第二章　实验二

Windows 进程的控制

1. 创建进程

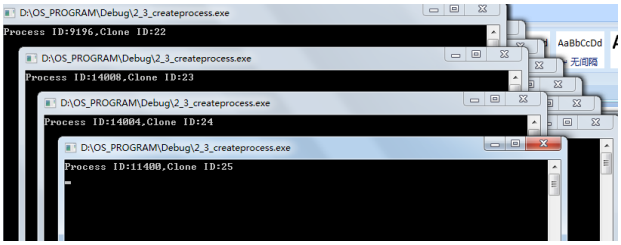
- WIN32API 函数 CreateProcess 用来创建一个新的进程和它的主线程，这个新进程运行指定的可执行文件。CreateProcess()调用：创建一个进程。
- 使用 Micrsoft Visual Studio C++ 6.0 编程程序 2_3_createprocess.cpp。基本功能：创建传递过来进程的克隆 25 个进程并赋予其 ID 值。
- 输入如下程序代码：

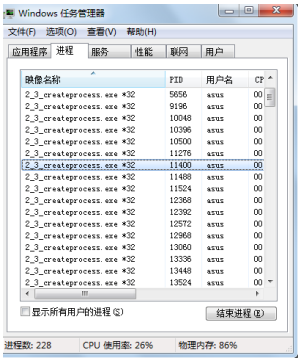
```
#include<windows.h>
#include<iostream>
#include<stdio.h>
//创建传递过来进程的克隆进程并赋予其 ID 值
void StartClone(int nCloneID)
{
    //提取用于当前可执行文件的文件名
    TCHAR szFilename[MAX_PATH];
    ::GetModuleFileName(NULL,szFilename,MAX_PATH);
    //格式化用于子进程的命令行并通知其 EXE 文件名和克隆 ID
    TCHAR szCmdLine[MAX_PATH];
    ::sprintf(szCmdLine, "\"%s\"%d", szFilename, nCloneID);
    //用于子进程的 STARTUPINFO 结构
    STARTUPINFO si;
    ::ZeroMemory(reinterpret_cast <void*>(&si),sizeof(si));
    si.cb = sizeof(si);                    //必须是本结构的大小
    //返回用于子进程的进程信息
    PROCESS_INFORMATION pi;
    //利用同样的可执行文件和命令行创建进程，并赋予其子进程的性质
    BOOL bCreateOK =:: CreateProcess(
        szFilename,              //产生这个 EXE 的应用程序的名称
        szCmdLine,               //告诉其行为像一个子进程的标志
        NULL,                    //默认的进程安全性
        NULL,                    //默认的线程安全性
        FALSE,                   //不继承句柄
        CREATE_NEW_CONSOLE, //使用新的控制台
        NULL,                    //新的环境
        NULL,                    //当前目录
        &si,                     //启动信息
        &pi                      //返回的进程信息
    );
    //对子进程释放引用
    if(bCreateOK)
    {
```

```
            ::CloseHandle(pi.hProcess);
            ::CloseHandle(pi.hThread);
        }
}

void main(int argc, char *argv[])
{
        //确定进程在列表中的位置
        int nClone(0);
        if(argc>1)
        {    //从第二参数中提取克隆 ID
            ::sscanf(argv[1], "%d", &nClone);
        }
        //显示进程位置
        std::cout<<"Process ID:"<<::GetCurrentProcessId()
                    <<",Clone ID:"<<nClone
                    <<std::endl;
        //检查是否有创建子进程的需要
        const int c_nCloneMax = 25;     //准备复制 25 个子进程
        if(nClone<c_nCloneMax)
        {    //发送新进程的命令行和克隆号
            StartClone(++nClone);
        }
        //在终止之前暂停二分之一秒
        ::Sleep(500);
        getchar();
}
```

运行结果：





2. 终止进程

- WIN32API 函数 ExitProcess 用来终止一个进程。
- 使用 Micrsoft Visual Studio C++ 6.0 编程程序 2_4_killprocess.cpp。基本功能：
- 输入如下程序代码：

```
#include<windows.h>
#include<iostream>
#include<stdio.h>
static LPCTSTR g_szMutexName = "w2kdg.ProcTerm.mutex.Suicide";
//创建当前进程的克隆进程的简单方法
void StartClone()
{
    //提取用于当前可执行文件的文件名
    TCHAR szFilename[MAX_PATH];
    ::GetModuleFileName(NULL,szFilename,MAX_PATH);
    //格式化用于子进程的命令行，指明它是一个 EXE 文件和子进程
    TCHAR szCmdLine[MAX_PATH];
    ::sprintf(szCmdLine, "\"%s\"child", szFilename);
    //子进程的 STARTUPINFO 启动信息结构
    STARTUPINFO si;
    ::ZeroMemory(reinterpret_cast <void*>(&si),sizeof(si));
    si.cb = sizeof(si);                    //必须是本结构的大小
    //返回用于子进程的进程信息
    PROCESS_INFORMATION pi;
    //利用同样的可执行文件和命令行创建进程，并指明它是一个子进程
    BOOL bCreateOK =:: CreateProcess(
        szFilename,              //产生这个 EXE 的应用程序的名称
        szCmdLine,               //告诉用户这个子进程的标志
        NULL,                    //默认的进程安全性
        NULL,                    //默认的线程安全性
        FALSE,                   //不继承句柄
        CREATE_NEW_CONSOLE, //使用新的控制台(创建新窗口)
        NULL,                    //新的环境
        NULL,                    //当前目录
        &si,                     //启动信息
        &pi                      //返回的进程信息
    );
    //释放指向子进程的引用
    if(bCreateOK)
    {
        ::CloseHandle(pi.hProcess);
        ::CloseHandle(pi.hThread);
    }
}
```

```
void Parent()
{
    //创建“自杀”互斥程序体
    HANDLE hMutexSuicide = :: CreateMutex(
        NULL,                    //默认的安全性
        TRUE,                    //最初拥有的
        g_szMutexName);          //为其命名
    if(hMutexSuicide !=NULL)
    {
        //创建子进程
        std::cout<<"Creating the child prcess."<<std::endl;
        ::StartClone();
        //暂停
        ::Sleep(5000);
        //指令子进程“杀”掉自身
        std::cout<<"Telling the child process to quit."<<std::endl;
        ::ReleaseMutex(hMutexSuicide);
        //消除句柄
        ::CloseHandle(hMutexSuicide);
    }
}

void Child()
{
    //打开“自杀”互斥体
    HANDLE hMutexSuicide =:: OpenMutex(
        SYNCHRONIZE,              //打开用于同步
        FALSE,                   //不需要向下传递
        g_szMutexName);           //名称
    if(hMutexSuicide != NULL)
    {
        //报告用户正在等待指令
        std::cout<<"Child waiting for suicide instruction."<<std::endl;
        ::WaitForSingleObject(hMutexSuicide, INFINITE);
        //准备好终止，清除句柄
        std::cout<<"Child quiting."<<std::endl;
        ::CloseHandle(hMutexSuicide);
        ::Sleep(1000);
    }
}

int main(int argc, char * argv[])
{
    //决定其行为是父进程还是子进程
```

```
    if(argc>1 && strcmp(argv[1],"child")==0)
    {
        Child();
    }
    else
    {
        Parent();
    }
    return 0;
    getchar();
}
```

3. 其它进程控制

参考“操作系统-多媒体资料”文件里 ebook

进入：实验 3 进程控制与描述

3.2 windows 2000 编程

3.2 windows 2000 进程的“一生”