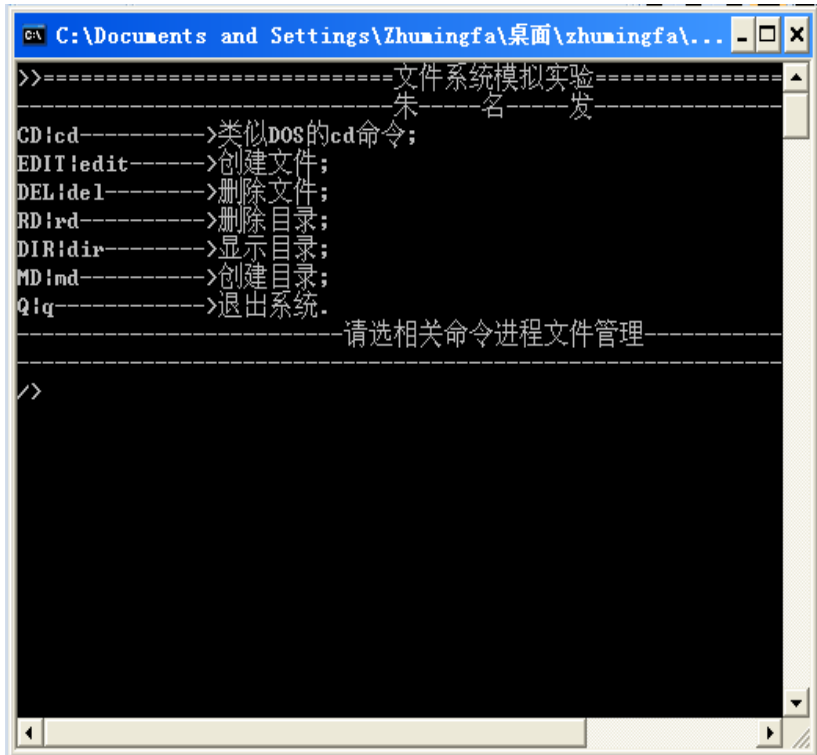
    //删除工作
    if(temp->parent==NULL)//不是第一个孩子
    {
        temp->siblingPrev->siblingNext=temp->siblingNext;
        if(temp->siblingNext)//处理是最后一个兄弟的情况
            temp->siblingNext->siblingPrev=temp->siblingPrev;
        temp->siblingPrev=temp->siblingNext=NULL;

    }//if
    else//第一个孩子
    {
        if(temp->siblingNext)//处理是最后一个兄弟的情况
            temp->siblingNext->parent=temp->parent;
        temp->parent->leftChild=    temp->siblingNext;
    }//else

    free(temp);
    return 1;
}

//显示目录子函数
int dirComd()
{
    if(strlen(Para2)>0)
    {
        int sign=FindPath(Para2);//查找路径
        if(sign==0)
        {
            return 0;
        }
        else
        {

```

	<pre>                 printf("\n%s&gt;", path);             }         }          if(cp!=root)             printf("    &lt;DIR&gt;                %s\n", "..");         if(cp-&gt;leftChild==NULL)         {             return 0;//指定目录为空         }          tp=cp;         //指定目录不为空，显示其所有子目录及文件名         tp=tp-&gt;leftChild;         while(tp)         {             if(tp-&gt;isDir)                 printf("    &lt;DIR&gt;                %s\n",tp-&gt;fileName);             else                 printf("    &lt;FILE&gt;                %s\n",tp-&gt;fileName);             tp=tp-&gt;siblingNext;         }     } </pre>
<p>实验结果分析(或错误原因分析)</p>	<p>一、初始窗口</p> 

## 二、测试过程及结果

```

C:\Documents and Settings\Zhumingfa\桌面\zhumingfa\FileSystem.exe
>>=====文件系统模拟实验=====<<
朱名发
-----
CD!cd----->类似DOS的cd命令;
EDIT!edit---->创建文件;
DEL!del----->删除文件;
RD!rd----->删除目录;
DIR!dir----->显示目录;
MD!md----->创建目录;
Q!q----->退出系统.
-----请选相关命令进程文件管理-----

/ >md zhumingfa
/ >cd zhumingfa
/zhumingfa>md zmf
/zhumingfa>cd zmf
/zhumingfa/zmf>edit zhumingfa.txt
/zhumingfa/zmf>edit hello.txt
/zhumingfa/zmf>dir
<DIR>          ..
<FILE>         zhumingfa.txt
<FILE>         hello.txt
/zhumingfa/zmf>del hello.txt
/zhumingfa/zmf>dir
<DIR>          ..
<FILE>         zhumingfa.txt
/zhumingfa/zmf>cd ..
/zhumingfa>rd zmf
  
```

小结

通过本次实验加深了对文件系统的基本功能及结构的认识与理解。

指导老  
师意见

## (四) shell 程序

实验 题目	Shell 程序		
姓名: 朱名发	学号: 0843041348	专业: 计算机科学与技术	完成日期: 2010-6-15
实验内容简 要描述(实验 内容、目的、 环境、主要方 法)	<p><b>一、实验目地</b></p> <p>Linux 操作系统中 shell 是用户与系统内核沟通的中介, 它为用户使用操作系统的服务提供了一个命令界面。用户在 shell 提示符(\$或#)下输入的每一个命令都由 shell 先解释, 然后传给内核执行。本实验要求用 C 语言编写一个简单的 shell 程序, 希望达到以下目的:</p> <ul style="list-style-type: none"> <li>➤ 用 C 语言编写清晰易读、设计优良的程序, 并附有详细的文档。</li> <li>➤ 熟悉使用 Linux 下的软件开发工具, 例如 gcc、gdb 和 make。</li> <li>➤ 在编写系统应用程序时熟练使用 man 帮助手册。</li> <li>➤ 学习使用 POSIX/UNIX 系统调用、对进程进行管理和完成进程之间的通信, 例如使用信号和管道进行进程间通信。</li> <li>➤ 理解并发程序中的同步问题。</li> <li>➤ 锻炼在团队成员之间的交流与合作能力。</li> </ul> <p><b>二、实验步骤</b></p> <ol style="list-style-type: none"> <li>(1) 阅读关于 fork、exec、wait 和 exit 系统调用的 man 帮助手册。</li> <li>(2) 编写小程序练习使用这些系统调用。</li> <li>(3) 阅读关于函数 tcsetpgrp 和 setpgid 的 man 帮助手册。</li> <li>(4) 练习编写控制进程组的小程序, 要注意信号 SIGTTIN 和 SIGTTOU。</li> <li>(5) 设计命令行分析器(包括设计文档)。</li> <li>(6) 实现命令行分析器。</li> <li>(7) 使用分析器, 写一个简单的 shell 程序, 使它能执行简单的命令。</li> <li>(8) 增加对程序在后台运行的支持, 不必担心后台作业运行结束时要打印一条信息(这属于异步通知)。增加 jobs 命令(这对于调试很有帮助)。</li> <li>(9) 增加输入输出重定向功能。</li> <li>(10) 添加代码支持在后台进程结束时打印出一条信息。</li> <li>(11) 添加作业控制特征, 主要实现对组合键 Ctrl+Z、Ctrl+C 的响应, 还有实现 fg 和 bg 命令功能。</li> <li>(12) 增加对管道的支持。</li> <li>(13) 实现上面的所有细节并集成。</li> <li>(14) 不断测试。</li> <li>(15) 写报告。</li> <li>(16) 结束。</li> </ol> <p><b>三、实验环境</b></p> <ol style="list-style-type: none"> <li>1、装有 linux 虚拟机的 Windows2000/XP 操作系统。</li> <li>2、gcc 编译器, gdb 调试器</li> </ol> <p>本实验的程序用 C 语言编写, 使用 makefile 文件编译整个程序, 生成一个名为 ysh 可执行程序, 在终端输入 “./ysh” 即可执行。makefile 文件的内容: ysh: ysh.c ; cc ysh.c-o ysh。</p>		



## 五、实验要求

本实验要实现一个简单的命令解释器，也就是 Linux 中的 shell 程序。实验程序起名为 ysh，要求其设计类似于目前流行的 shell 解释程序，如 bash、csh、tcsh，但不需要具备那么复杂的功能。ysh 程序应当具有如下一些重要的特征：

- 能够执行外部程序命令，命令可以带参数。
- 能够执行 fg、bg、cd、history、exit 等内部命令。
- 使用管道和输入输出重定向。
- 支持前后台作业，提供作业控制功能，包括打印作业的清单，改变当前运行作业的前台/后台状态，以及控制作业的挂起、中止和继续运行。除此之外，在这个实验中还须做到：
- 使用 make 工具建立工程。
- 使用调试器 gdb 来调试程序。
- 提供清晰、详细的设计文档和解决方案。

## 五、实验原理及方法（函数）

### ➤ 程序的结构

在 shell 命令里，我们将 ysh 中的命令分成 4 种：普通命令，重定向命令，管道命令和内部命令。这 4 种命令的分析和执行各有不同，每一种命令的分析执行程序都应包括：初始化环境，打印提示符，获取用户输入命令，解析命令，寻找命令文件和执行命令几个步骤，具体程序流程如图 5.4 所示



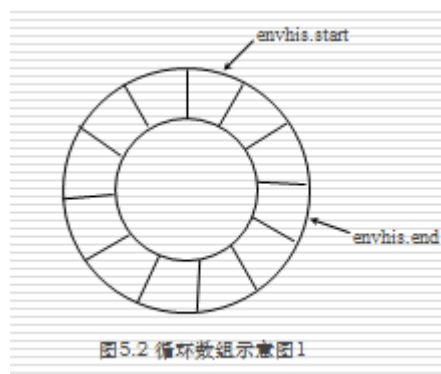
### ➤ 循环数组

在 history 命令中，用数组来存放我们输入过的历史命令。假设我们设定一个能够记录 12 条历史纪录的数组如图 5.2 所示。数组的定义如下：

```

typedef struct ENV_HISTROY{
    int start=0;
    int end=0;
}
    
```

```
char his_cmd[12][100];
}ENV_HISTORY;
ENV_HISTORY envhis;
```

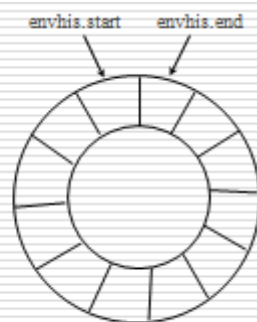


可以看到，每个 `his_cmd[i]` 对应图中一块圆环 (`end` 不一定为 12)，一共 12 块，能存放十二条命令。当用户输入一命令时，只须执行如下语句即可将输入命令存放进相应数组中：

```
envhis.end=envhisend+1;
strcpy(his_cmd[envhis.end], input)
```

但是还需要考虑如图5.3所示的情况：

在这种情况下，`end=12`，当我们在输入一条命令时，如果还是用上述两条命令进行处理 `end=end+1`，则 `end=13` 就会出错。所以应对程序进一步修改。（同学思考）



### ➤ 链表

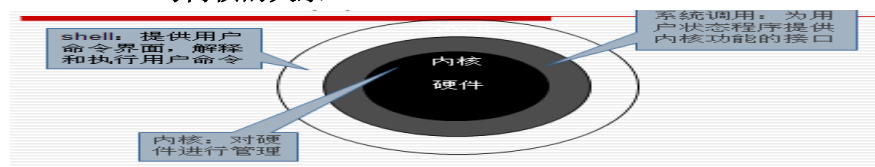
由于我们把作业以链表的形式保存起来，所以在处理 `jobs` 命令时，实际上就是对链表的操作。

定义链表的节点：

```
typedef struct NODE(
    pid_t pid;           /*进程号*/
    char cmd[100];       /*命令名*/
    char state[10];      /*作业状态*/
    struct NODE * link;  /*下一节点指针*/
)NODE;
```

`NODE *head`, `*end`//`head` 指针指向链表表头，`end` 指针指向链表尾。

### ➤ shell 与内核的关系



	<p>➤ 主要方法</p> <pre> int is_founded(char * cmd);//查找命令文件 void GetEnviron(int n,char * s);//将 getline(fd, buf)读到 buf 中的信息以冒号 分开, 分别放于 envpath[]中(见 ysh.h 中的定义), 为后面查找命令做准备 int GetLine(int fd,char * buf); //读取一行的信息到 buf 中 void init_environ();//初始化环境 int redirect(char * in,int len); int pipel(char * input,int len);//管道操作, 包括命令解析、寻找命令文件、 命令的执行和释放空间等 void add_history(char * inputcmd);//添加历史记录 void history_cmd();//历史记录 void cd_cmd(char * route);//类似于 windows 下的 cd 命令 void jobs_cmd();//作业控制命令 //增删节点 void add_node(char * input_cmd,int node_pid); void del_node(int sig,signfo_t * sip); void setflag(); void ctrl_z(); //下面是两个作业控制命令 (bg,fg) 处理函数  void bg_cmd(int job_num); void fg_cmd(int job_num); </pre>
<p>解决问题所 用的主要代 码及描述 (附注释)</p>	<pre> //////////////////////////////////// //文件名:fg_cmd.c //程序功能:shell 程序 //////////////////////////////////// #include &lt;sys/types.h&gt; #include &lt;sys/stat.h&gt; #include &lt;stdio.h&gt; #include &lt;math.h&gt; #include &lt;ctype.h&gt; #include &lt;string.h&gt; #include &lt;unistd.h&gt; #include &lt;fcntl.h&gt; #include &lt;sys/wait.h&gt; #include &lt;stdlib.h&gt; #include &lt;signal.h&gt; #include "ysh.h"  //定义宏, 初始化文件描述符 #define NO_PIPE -1 #define FD_READ 0 #define FD_WRITE 1 </pre>

```

int is_founded(char * cmd);//查找命令文件
void GetEnviron(int n,char * s);//将 getline(fd, buf)读到 buf 中的信息以冒号分开，分别放于 envpath[]中(见 ysh.h 中的定义)，为后面查找命令做准备
int GetLine(int fd,char * buf); //读取一行的信息到 buf 中
void init_environ();//初始化环境
int redirect(char * in,int len);
int pipel(char * input,int len);//管道操作，包括命令解析、寻找命令文件、命令的执行和释放空间等
void add_history(char * inputcmd);//添加历史记录
void history_cmd();//历史记录
void cd_cmd(char * route);//类似于 windows 下的 cd 命令
void jobs_cmd();//作业控制命令
//增删节点
void add_node(char * input_cmd,int node_pid);
void del_node(int sig,signinfo_t * sip);
void setflag();
void ctrl_z();
//下面是两个作业控制命令（bg,fg）处理函数

void bg_cmd(int job_num);
void fg_cmd(int job_num);

int main(void)
{
    init_environ();//初始化环境
    while(1)
    {
        char c;
        char * arg[20];
        int i = 0,j = 0,k = 0;
        int is_pr = 0,is_bg = 0;
        int input_len = 0,path;
        int pid = 0,status = 0;
        struct sigaction action;
        action.sa_sigaction = del_node;
        sigfillset(& action.sa_mask);
        action.sa_flags = SA_SIGINFO;
        sigaction(SIGCHLD,& action,NULL);
        signal(SIGTSTP,ctrl_z);
        path = get_current_dir_name();
        printf("ysh>>%s ",(char *)path);
        while(((c = getchar()) == ' ') || (c == '\t') || (c == EOF))
        {

```

```

;
}
if(c == '\n')
{
    continue;
}
while(c != '\n')
{
    buf[input_len++] = c;
    c = getchar();
}
buf[input_len] = '\0';
input = (char *)malloc(sizeof(char) * (input_len + 1));
strcpy(input,buf);
for(i = 0,j = 0,k = 0;i <= input_len;i++)
{
    if((input[i] == '<') || (input[i] == '>') || (input[i] == '|'))
    {
        if(input[i] == '|')
        {
            pipel(input,input_len);
            add_history(input);
            free(input);
        }
        else
        {
            redirect(input,input_len);
            add_history(input);
            free(input);
        }
        is_pr = 1;
        break;
    }
}
if(is_pr == 1)
{
    continue;
}
for(i = 0,j = 0,k = 0;i <= input_len;i++)
{
    if((input[i] == ' ') || (input[i] == '\0'))
    {
        if(j == 0)
        {

```

	<pre>         continue;     }     else     {         buf[j++] = '\0';         arg[k] = (char *)malloc(sizeof(char) * j);         strcpy(arg[k++],buf);         j = 0;     } } else {     if((input[i] == '&amp;') &amp;&amp; (input[i + 1] == '\0'))     {         is_bg = 1;         continue;     }     buf[j++] = input[i]; } } if(strcmp(arg[0],"exit") == 0) {     add_history(input);     printf("Bye bye!\n");     free(input);     break; } if(strcmp(arg[0],"history") == 0) {     add_history(input);     history_cmd();     free(input);     continue; } if(strcmp(arg[0],"cd") == 0) {     add_history(input);     for(i = 3,j = 0;i &lt;= input_len;i++)     {         buf[j++] = input[i];     }     buf[j] = '\0';     arg[1] = (char *)malloc(sizeof(char) * j);     strcpy(arg[1],buf); </pre>
--	---

	<pre> cd_cmd(arg[1]); free(input); continue; } if(strcmp(arg[0],"jobs") == 0) {     add_history(input);     jobs_cmd();     free(input);     continue; } if(strcmp(arg[0],"bg") == 0) {     add_history(input);     for(i = 0;i &lt;= input_len;i++)     {         if(input[i] == '%')         {             break;         }     }     i++;     for(;i &lt;= input_len;i++)     {         buf[j++] = input[i];     }     buf[j] = '\0';     arg[1] = (char *)malloc(sizeof(char) * j);     strcpy(arg[1],buf);     bg_cmd(atoi(arg[1]));     free(input);     continue; } if(strcmp(arg[0],"fg") == 0) {     add_history(input);     for(i = 0;i &lt;= input_len;i++)     {         if(input[i] == '%')         {             break;         }     }     i++; </pre>
--	---

	<pre> for(i &lt;= input_len;i++) {     buf[j++] = input[i]; } buf[j] = '\0'; arg[1] = (char *)malloc(sizeof(char) * j); strcpy(arg[1],buf); fg_cmd(atoi(arg[1])); free(input); continue; } if(is_pr == 0) {     arg[k] = (char *)malloc(sizeof(char));     arg[k] = NULL;     if(is_founded(arg[0]) == 0)     {         printf("This command is not founded!\n");         for(i = 0;i &lt;= k;i++)         {             free(arg[i]);         }         continue;     } } add_history(input); if((pid = fork()) == 0) {     if(is_bg == 1)     {         while(sig_flag == 0)         {             signal(SIGUSR1,setflag);         }         sig_flag = 0;     } } else {     pid1 = pid;     if(is_bg == 1)     {         add_node(input,pid1);         kill(pid,SIGUSR1);     } } </pre>
--	---



```

        pid1 = 0;
    }
    if(is_bg == 0)
    {
        waitpid(pid,& status,0);
    }
}
if(is_bg == 1)
{
    sleep(1);
}
for(i = 0; i < k; i++)
{
    free(arg[i]);
    free(input);
}
}
return 0;
}

int is_founded(char * cmd)
{
    int k = 0;
    while(envpath[k] != NULL)
    {
        strcpy(buf,envpath[k]);
        strcat(buf,cmd);
        if(access(buf,F_OK) == 0)
        {
            return 1;
        }
        k++;
    }
    return 0;
}

void GetEnviron(int n,char * s)
{
    int i = 0,j = 0,k = 0;
    char c;
    char buff[80];
    char * p;
    while((c=s[i]) != '=')
    {

```

```

        buff[i++] = c;
    }
    buff[i++] = '\0';
    if(strcmp(buff,"PATH") == 0)
    {
        while(s[i] != '\0')
        {
            if(s[i] == ':')
            {
                buff[j++] = '/';
                buff[j] = '\0';
                p = (char *)malloc(strlen(buff) + 1);
                strcpy(p,buff);
                envpath[k++] = p;
                envpath[k] = NULL;
                j = 0;
                i++;
            }
            else
            {
                buff[j] = s[i];
                j++;
                i++;
            }
        }
    }
    else
    {
        fprintf(stderr,"No match");
    }
}

int GetLine(int fd,char * buf)
{
    int i = 0;
    char c;
    while(read(fd,& c,1))
    {
        buff[i++] = c;
        if(c == '\n')
        {
            buff[i-1] = '\0';
            return i;
        }
    }
}

```

```

    }
    return i;
}

void init_envron()
{
    int fd,n;
    char buf[80];
    if((fd = open("ysh_profile",O_RDONLY,660)) == -1)
    {
        printf("init environ variable error!\n");
        exit(1);
    }
    while((n = GetLine(fd,buf)) != 0)
    {
        GetEnviron(n,buf);
    }
    envhis.start = 0;
    envhis.end = 0;
    head = end = NULL;
}

int redirect(char * in,int len)
{
    char * argv[20],* filename[20];
    pid_t pid;
    int i,j,k,fd_in,fd_out,is_in = -1,is_out = -1,num = 0;
    int is_back = 0,status = 0;
    for(i = 0,j = 0,k = 0;i <= len;i++)
    {
        if((in[i] == ' ') || (in[i] == '\t') || (in[i] == '\0') || (in[i] == '<') || (in[i] == '>'))
        {
            if((in[i] == '>') || (in[i] == '<'))
            {
                if(num < 3)
                {
                    num++;
                    if(in[i] == '<')
                    {
                        is_in = num - 1;
                    }
                    else
                    {
                        is_out = num - 1;
                    }
                }
            }
        }
    }
}

```

	<pre>         }         if(j &gt; 0 &amp;&amp; num == 1)         {             buf[j++] = '\0';             argv[k] = (char *)malloc(sizeof(char) * j);             strcpy(argv[k],buf);             k++;             j = 0;         }     }     else     {         printf("Error command!\n");         return 0;     } } if(j == 0) {     continue; } else {     buf[j++] = '\0';     if(num == 0)     {         argv[k] = (char *)malloc(sizeof(char) * j);         strcpy(argv[k],buf);         k++;         j = 0;     }     else     {         filename[status] = (char *)malloc(sizeof(char) * j);         strcpy(filename[status++],buf);         j = 0;     } } } else {     if(in[j] == '&amp;' &amp;&amp; in[j + 1] == '\0')     {         is_back = 1;         continue; </pre>
--	--

```

    }
    buf[j++] = in[i];
    }
}
argv[k] = (char *)malloc(sizeof(char));
argv[k] = (char *)0;
if(is_founded(argv[0]) == 0)
{
    printf("This command is not founded!\n");
    for(i = 0; i <= k; i++)
    {
        free(argv[i]);
    }
    return 0;
}
if((pid = fork()) == 0)
{
    if(is_out != -1)
    {
        if(fd_out == open(filename[is_out], O_WRONLY | O_TRUNC, S_IRUSR |
S_IWUSR) == -1)
        {
            printf("Can not open%s\n", filename[is_out]);
            return 0;
        }
    }
    if(is_in != -1)
    {
        if((fd_in = open(filename[is_in], O_RDONLY, S_IRUSR | S_IWUSR)) == -1)
        {
            printf("Can not open%s\n", filename[is_in]);
            return 0;
        }
    }
    if(is_out != -1)
    {
        if(dup2(fd_out, STDOUT_FILENO) == -1)
        {
            printf("Redirect standard out error!\n");
            exit(1);
        }
    }
    if(is_in != -1)
    {

```

```

        if(dup2(fd_in,STDIN_FILENO) == -1)
        {
            printf("Redirect standard in error!\n");
            exit(1);
        }
    }
    execv(buf,argv);
}
else
{
    if(is_back == 0)
    {
        waitpid(pid,& status,0);
    }
}
for(i = 0;i <= k;i++)
{
    free(argv[i]);
}
if(is_out != -1)
{
    free(filename[is_out]);
    close(fd_out);
}
if(is_in != -1)
{
    free(filename[is_in]);
    close(fd_in);
}
return 0;
}

```

```

int pipel(char * input,int len)
{
    char * argv[10][30];
    char * filename[0];
    int i,j,k,is_bg = 0;
    int li_cmd = 0;
    int fd[10][1],pipe_in = -1;
    int pipe_out = -1,flag = 0;
    pid_t pid;
    for(i = 0,j = 0,k = 0;i <= len;i++)
    {

```

	<pre> if((input[i] == ' ')    (input[i] == '\t')    (input[i] == '\0')       (input[i] == ' ')    (input[i] == '&gt;')    (input[i] == '\n')) {     if((input[i] == ' ')    (input[i] == '&gt;'))     {         if(input[i] == '&gt;')         {             flag=1;         }         if(j &gt; 0 )         {             buf[j++] = '\0';             argv[li_cmd][k] = (char *)malloc(sizeof(char) * j);             strcpy(argv[li_cmd][k],buf);             k++;         }         argv[li_cmd][k] = (char *)0;         li_cmd++;         k = 0;         j = 0;     }     if(j == 0)     {         continue;     }     else     {         buf[j++] = '\0';         if(flag == 0)         {             argv[li_cmd][k] = (char *)malloc(sizeof(char) * j);             strcpy(argv[li_cmd][k],buf);             k++;         }         else         {             filename[0] = (char *)malloc(sizeof(char) * j);             strcpy(filename[0],buf);         }     }     j = 0; } else { </pre>
--	--

	<pre>         if((input[i] == '&amp;') &amp;&amp; (input[i] == '\0'))         {             is_bg = 1;             continue;         }         buf[j++] = input[i];     } } argv[li_cmd][k++] = NULL; for(i = 0; i &lt;= 10; i++) {     fd[i][FD_READ] = NO_PIPE;     fd[i][FD_WRITE] = NO_PIPE; } for(i = 0; i &lt; li_cmd; i++) {     if(pipe(fd[i]) == -1)     {         printf("Can not open pipe!\n");         return 0;     } } for(i = 0; i &lt; li_cmd; i++) {     if(is_founded(argv[i][0]) == 0)     {         printf("Can not found command!\n");         break;     } } if(i != 0) {     pipe_in = fd[i - 1][FD_READ]; } else {     pipe_in = NO_PIPE; } if(i != li_cmd) {     pipe_out = fd[i][FD_WRITE]; } else { </pre>
--	--



```

if(flag == 1)
{
    if((pipe_out = open(filename[0],O_WRONLY | O_CREAT | O_TRUNC,
        S_IRUSR | S_IWUSR)) == -1)
    {
        printf("Can not open %s\n",filename[0]);
        return 0;
    }
}
else
{
    pipe_out = NO_PIPE;
}
}
if((pid = fork()) < 0)
{
    printf("Fork failed!\n");
    return 0;
}
if(pid == 0)
{
    if(pipe_in == NO_PIPE)
    {
        close(pipe_in);
    }
    if(pipe_out == NO_PIPE)
    {
        close(pipe_out);
    }
    if(pipe_out != NO_PIPE)
    {
        dup2(pipe_out,1);
        close(pipe_out);
    }
    if(pipe_in != NO_PIPE)
    {
        dup2(pipe_in,0);
        close(pipe_in);
    }
    execv(buf,argv[i]);
}
else
{
    if(is_bg == 0)

```

```

{
    waitpid(pid,NULL,0);
}
close(pipe_in);
close(pipe_out);
}
return 0;
}

void add_history(char * inputcmd)
{
    envhis.end = (envhis.end + 1) % HISNUM;
    if(envhis.end == envhis.start)
    {
        envhis.start = (envhis.start + 1) % HISNUM;
    }
    strcpy(envhis.his_cmd[envhis.end],inputcmd);
}

void history_cmd()
{
    int i,j = 0;
    if(envhis.start == envhis.end)
    {
        return;
    }
    else if(envhis.start < envhis.end)
    {
        for(i = envhis.start + 1;i <= envhis.end;i++)
        {
            printf("%d\t%s\n",j,envhis.his_cmd[i]);
            j++;
        }
    }
    else
    {
        for(i = envhis.start + 1;i < HISNUM;i++)
        {
            printf("%d\t%s\n",j,envhis.his_cmd[i]);
            j++;
        }
        for(i = 0;i <= envhis.end;i++)
        {

```

```

        printf("%d\t%s\n",j,envhis.his_cmd[i]);
        j++;
    }
}

void cd_cmd(char * route)
{
    if(route != NULL)
    {
        if(chdir(route) < 0)
        {
            printf("cd;%s Error file or directory!\n",route);
        }
    }
}

void jobs_cmd()
{
    struct NODE * p;
    int i = 1;
    p = head;
    if(head != NULL)
    {
        do
        {
            printf("%d  %d  %s\t%s\n",i,p -> pid,p -> state,p -> cmd);
            i++;
            p = p -> link;
        }while(p != NULL);
    }
    else
    {
        printf("No jobs!\n");
    }
}

void add_node(char * input_cmd,int node_pid)
{
    struct NODE * p;
    p = (struct NODE *)malloc(sizeof(struct NODE));
    p -> pid = node_pid;
    strcpy(p -> state,input_cmd);
    strcpy(p -> state,"running");
    p -> link = NULL;
    if(head == NULL)

```

```
{
    head = p;
    end = p;
}
else
{
    end -> link = p;
    end = p;
}
}

void del_node(int sig, siginfo_t * sip)
{
    struct NODE * q;
    struct NODE * p;
    int id;
    if(sig_z == 1)
    {
        sig_z = 0;
        goto out;
    }
    id = sip -> si_pid;
    p = q = head;
    if(head == NULL)
    {
        goto out;
    }
    while(p -> pid != id && p -> link != NULL)
    {
        p = p -> link;
    }
    if(p -> pid != id)
    {
        goto out;
    }
    if(p == head)
    {
        head = head -> link;
    }
    else
    {
        while(q -> link != p)
        {
            q = q -> link;
        }
    }
}
```

```
}
if(p == end)
{
    end = q;
    q -> link = NULL;
}
else
{
    q -> link = p -> link;
}
}
free(p);
out:return;
}
void setflag()
{
    sig_flag = 1;
}
void ctrl_z()
{
    struct NODE * p;
    int i = 1;
    if(pid1 == 0)
    {
        goto out;
    }
    if(head != NULL)
    {
        p = head;
        while((p -> pid != pid1) && (p -> link != NULL))
        {
            p = p -> link;
        }
        if(p -> pid == pid1)
        {
            strcpy(p -> state, "stopped");
        }
        else
        {
            add_node(input, pid1);
            strcpy(end -> state, "stopped");
        }
    }
    else
```

```

{
    add_node(input,pid1);
    strcpy(end -> state,"stopped");
}
sig_z = 1;
kill(pid1,SIGSTOP);
for(p = head;p -> pid != pid1;p = p -> link)
{
    i++;
}
printf("[%d]\t%s\t%s\n",i,end -> state,end -> cmd);
pid1 = 0;
out:return;
}
void bg_cmd(int job_num)
{
    struct NODE * p;
    int i = 0;
    p = head;
    for(i = 1;i < job_num;i++)
    {
        p = p -> link;
    }
    kill(p -> pid,SIGCONT);
    strcpy(p -> state,"running");
}
void fg_cmd(int job_num)
{
    struct NODE * p;
    int i = 0;
    p = head;
    for(i = 1;i < job_num;i++)
    {
        p = p -> link;
    }
    strcpy(p -> state,"running");
    strcpy(input,p -> cmd);
    pid1 = p -> pid;
    signal(SIGTSTP,ctrl_z);
    kill(p -> pid,SIGCONT);
    waitpid(p -> pid,NULL,0);
}

```

## 一、调试过程

实验结果分  
析(或错误原  
因分析)

```

brucezmf@brucezmf-laptop: ~/桌面/exp1
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
brucezmf@brucezmf-laptop:~$ cd '/home/brucezmf/桌面/exp1'
brucezmf@brucezmf-laptop:~/桌面/exp1$ gcc -g fg_cmd.c -o myshell
fg_cmd.c: In function 'main':
fg_cmd.c:52: warning: assignment from incompatible pointer type
brucezmf@brucezmf-laptop:~/桌面/exp1$ gdb myshell
GNU gdb (GDB) 7.0-ubuntu
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/brucezmf/桌面/exp1/myshell...done.
(gdb) l
33     void ctrl_z();
34     //下面是两个作业控制命令 (bg,fg) 处理函数
35
36     void bg_cmd(int job_num);
37     void fg_cmd(int job_num);
38
39
40     int main(void)
41     {
42         init_envron();    //初始化环境
(gdb)
43         while(1)
44         {
45             char c;
46             char * arg[20];
47             int i = 0,j = 0,k = 0;
48             int is pr = 0,is bg = 0;
49
50             int input_len = 0,path;
51             int pid = 0,status = 0;
52             struct sigaction action;
53             action.sa_sigaction = del_node;
(gdb) break 42
Breakpoint 1 at 0x8048a71: file fg_cmd.c, line 42.
(gdb) break init_envron
Breakpoint 2 at 0x804964d: file fg_cmd.c, line 330.
(gdb) info break
Num      Type             Disp Enb Address          What
1        breakpoint       keep y   0x08048a71 in main at fg_cmd.c:42
2        breakpoint       keep y   0x0804964d in init_envron at fg_cmd.c:330
(gdb) r
Starting program: /home/brucezmf/桌面/exp1/myshell

Breakpoint 1, main () at fg_cmd.c:42
42         init_envron();    //初始化环境
(gdb) n

Breakpoint 2, init_envron () at fg_cmd.c:330
330     {
(gdb) n
333         if((fd = open("ysh_profile",O_RDONLY,660)) == -1)
(gdb) n
338         while((n = GetLine(fd,buf)) != 0)
(gdb) c
Continuing.
ysh>>/home/brucezmf/桌面/exp1

```

## 二、测试过程及结果

```

brucezmf@brucezmf-laptop: ~/桌面/exp1
文件(E) 编辑(E) 查看(V) 终端(T) 帮助(H)
brucezmf@brucezmf-laptop:~$ cd '/home/brucezmf/桌面/exp1'
brucezmf@brucezmf-laptop:~/桌面/exp1$ ./myshell
ysh>>/home/brucezmf/桌面/exp1 jobs
No jobs!
ysh>>/home/brucezmf/桌面/exp1 cd zhumingfa
ysh>>/home/brucezmf/桌面/exp1/zhumingfa cd ..
ysh>>/home/brucezmf/桌面/exp1 history
0      jobs
1      cd zhumingfa
2      cd ..
3      history
ysh>>/home/brucezmf/桌面/exp1 exit
Bye bye!
brucezmf@brucezmf-laptop:~/桌面/exp1$
    
```

## 三、运行前后包的变化

### ➤ 运行前



### ➤ 运行后



### 小结

通过本次实验熟悉了 linux 的基本使用方法, 加深了对 shell 程序的理解, 对其内部命令的运行机制有了个简陋的认识。学会了在 linux 环境下进行一些简单的 c 语言编程。对 gdb 调试也有了个粗步的认识!

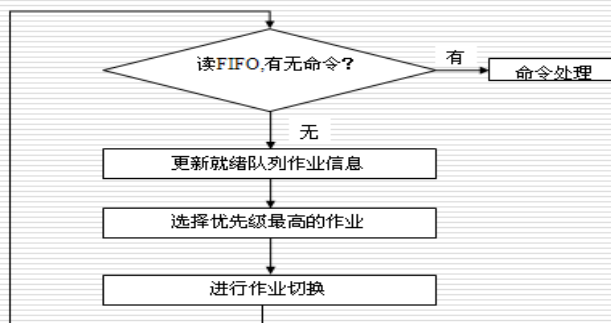
### 指导老师 老师意见



## (五) 作业调度系统

实验 题目	作业调度系统		
姓名:朱名发	学号: 0843041348	专业: 计算机科学与技术	完成日期: 2010-6-15
实验内容简 要描述(实验 内容、目的、 环境、主要方 法)	<p><b>一、实验目地</b></p> <ul style="list-style-type: none"> <li>➤ 理解操作系统中调度的概念和调度算法。</li> <li>➤ 学习 Linux 下进程控制以及进程之间通信的知识。</li> <li>➤ 理解在操作系统中作业是如何被调度的，如何协调和控制各作业对 CPU 的使用。</li> </ul> <p><b>二、实验内容</b></p> <p>要求实现用户空间内的作业调度系统，通过作业调度系统实现以下的操作：</p> <ol style="list-style-type: none"> <li>(1) 提交自己的作业。</li> <li>(2) 将自己提交的作业移出。</li> <li>(3) 查看作业状态。</li> </ol> <p><b>三、实验环境</b></p> <ol style="list-style-type: none"> <li>1、ubuntu 操作系统。</li> <li>2、gcc 编译器，gdb 调试器，gedit 编辑器。</li> </ol> <p><b>四、实验设计与分析</b></p> <p>(一) 为实现作业调度系统，本实验应包括以下程序：</p> <ol style="list-style-type: none"> <li>1. 作业调度程序</li> <li>2. 作业入队命令</li> <li>3. 作业出队命令</li> <li>4. 作业状态查看命令</li> </ol> <p>(二) 作业调度程序</p> <p>scheduler 是作业调度程序，负责整个系统的运行。这是一个不断循环运行的进程，其任务是处理作业的入队、出队以及状态查看请求，在合适的时间调度各作业运行。</p> <p>作业执行一次的时间为 10 毫秒，每 10 毫秒结束时，强制当前运行的进程停止执行(抢占式多任务)，将其状态改为 READY。</p> <p>其中，检查是否有入队、出队以及状态查看等请求：</p> <ul style="list-style-type: none"> <li>➤ 如果有新的作业到来，为其创建一个进程，其状态为就绪，然后将其放入就绪队列中。</li> <li>➤ 如果有出队请求则使该作业出队，然后清除相关的数据结构，若该作业当前正在运行，则发信号给它，使它停止运行，然后出队。</li> <li>➤ 如果是状态查看请求，则将状态输出。</li> <li>➤ 响应这些请求，然后更新所有作业的当前优先级，从中选择一个当前优先级最高的作业来运行。</li> <li>➤ 若当前没有作业可以调度，即作业队列为空，则运行 scheduler 自身(等待，直到这个时间片到期)。</li> </ul>		

作业调度流程图：



### (三) 作业入队命令

enq 是作业入队命令，其格式：

enq [-p num] e\_file args

其中：

- 1) -p num: 可通过-p 选项以及参数 num 指定该作业的初始优先级。若不指定，则作业默认的初始优先级为 0。如果 num 大于 3 或者小于 0，其初始优先级也为 0。
- 2) e\_file args: e\_file 是可执行文件的名字(必须是以“/”开始的绝对路径名)，args 是可执行文件的参数
- 3) 用户通过本命令，给 scheduler 调度程序发出入队请求，将作业提交给系统运行。

### (四) 作业出队命令

deq 是作业出队命令。

格式：deq jid

每一个作业提交时，scheduler 调度程序都将为其分配一个惟一的 jid。

使用 deq 命令，给 scheduler 调度程序发出一个出队请求

### (五) 作业状态查看命令

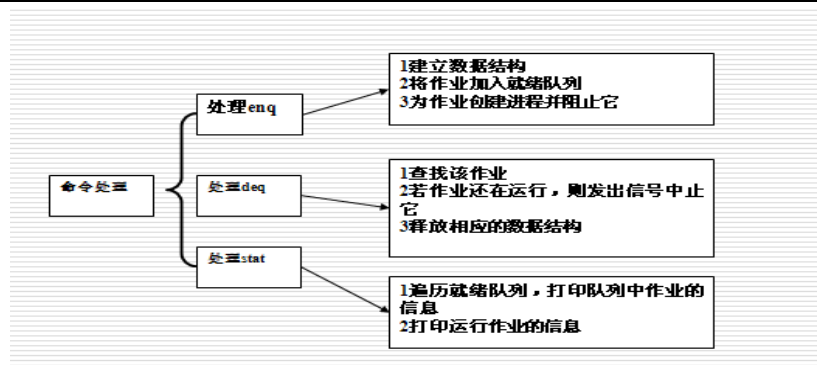
- 1) stat 是查看作业状态的命令。

- 2) 格式：stat

3) 在标准输出上打印出就绪队列中各作业的信息。状态信息应该包括如下几点：

- 进程的 pid;
- 作业提交者的 user name;
- 作业执行时间，就绪队列中总的等待时间;
- 作业创建的时刻;
- 此时作业的状态(READY、RUNNING)。

4) 以上三个命令程序的实现：接收用户传进来的参数，对命令进行包装，将数据结构写进 FIFO 中。



### (六) 调度策略

采用多级反馈的循环(round robin)调度策略。每个作业有其动态的优先级，在用完分配的时间片后，可以被优先级更高的作业抢占运行。等待队列中的作业等待时间越长，其优先级越高。优先级分为 0、1、2 和 3 共 4 个等级，3 为最高。其中每个作业都有两种优先级：

- (1) 初始优先级：作业提交时指定并保持不变，直至作业结束
- (2) 当前优先级：由 scheduler 调度更新，用以调度

作业当前优先级的更新主要取决于以下两种情况：

- (1) 一个作业在就绪队列中等待了 100 毫秒，将它的当前优先级加 1(最高为 3)。
- (2) 若当前运行的作业时间片到时，使当前执行的进程停止执行(抢占式多任务)，将其放入就绪队列中，它的当前优先级也恢复为初始优先级。这样，使得每个作业都有执行的机会，避免了低优先级的作业迟迟不能执行而“饿死”，而高优先级的作业将不断执行的情况发生

### (七) 作业调度程序的实现

#### ➤ 设定时钟

实验要求在一个时间片到期时进行调度。我们可以采用设定计时器的方法。Linux 操作系统为每一个进程提供了三个内部间隔定时器 (ITIMER\_REAL / ITIMER\_VIRTUAL / ITIMER\_PROF )

函数格式：

```
int setitimer(int which, struct itimerval *value, struct itimerval *ovalue);
struct itimerval {
    struct timeval it_interval;
    struct timeval it_value;
}
```

#### ➤ 调度进程与命令程序之间通过 FIFO 实现进程间通信

在实例程序中采用了 FIFO，调度程序负责创建一个 FIFO 文件，它每次从 FIFO 中读取用户提交的命令，而命令程序负责把命令按照 struct jobcmd 格式写进 FIFO 中。这样就实现了进程之间的通信

#### ➤ 作业切换

当一个作业运行到期时，如果一个新的作业具有一个最高优先级，那么就要调度它运行，原来的作业要放回到就绪队列当中。使用了信号

	<p>SIGSTOP 和 SIGCONT 控制一个进程运行，另一个进程停止，其中，SIGSTOP 信号默认的动作就是暂停进程的运行，而 SIGCONT 信号默认的动作是使停止的进程继续运行。</p> <p>➤ 从 FIFO 中读取数据</p> <p>1) 如果有进程写打开 FIFO，且当前 FIFO 内没有数据，则对于设置了阻塞标志的读操作来说，将一直阻塞。对于没有设置阻塞标志的读操作来说则返回-1，当前 errno 值为 EAGAIN，提醒以后再试。</p> <p>2) 对于设置了阻塞标志的读操作来说，造成阻塞的原因有两种：一种是当前 FIFO 内有数据，但有其它进程再读这些数据；另一种是 FIFO 内没有数据，阻塞原因是 FIFO 中有新的数据写入，而不论新写入数据量的大小，也不论读操作请求多少数据量。</p> <p>3) 读打开的阻塞标志只对本进程第一个读操作施加作用，如果本进程内有多个读操作序列，则在第一个读操作被唤醒并完成读操作后，其他将要执行的读操作将不再阻塞，即使在执行读操作时，FIFO 中没有数据也一样（此时，读操作返回 0）</p> <p>4) 如果没有进程写打开 FIFO，则设置了阻塞标志的读操作会阻塞</p> <p>5) 如果 FIFO 中有数据，则设置了阻塞标志的读操作不会因为 FIFO 中的字节数小于请求读的字节数而阻塞，此时，读操作会返回 FIFO 中现有的数据量。</p> <p>➤ 向 FIFO 中写入数据</p> <p>对于设置了阻塞标志的写操作：</p> <p>1) 当要写入的数据量不大于 PIPE_BUF 时，Linux 将保证写入的原子性。如果此时管道空闲缓冲区不足以容纳要写入的字节数，则进入睡眠，知道当缓冲区中能够容纳要写入的字节数时，才开始进行一次性写操作</p> <p>2) 当要写入的数据量大于 PIPE_BUF 时，Linux 将不再保证写入的原子性。FIFO 缓冲区一有空闲区域，写进程就会试图向管道写入数据，写操作在写完所有请求写的数据后返回。</p> <p>对于没有设置阻塞标志的写操作：</p> <p>1) 当要写入的数据量大于 PIPE_BUF 时，Linux 将不再保证写入的原子性。再写满所有 FIFO 空闲缓冲区后，写操作返回。</p> <p>2) 当要写入的数据量不大于 PIPE_BUF 时，Linux 将保证写入的原子性。如果当前 FIFO 空闲缓冲区能够容纳请求写入的字节数，写完后成功返回；如果当前 FIFO 空闲缓冲区不能够容纳请求写入的字节数，则返回 EAGAIN 错误，提醒以后再写。</p>
<p>解决问题所用的主要代码及描述（附注释）</p>	<p>一、job.h</p> <pre> //////////////////////////////////// //文件名:job.h //程序功能:公共头文件，完成必要的预处理 //日期:2010 年 6 月 15 日 //////////////////////////////////// #ifndef JOB_H #define JOB_H </pre>

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdarg.h>
#include <signal.h>
#include <sys/types.h>

#define FIFO "/home/brucezmf/桌面/zhumingfa/sever"

#ifndef DEBUG
#define DEBUG
#endif

#undef DEBUG

#define BUFLen 100
#define GLOBALFILE "screendump"

enum jobstate {
    READY, RUNNING, DONE
};

enum cmdtype {
    ENQ = -1, DEQ = -2, STAT = -3
};

/* this is data passed in fifo */
struct jobcmd {
    enum cmdtype type;
    int argnum;
    int owner;
    int defpri;
    char data[BUFLen];
};

#define DATALEN sizeof(struct jobcmd)
#define error_sys printf

struct jobinfo {
    int jid; /* job id */
    int pid; /* process id */
    char** cmdarg; /* the command & args to execute */
    int defpri; /* default priority */
    int curpri; /* current priority */
}
```

```

int    ownerid;          /* the job owner id */
int    wait_time;        /* the time job in waitqueue */
time_t create_time;      /* the time job create */
int    run_time;         /* the time job running */
enum   jobstate state;    /* job state */
};

struct waitqueue {        /* double link list */
    struct waitqueue *next;
    struct jobinfo *job;
};

void schedule();
void sig_handler(int sig, siginfo_t *info, void *notused);
int  allocjid();
void do_enq(struct jobinfo *newjob, struct jobcmd enqcmd);
void do_deq(struct jobcmd deqcmd);
void do_stat();
void updateall();
struct waitqueue* jobselect();
void jobswitch();
#endif

```

## 二、scheduler.c

```

////////////////////////////////////
//文件名:scheduler.c
//程序功能:作业调度程序, 负责整个系统的运行
//日期:2010 年 6 月 15 日
////////////////////////////////////
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <string.h>
#include <signal.h>
#include <fcntl.h>
#include <time.h>
#include "job.h"

int jobid = 0;
int siginfo = 1;
int fifo;

```

```

int globalfd;

struct waitqueue *head = NULL;
struct waitqueue *next = NULL, *current = NULL;

void schedule()
{
    struct jobinfo *newjob = NULL;
    struct jobcmd cmd;
    int count = 0;

    bzero(&cmd, DATALEN);
    if ((count = read(fifo, &cmd, DATALEN)) < 0)
        error_sys("read fifo failed");

#ifdef DEBUG

    if (count) {
        printf("cmd cmdtype\t%d\n"
               "cmd defpri\t%d\n"
               "cmd data\t%s\n",
               cmd.type, cmd.defpri, cmd.data);
    } // else
    // printf("no data read\n");

#endif

    switch (cmd.type) {
    case ENQ:
        do_enq(newjob, cmd);
        break;
    case DEQ:
        do_deq(cmd);
        break;
    case STAT:
        do_stat(cmd);
        break;
    default:
        break;
    }

    /* Update jobs in waitqueue */

    updateall();

```

```

/* select the highest priority job to run */

next = jobselect();

/* stop current job, run next job */

jobswitch();
}

int allocjid()
{
    return ++jobid;
}

void updateall()
{
    struct waitqueue *p;

    /* update running job's run_time */
    if (current)
        current->job->run_time += 1;    /* add 1 represent 100 ms */

    /* update ready job's wait_time */
    for (p = head; p != NULL; p = p->next) {
        p->job->wait_time += 100;

        if (p->job->wait_time >= 1000 && p->job->curpri < 3)
            p->job->curpri++;
    }
}

struct waitqueue* jobselect()
{
    struct waitqueue *p, *prev, *select, *selectprev;
    int highest = -1;

    select = NULL;
    selectprev = NULL;

    if (head) {
        for (prev = head, p = head; p != NULL; prev = p, p = p->next) {

            if (p->job->curpri > highest) {
                select = p;
            }
        }
    }
}

```



	<pre>                 selectprev = prev;                 highest = p-&gt;job-&gt;curpri;             }         }          selectprev-&gt;next = select-&gt;next;          if (select == selectprev) head = NULL;     }      return select; }  void jobswitch() {     struct waitqueue *p;     int i;      if (current &amp;&amp; current-&gt;job-&gt;state == DONE) {          /* current job finished */          /* job has been done, remove it */         for (i = 0; (current-&gt;job-&gt;cmdarg)[i] != NULL; i++) {             free((current-&gt;job-&gt;cmdarg)[i]);             (current-&gt;job-&gt;cmdarg)[i] = NULL;         }          free(current-&gt;job-&gt;cmdarg);         free(current-&gt;job);         free(current);          current = NULL;     }      if (next == NULL &amp;&amp; current == NULL)                    /* no job to run */          return;      else if (next != NULL &amp;&amp; current == NULL) {             /* start new job */          printf("begin start new job\n");         current = next;         next = NULL;         current-&gt;job-&gt;state = RUNNING; </pre>
--	--

	<pre> kill(current-&gt;job-&gt;pid, SIGCONT); return;  } else if (next != NULL &amp;&amp; current != NULL) { /* do switch */      kill(current-&gt;job-&gt;pid, SIGSTOP);     current-&gt;job-&gt;curpri = current-&gt;job-&gt;defpri;     current-&gt;job-&gt;wait_time = 0;     current-&gt;job-&gt;state = READY;      /* move back to the queue */      if (head) {         for (p = head; p-&gt;next != NULL; p = p-&gt;next);          p-&gt;next = current;      } else {         head = current;     }      current = next;     next = NULL;     current-&gt;job-&gt;state = RUNNING;     kill(current-&gt;job-&gt;pid, SIGCONT);      //printf("\nbegin switch: current jid=%d, pid=%d\n",     //        current-&gt;job-&gt;jid, current-&gt;job-&gt;pid);     return;  } else {    /* next == NULL &amp;&amp; current != NULL, no switch */      return; }  }  void sig_handler(int sig, siginfo_t *info, void *notused) {     int status;     int ret;      switch (sig) {     case SIGVTALRM:         schedule(); </pre>
--	---

```

        return;

    case SIGCHLD:
        ret = waitpid(-1, &status, WNOHANG);
        if (ret == 0 || ret == -1)
            return;

        if (WIFEXITED(status)) {
            #ifdef DEBUG
            //printf("%d %d %d\n", ret, info->si_pid, current->job->pid);
            //do_stat();
            #endif
            current->job->state = DONE;
            printf("normal termination, exit status = %d\tjid = %d, pid
= %d\n\n",
                WEXITSTATUS(status), current->job->jid, current->job->pid);

        } else if (WIFSIGNALED(status)) {
            printf("abnormal termination, signal number = %d\tjid = %d, pid
= %d\n\n",
                WTERMSIG(status), current->job->jid, current->job->pid);

        } else if (WIFSTOPPED(status)) {
            printf("child stopped, signal number = %d\tjid = %d, pid
= %d\n\n",
                WSTOPSIG(status), current->job->jid, current->job->pid);
        }
        return;

    default:
        return;
}

void do_enq(struct jobinfo *newjob, struct jobcmd enqcmd)
{
    struct    waitqueue *newnode, *p;
    int       i=0, pid;
    char *offset, *argvec, *q;
    char **arglist;
    sigset_t  zeromask;

    sigemptyset(&zeromask);

```

	<pre> /* fill jobinfo struct */  newjob = (struct jobinfo *)malloc(sizeof(struct jobinfo)); newjob-&gt;jid = allocjid(); newjob-&gt;defpri = enqcmd.defpri; newjob-&gt;curpri = enqcmd.defpri; newjob-&gt;ownerid = enqcmd.owner; newjob-&gt;state = READY; newjob-&gt;create_time = time(NULL); newjob-&gt;wait_time = 0; newjob-&gt;run_time = 0; arglist = (char**)malloc(sizeof(char*)*(enqcmd.argnum+1)); newjob-&gt;cmdarg = arglist; offset = enqcmd.data; argvec = enqcmd.data; while (i &lt; enqcmd.argnum) {      if (*offset == ':') {          *offset++ = '\0';         q = (char*)malloc(offset - argvec);         strcpy(q,argvec);         arglist[i++] = q;         argvec = offset;      } else         offset++;  }  arglist[i] = NULL;  #ifdef DEBUG  printf("enqcmd argnum %d\n",enqcmd.argnum); for (i = 0; i &lt; enqcmd.argnum; i++)     printf("parse enqcmd:%s\n",arglist[i]);  #endif  /* add new job to the queue */  newnode = (struct waitqueue*)malloc(sizeof(struct waitqueue)); newnode-&gt;next = NULL; newnode-&gt;job = newjob; </pre>
--	--

```

if (head) {
    for (p = head; p->next != NULL; p = p->next);

    p->next = newnode;
} else
    head = newnode;

/* create process for the job */

if ((pid = fork()) < 0)
    error_sys("enq fork failed");

/* In child process */

if (pid == 0) {

    newjob->pid = getpid();

    /* block the child wait for run */

    raise(SIGSTOP);

#ifdef DEBUG

    printf("begin running\n");
    for (i = 0; arglist[i] != NULL; i++)
        printf("arglist %s\n", arglist[i]);

#endif

    /* dup the globalfile descriptor to stdout */
    dup2(globalfd, 1);
    if (execv(arglist[0], arglist) < 0)
        printf("exec failed\n");

    exit(1);

} else {

    newjob->pid = pid;
    printf("\nnew job: jid=%d, pid=%d\n", newjob->jid, newjob->pid);

}

```

```

}

/* bug to fix */
void do_deq(struct jobcmd deqcmd)
{
    int deqid,i;
    struct waitqueue *p,*prev,*select,*selectprev;

    deqid = atoi(deqcmd.data);

#ifdef DEBUG
    printf("deq jid %d\n",deqid);
#endif

    /* current jodid == deqid, terminate current job */
    if (current && current->job->jid == deqid) {

        printf("terminate job: %d\n", current->job->jid);
        kill(SIGTERM, current->job->pid);

        for (i = 0; (current->job->cmdarg)[i] != NULL; i++) {

            free((current->job->cmdarg)[i]);
            (current->job->cmdarg)[i] = NULL;
        }

        free(current->job->cmdarg);
        free(current->job);
        free(current);

        current = NULL;

    } else { /* maybe in waitqueue, search it */
        select = NULL;
        selectprev = NULL;

        if (head) {
            for (prev = head, p = head; p != NULL; prev = p, p = p->next) {
                if (p->job->jid == deqid) {
                    select = p;
                    selectprev = prev;
                    break;
                }
            }
        }
    }
}

```

```

        selectprev->next = select->next;
        if (select == selectprev) head = NULL;
    }

    if (select) {
        for (i = 0; (select->job->cmdarg)[i] != NULL; i++) {
            free((select->job->cmdarg)[i]);
            (select->job->cmdarg)[i] = NULL;
        }

        free(select->job->cmdarg);
        free(select->job);
        free(select);

        select = NULL;
    }
}

void do_stat()
{
    /*
     * Print job statistics of all jobs:
     * 1. job id
     * 2. job pid
     * 3. job owner
     * 4. job run time
     * 5. job wait time
     * 6. job create time
     * 7. job state
     */

    struct waitqueue *p;
    char timebuf[BUFLen];

    printf( "JID\tPID\tOWNER\tRUNTIME\tWAITTIME\tCREATETIME\tSTATE\nJOB
NAME\tCURPRI\tDEFPRI\n");

    if (current) {
        strcpy(timebuf,ctime(&(current->job->create_time)));
        timebuf[strlen(timebuf) - 1] = '\0';
        printf("%d\t%d\t%d\t%d\t%d\t%s\t%s\n%s\t%d\t%d\n",
            current->job->jid,

```

```

        current->job->pid,
        current->job->ownerid,
        current->job->run_time,
        current->job->wait_time,
        timebuf,
        "RUNNING",
        *(current->job->cmdarg),
        current->job->curpri,
        current->job->defpri
    );
}

for (p = head; p != NULL; p = p->next) {
    strcpy (timebuf,ctime(&(p->job->create_time)));
    timebuf[strlen(timebuf) - 1] = '\0';
    printf("%d\t%d\t%d\t%d\t%d\t%s\t%s\n%s\t%d\t%d\n",
        p->job->jid,
        p->job->pid,
        p->job->ownerid,
        p->job->run_time,
        p->job->wait_time,
        timebuf,
        "READY" ,
        *(current->job->cmdarg),
        p->job->curpri,
        p->job->defpri
    );
}

printf("\n");
}

int main()
{
    struct timeval interval;
    struct itimerval new,old;
    struct stat statbuf;
    struct sigaction newact,oldact1,oldact2;

    if (stat(FIFO,&statbuf) == 0) {

        /* if fifo file exists, remove it */

        if (remove(FIFO) < 0)

```



```
        error_sys("remove failed");
    }

    if (mkfifo(FIFO,0666) < 0)
        error_sys("mkfifo failed");

    /* open fifo in nonblock mode */

    if ((fifo = open(FIFO,O_RDONLY|O_NONBLOCK)) < 0)
        error_sys("open fifo failed");

    /* open global file for job output */

    if ((globalfd = open("/dev/null",O_WRONLY)) < 0)
        error_sys("open global file failed");

    /* setup signal handler */
    newact.sa_sigaction = sig_handler;
    sigemptyset(&newact.sa_mask);
    newact.sa_flags = SA_SIGINFO;

    sigaction(SIGCHLD,&newact,&oldact1);
    sigaction(SIGVTALRM,&newact,&oldact2);

    /* timer interval: 0s, 100ms */

    interval.tv_sec = 0;
    interval.tv_usec = 100;

    new.it_interval = interval;
    new.it_value = interval;
    setitimer(ITIMER_VIRTUAL,&new,&old);

    printf("OK! Scheduler is starting now!!\n");

    while (siginfo == 1);

    close(fifo);
    close(globalfd);
    return 0;
}
```

### 三、myjob.c

////////////////////////////////////

//文件名:myjob.c

//程序功能:新建作业

//日期:2010 年 6 月 15 日

////////////////////////////////////

#include <stdio.h>

int main(int argc,char \*argv[]){

int i=0;

for(i=0;i<argc;i++){

printf("%s\n",argv[i]);

}

sleep(404);//休息 404 秒

return 0;

}

### 四、enq.c

////////////////////////////////////

//文件名:enq.c

//程序功能:提交作业

//日期:2010 年 6 月 15 日

////////////////////////////////////

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/ipc.h>

#include <fcntl.h>

#include "job.h"

/\*

\* command syntax

\* enq [-p num] e\_file args

\*/

void usage()

{

printf("Usage: enq [-p num] e\_file args\n"

"\t-p num\t\t specify the job priority\n"

"\te\_file\t\t the absolute path of the exefile\n"

"\targs\t\t the args passed to the e\_file\n");

}

int main(int argc,char \*argv[])

{

```

int p = 0;
int fd;
char c, *offset;
struct jobcmd enqcmd;

if (argc == 1) {
    usage();
    return 1;
}

while (--argc > 0 && (*++argv)[0] == '-') {

    while ((c = *++argv[0]))
        switch (c) {
            case 'p':
                p = atoi(*++argv);
                argc--;
                break;
            default:
                printf("Illegal option %c\n", c);
                return 1;
        }
}

if (p < 0 || p > 3) {
    printf("invalid priority: must between 0 and 3\n");
    return 1;
}

enqcmd.type = ENQ;
enqcmd.defpri = p;
enqcmd.owner = getuid();
enqcmd.argnum = argc;
offset = enqcmd.data;

while (argc-- > 0) {
    strcpy(offset, *argv);
    strcat(offset, ":");
    offset = offset + strlen(*argv) + 1;
    argv++;
}

#ifdef DEBUG
printf("enqcmd cmdtype\t%d\n"

```

```

"enqcmd owner\t%d\n"
"enqcmd defpri\t%d\n"
"enqcmd data\t%s\n",
enqcmd.type, enqcmd.owner, enqcmd.defpri, enqcmd.data);
#endif

if ((fd = open(FIFO,O_WRONLY)) < 0)
    error_sys("enq open fifo failed");

if (write(fd,&enqcmd,DATALEN)< 0)
    error_sys("enq write failed");

close(fd);
return 0;
}

```

### 五、stat.c

```

////////////////////////////////////
//文件名:stat.c
//程序功能:查看状态信息
//日期:2010 年 6 月 15 日
////////////////////////////////////
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <fcntl.h>
#include <string.h>
#include "job.h"

/*
 * command syntax
 *      stat
 */
void usage()
{
    printf ("Usage: stat\n");
}

int main (int argc,char *argv[])
{
    struct jobcmd statcmd;
    int fd;

```

```

    if (argc !=1) {
        usage();
        return 1;
    }

    statcmd.type = STAT;
    statcmd.defpri = 0;
    statcmd.owner = getuid();
    statcmd.argnum = 0;
    strcpy(statcmd.data, "\0");

    if ((fd = open(FIFO,O_WRONLY)) < 0 )
        error_sys("stat open fifo failed");

    if (write(fd,&statcmd,DATALEN) < 0)
        error_sys("stat write failed");

    close (fd);
    return 0;
}

```

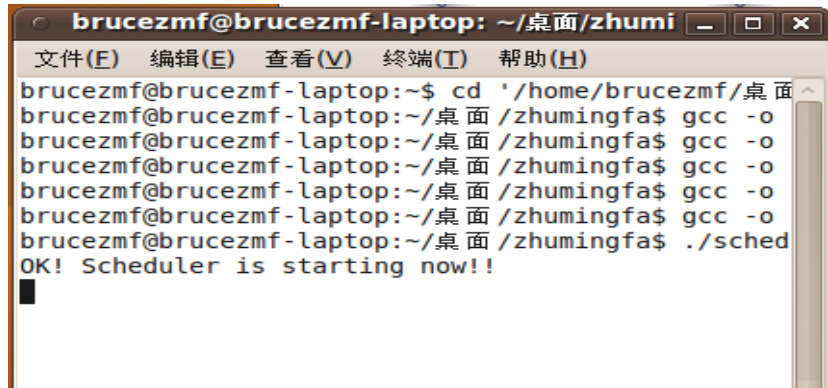
## 六、deq.c

```

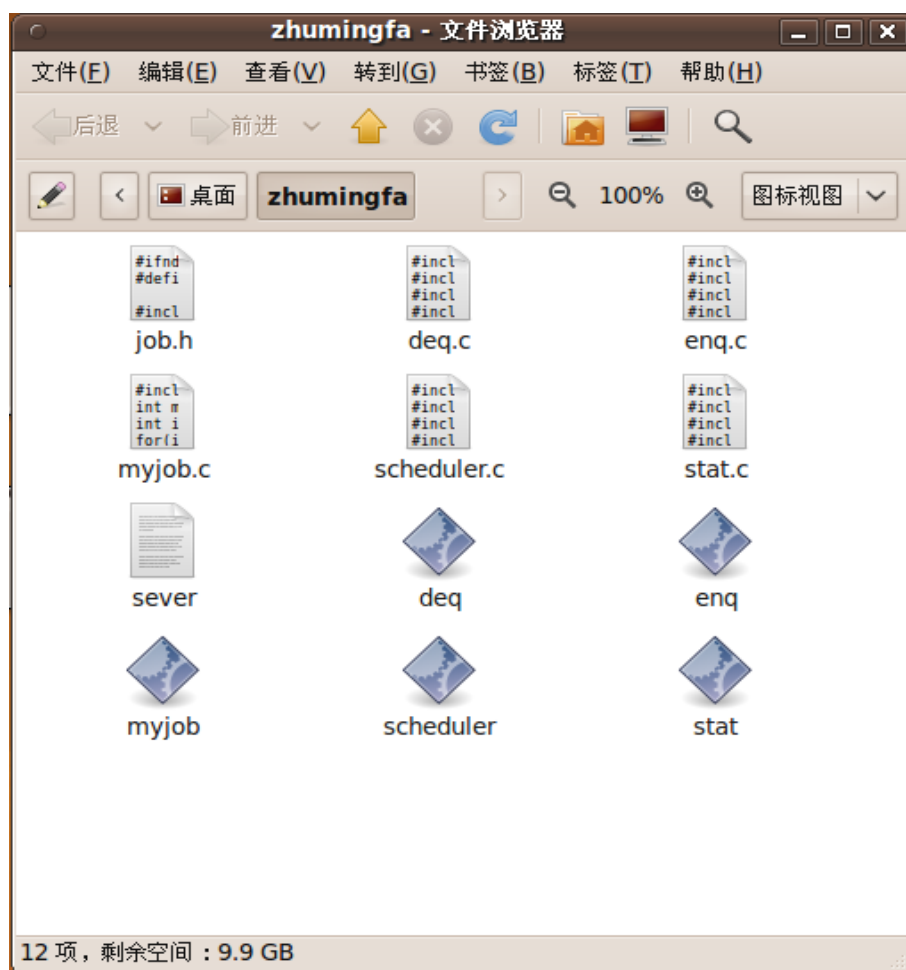
////////////////////////////////////
//文件名:deq.c
//程序功能:删除作业
//日期:2010 年 6 月 15 日
////////////////////////////////////
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <fcntl.h>
#include "job.h"

/*
 * command syntax
 *      deq jid
 */
void usage()
{
    printf("Usage:  deq jid\n"

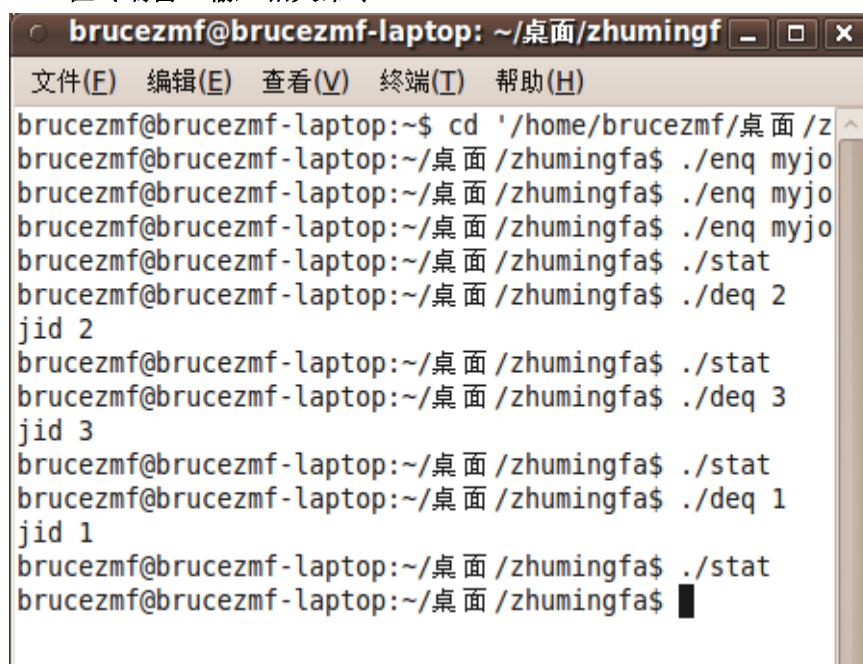
```

	<pre>         "\tjid\t\t the job id\n");     }  int main(int argc,char *argv[]) {     struct jobcmd deqcmd;     int fd;      if (argc != 2) {         usage();         return 1;     }      deqcmd.type = DEQ;     deqcmd.defpri = 0;     deqcmd.owner = getuid();     deqcmd.argnum = 1;      strcpy(deqcmd.data,*++argv);     printf("jid %s\n",deqcmd.data);      if ((fd = open(FIFO,O_WRONLY)) &lt; 0)         error_sys("deq open fifo failed");      if (write(fd,&amp;deqcmd,DATALEN)&lt; 0)         error_sys("deq write failed");      close(fd);     return 0; } </pre>
<p>实验结果分析(或错误原因分析)</p>	<p>一、编译所有文件并运行 scheduler</p>  <pre> brucezmf@brucezmf-laptop: ~/桌面/zhumi 文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H) brucezmf@brucezmf-laptop:~\$ cd '/home/brucezmf/桌面 brucezmf@brucezmf-laptop:~/桌面/zhumingfa\$ gcc -o brucezmf@brucezmf-laptop:~/桌面/zhumingfa\$ gcc -o brucezmf@brucezmf-laptop:~/桌面/zhumingfa\$ gcc -o brucezmf@brucezmf-laptop:~/桌面/zhumingfa\$ gcc -o brucezmf@brucezmf-laptop:~/桌面/zhumingfa\$ gcc -o brucezmf@brucezmf-laptop:~/桌面/zhumingfa\$ ./sched OK! Scheduler is starting now!! </pre>

## 二、文件包变化



## 三、在终端窗口输入相关命令



#### 四、scheduler 运行窗口变化

```

brucezmf@brucezmf-laptop: ~$ cd '/home/brucezmf/桌面/z
brucezmf@brucezmf-laptop:~/桌面/zhumingfa$ gcc -o myj
brucezmf@brucezmf-laptop:~/桌面/zhumingfa$ gcc -o enq
brucezmf@brucezmf-laptop:~/桌面/zhumingfa$ gcc -o deq
brucezmf@brucezmf-laptop:~/桌面/zhumingfa$ gcc -o sta
brucezmf@brucezmf-laptop:~/桌面/zhumingfa$ gcc -o sch
brucezmf@brucezmf-laptop:~/桌面/zhumingfa$ ./schedule
OK! Scheduler is starting now!!

new job: jid=1, pid=5365
begin start new job

new job: jid=2, pid=5369

new job: jid=3, pid=5373
JID      PID      OWNER    RUNTIME  WAITTIME      CR
EATTIME  STATE
JOBNAME  CURPRI    DEFPRI
2         5369      1000     1394     100           Thu Jun 17
01:47:41 2010    RUNNING
myjob    0
1         5365      1000     4161     0             Thu Jun 17
01:47:18 2010    READY
myjob    0
3         5373      1000     1050     600           Thu Jun 17
01:47:50 2010    READY
myjob    0

JID      PID      OWNER    RUNTIME  WAITTIME      CR
EATTIME  STATE
JOBNAME  CURPRI    DEFPRI
1         5365      1000     7147     100           Thu Jun 17
01:47:18 2010    RUNNING
myjob    0

JID      PID      OWNER    RUNTIME  WAITTIME      CR
EATTIME  STATE
JOBNAME  CURPRI    DEFPRI
1         5365      1000     13719    100           Thu Jun 17
01:47:18 2010    RUNNING
myjob    0

terminate job: 1
JID      PID      OWNER    RUNTIME  WAITTIME      CR
EATTIME  STATE
JOBNAME  CURPRI    DEFPRI

```

小结

通过本次实验加深了对操作系统中调度概念及调度算法的理解，了解了一些 Linux 下进程控制以及进程之间通信的相关知识。  
对操作系统中调度、协调和控制各作业对 CPU 的使用原理，有了进一步的理解。

指导老  
师意见