在所有现代Windows操作系统中，系统的环境变量保存在这两个注册表键中的注册表值：  
系统级：HKEY\_LOCAL\_MACHINE\\System\\CurrentControlSet\\Control\\Session Manager\\Environment  
用户级：HKEY\_CURRENT\_USER\\Environment  
添加、修改注册表值，最方便的方法是使用RegSetKeyValue。删除注册表值可以用RegDeleteKeyValue。如果需要兼容Windows XP等老系统，需要使用RegOpenKeyEx-RegSetValueEx/RegDeleteValue-RegCloseKey的方式。  
  
修改了这两个地方，不会立即生效，还需要通知explorer.exe环境变量已经改变，需要重新读取注册表。方法是向所有窗口（HWND\_BROADCAST）发送WM\_SETTINGCHANGE消息，wParam为0，lParam为(LPARAM)TEXT("Environment")。向窗口发送消息可以用SendMessage函数，但为了避免受到挂起的窗口影响，这里一般是使用SendMessageTimeout或SendNotifyMessage函数代替，但不能使用PostMessage，这是因为这个消息需要直接调用WndProc，不能被放到消息队列中。  
  
不过，程序自身的环境块是相对独立，并且继承自父进程的。如果需要根据注册表生成环境块以供CreateProcess使用，需要使用CreateEnvironmentBlock，它还需要一个用户token，可以用OpenProcessToken获取。注意这样获取的环境块只能是Unicode（wchar\_t）的，如果需要ANSI（char）的，需要自己转换，这也意味着，CreateProcess时需要指定CREATE\_UNICODE\_ENVIRONMENT开关。用完以后，使用DestroyEnvironmentBlock释放。

#include <tchar.h>

#include <stdio.h>

#include <userenv.h>

#pragma comment(lib, "userenv.lib")

int \_tmain(int argc, TCHAR \*\*argv)

{

// 设置(删除)用户级(系统级)环境变量，并通知explorer.exe读取注册表环境变量

TCHAR envstr[] = \_T("My Environment String");

RegSetKeyValue(HKEY\_CURRENT\_USER, \_T("Environment"), \_T("myenv"), REG\_SZ, envstr, lstrlen(envstr) \* sizeof (TCHAR));

//RegDeleteKeyValue(HKEY\_CURRENT\_USER, \_T("Environment"), \_T("myenv"));

RegSetKeyValue(HKEY\_LOCAL\_MACHINE, \_T("System\\CurrentControlSet\\Control\\Session Manager\\Environment"),

\_T("mysysenv"), REG\_SZ, envstr, lstrlen(envstr) \* sizeof (TCHAR));

SendNotifyMessage(HWND\_BROADCAST, WM\_SETTINGCHANGE, 0, (LPARAM)\_T("Environment"));

// 根据NT系统的userenv.dll生成Unicode环境块

HANDLE htoken = NULL;

OpenProcessToken(GetCurrentProcess(), TOKEN\_QUERY, &htoken);

LPVOID envblock = NULL;

CreateEnvironmentBlock(&envblock, htoken, FALSE);

CloseHandle(htoken);

// 以这个环境块运行cmd /k set以查看结果

TCHAR cmdline[1024] = \_T("cmd /k set");

STARTUPINFO si = { sizeof si };

PROCESS\_INFORMATION pi = { 0 };

CreateProcess(NULL, cmdline, NULL, NULL, FALSE, CREATE\_UNICODE\_ENVIRONMENT, envblock, NULL, &si, &pi);

CloseHandle(pi.hThread);

// 等待程序结束

WaitForSingleObject(pi.hProcess, INFINITE);

CloseHandle(pi.hProcess);

// 销毁环境块

DestroyEnvironmentBlock(envblock);

return 0;

}