




# GWU2U Driver (Windows VCP)

## 用户指南

UG1007-1.1, 2021-12-23

版权所有 © 2021 广东高云半导体科技股份有限公司

 GOWIN高云, Gowin, 高云均为广东高云半导体科技股份有限公司注册商标, 本手册中提到的其他任何商标, 其所有权利属其所有者所有。未经本公司书面许可, 任何单位和个人都不得擅自摘抄、复制、翻译本文档内容的部分或全部, 并不得以任何形式传播。

### 免责声明

本文档并未授予任何知识产权的许可, 并未以明示或暗示, 或以禁止发言或其它方式授予任何知识产权许可。除高云半导体在其产品的销售条款和条件中声明的责任之外, 高云半导体概不承担任何法律或非法律责任。高云半导体对高云半导体产品的销售和 / 或使用不作任何明示或暗示的担保, 包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等, 均不作担保。高云半导体对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任, 高云半导体保留修改文档中任何内容的权利, 恕不另行通知。高云半导体不承诺对这些文档进行适时的更新。

## 版本信息

日期	版本	说明
2021/06/29	1.0	初始版本。
2021/12/23	1.1	更新第 3 章 GWU2U 虚拟串口编程指南。

# 目录

目录 .....	i
图目录.....	ii
表目录.....	iii
<b>1 GWU2U 的虚拟串口驱动程序.....</b>	<b>1</b>
<b>2 驱动程序的卸载 .....</b>	<b>3</b>
<b>3 GWU2U 虚拟串口编程指南 .....</b>	<b>5</b>
3.1 打开虚拟串口设备 .....	5
3.2 超时设置.....	7
3.3 参数设置.....	9
3.4 清除串口缓冲 .....	10
3.5 清除串口错误 .....	11
3.6 通过虚拟串口设备发送数据（同步） .....	11
3.7 通过虚拟串口设备接收数据（同步） .....	12
3.8 异步读写串口 .....	13
<b>术语、缩略语.....</b>	<b>23</b>
<b>技术支持与反馈 .....</b>	<b>24</b>

# 图目录

图 1-1 列出所有设备 .....	1
图 1-2 选择需要安装驱动的设备 .....	2
图 1-3 选择要安装的驱动程序 .....	2
图 2-1 打开设备管理器 .....	3
图 2-2 卸载设备 .....	4

# 表目录

表 A-1 术语、缩略语 .....	23
--------------------	----

# 1 GWU2U 的虚拟串口驱动程序

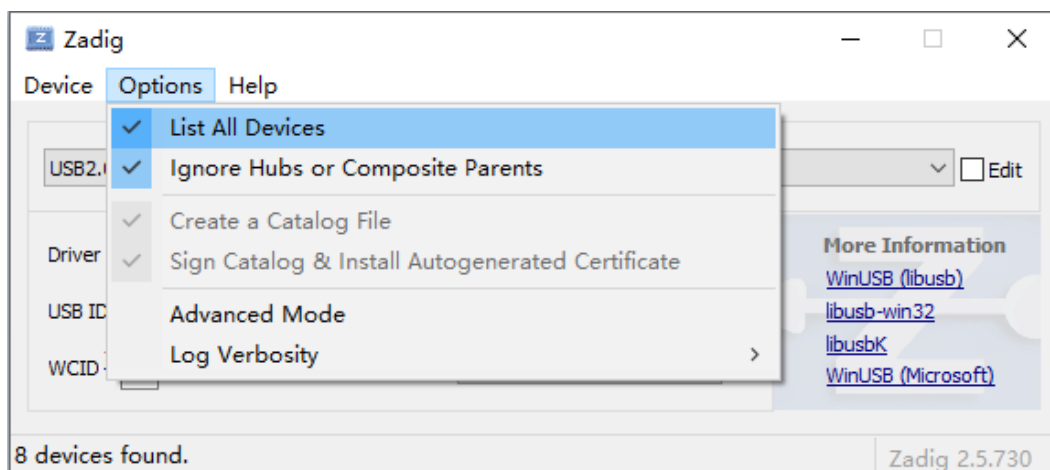
GWU2U 可使用 USB 虚拟串口驱动（usbser.sys）进行编程操作，实现数据的收发，以及波特率、校验位、停止位、字节位宽等参数的设置。

安装驱动，可使用 Gowin 提供的驱动安装工具，亦可使用开源驱动安装工具 Zadig（<https://zadig.akeo.ie/>）。安装驱动需要使用管理员权限。

## 使用 Zadig 安装驱动程序

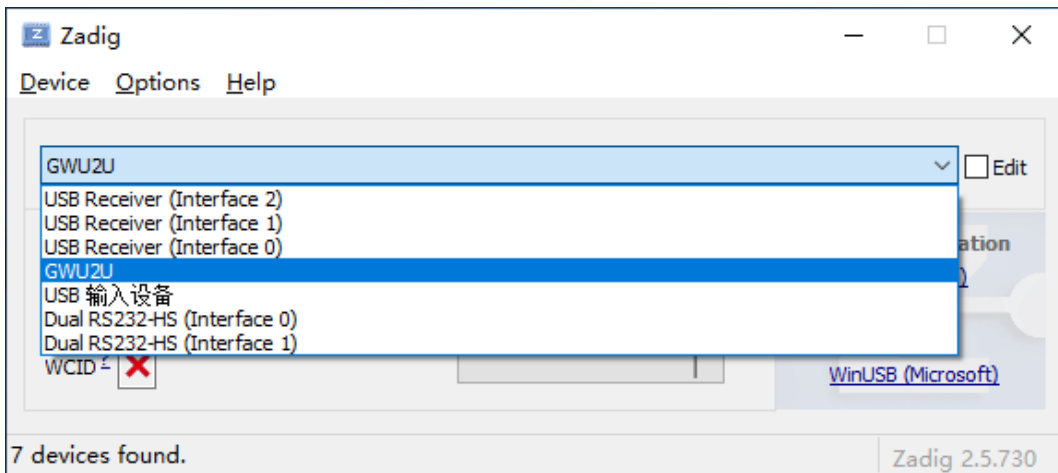
连接 GWU2U 设备到电脑 USB 接口，双击打开 Zadig（需要管理员权限），点击 Options，勾选“List All Device”选项，此时会列举出所有连接到电脑的 USB 设备。

图 1-1 列出所有设备



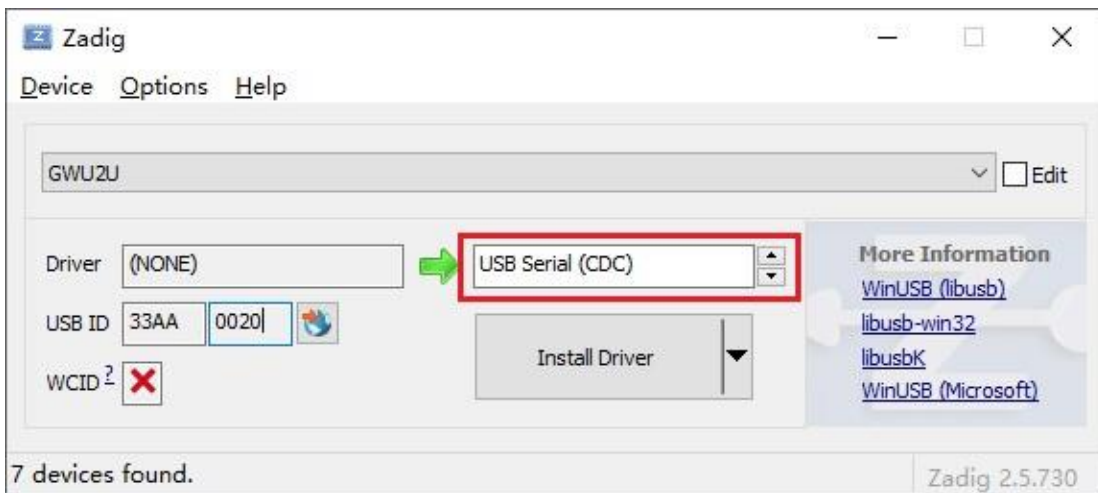
选择需要安装驱动的设备 Gowin U2U。

图 1-2 选择需要安装驱动的设备



选择要安装的驱动程序，使用虚拟串口(VCP)的形式，请选择 USB Serial (CDC)。

图 1-3 选择要安装的驱动程序



点击“Install Driver”按钮安装驱动。稍等片刻即可完成驱动安装。

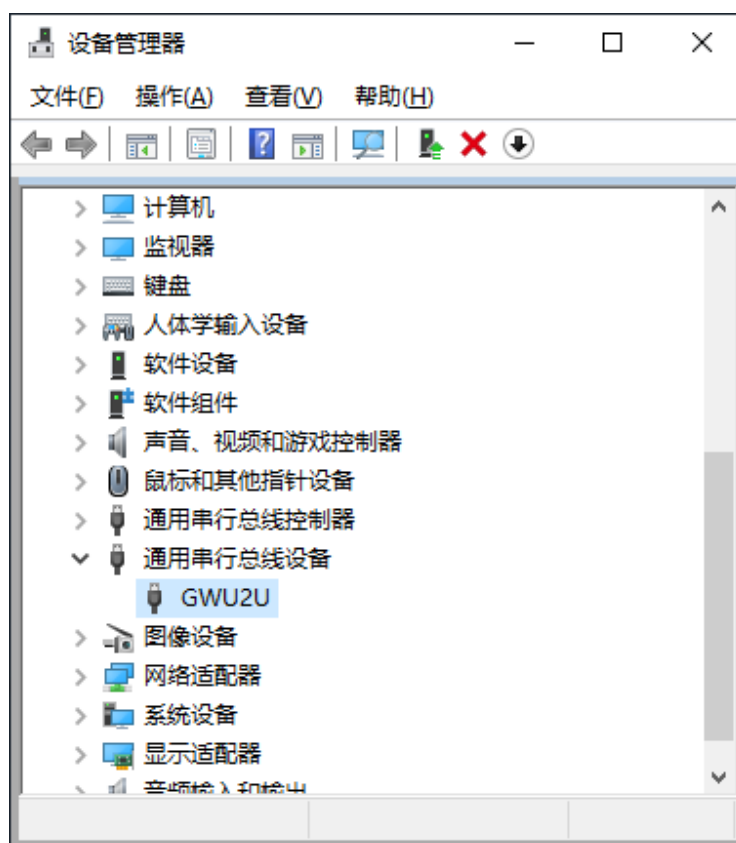
需要注意的是，若当前未安装驱动，该按钮显示为“Install Driver”，若当前安装了其他驱动，则显示为“Replace Driver”。



# 2 驱动程序的卸载

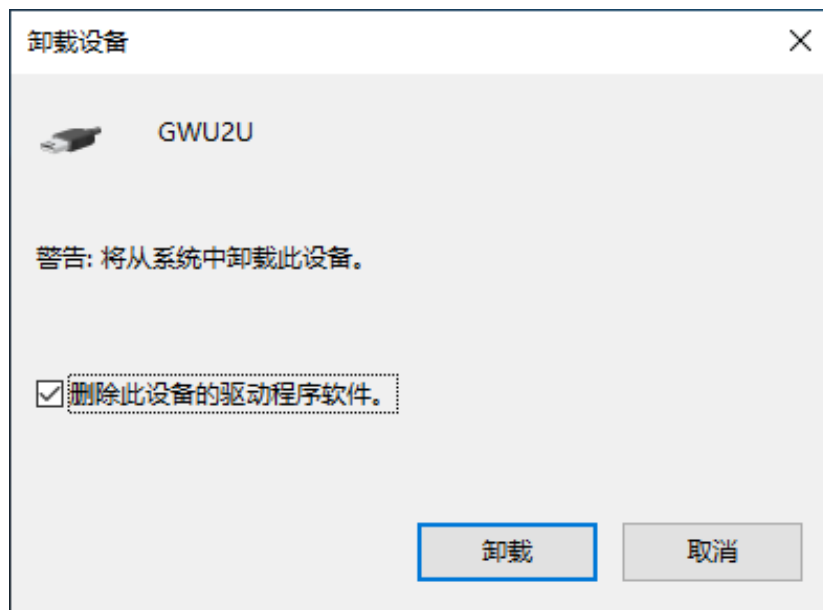
卸载驱动程序时，连接 GWU2U 设备到电脑，打开 windows 设备管理器，在“通用串行总线设备”列表中，找到 Gowin USB Serial 设备。右单击该设备名称，在弹出菜单中选择“卸载设备”选项。

图 2-1 打开设备管理器



在弹出的对话框中，先勾选“删除此设备的驱动程序软件”，再点击卸载按钮，即可卸载驱动程序。

图 2-2 卸载设备



# 3 GWU2U 虚拟串口编程指南

使用 Windows 的库函数即可对 GWU2U 的虚拟串口设备进行操作。编程时，需包含头文件<windows.h>。

## 3.1 打开虚拟串口设备

使用 windows 库函数 `CreateFile` () 即可打开虚拟串口设备，并返回一个该串口的句柄。其函数接口定义如下：

```
HANDLE WINAPI CreateFile(  
    _In_ LPCTSTR lpFileName, //要打开或创建的文件名  
    _In_ DWORD dwDesiredAccess, //访问类型  
    _In_ DWORD dwShareMode, //共享方式  
    _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes, //  
    安全属性  
    _In_ DWORD dwCreationDisposition, //指定要打开的文件已存在  
    或不存在的动作  
    _In_ DWORD dwFlagsAndAttributes, //文件属性和标志  
    _In_opt_ HANDLE hTemplateFile //一个指向模板文件的句柄  
);
```

函数参数的定义如下：

**lpFileName:** 要打开或创建的文件名。

**dwDesiredAccess:** 访问方式：

- 0 为设备查询访问方式
- `GENERIC_READ` 为读访问
- `GENERIC_WRITE` 为写访问

**dwShareMode:** 共享方式:

- 0 表示文件不能被共享，其它打开文件的操作都会失败
- FILE\_SHARE\_READ 表示允许其它读操作
- FILE\_SHARE\_WRITE 表示允许其它写操作
- FILE\_SHARE\_DELETE 表示允许其它删除操作

**lpSecurityAttributes:** 安全属性。一个指向 SECURITY\_ATTRIBUTES 结构的指针。

**dwCreationDisposition:** 创建或打开文件时的动作:

- OPEN\_ALWAYS: 打开文件，如果文件不存在则创建它
- TRUNCATE\_EXISTING 打开文件，且将文件清空（故需要 GENERIC\_WRITE 权限），如果文件不存在则会失败
- OPEN\_EXISTING 打开文件，文件若不存在则会失败
- CREATE\_ALWAYS 创建文件，如果文件已存在则清空
- CREATE\_NEW 创建文件，如文件存在则会失败

**dwFlagsAndAttributes:** 文件标志属性:

- FILE\_ATTRIBUTE\_NORMAL 常规属性
- FILE\_FLAG\_OVERLAPPED 异步 I/O 标志，如果不指定此标志则默认为同步 IO
- FILE\_ATTRIBUTE\_READONLY 文件为只读
- FILE\_ATTRIBUTE\_HIDDEN 文件为隐藏
- FILE\_FLAG\_DELETE\_ON\_CLOSE 所有文件句柄关闭后文件被删除
- 其它标志和属性请参考微软官方文档

**hTemplateFile:** 一个文件的句柄，且该文件必须是以 GENERIC\_READ 访问方式打开的。如果此参数不是 NULL，则会使用 hTemplateFile 关联的文件的属性和标志来创建文件。如果是打开一个现有文件，则该参数被忽略。

使用 CreateFile()打开串口时需要注意的是:

- lpFileName 文件名，如果是 COM10 以下的串口，则直接写串口号名，如“COM1”；如果是 COM10 及以上的串口，则串口名格式应为：[\\\\.\\COM10](#)；
- dwShareMode 共享方式应为 0，即串口应为独占方式；
- dwCreationDisposition 打开时的动作应为 OPEN\_EXISTING，即串口必须存在。

关闭串口时，调用 CloseHandle()函数关闭串口，函数的参数为串口句柄。

示例如下:

```
HANDLE hCom_1 =  
CreateFile (  
    ComName,  
    GENERIC_READ | GENERIC_WRITE,  
    0,  
    NULL,  
    OPEN_EXISTING,  
    FILE_FLAG_OVERLAPPED, //表示以 OVERLAPPED 模式打开串口，进  
    行异步传输，若使用同步传输则使用参数 0  
    NULL  
); // 打开串口设备  
  
if (hCom_1 == INVALID_HANDLE_VALUE)  
{  
    int a = GetLastError();  
    printf("Error: %d\n", a);  
    return -1;  
}  
  
CloseHandle(hCom_1); // 关闭串口设备
```

## 3.2 超时设置

在调用 `ReadFile()`和 `WriteFile()`读写串口的时候，如果没有指定异步操作的话，读写都会一直等待指定大小的数据，这时候我们可能想要设置一个读写的超时时间。

调用 `SetCommTimeouts()`可以设置串口读写超时时间，`GetCommTimeouts()`可以获得当前的超时设置，一般先利用 `GetCommTimeouts` 获得当前超时信息到一个 `COMMTIMEOUTS` 结构，然后对这个结构自定义，再调用 `SetCommTimeouts()`进行设置。

`COMMTIMEOUTS` 的成员变量介绍如下：

**ReadIntervalTimeout:** 两字符之间最大的延时，当读取串口数据时，一旦两个字符传输的时间差超过该时间，读取函数将返回现有的数据。设置为 0 表示该参数不起作用。指定通讯线上两个字符到达的最大时延，以毫秒为单位。

在 **ReadFile** 操作期间，时间周期从第一个字符接收到算起。如果收到的两个字符之间的间隔超过该值，**ReadFile** 操作完毕并返回所有缓冲数据。如果 **ReadIntervalTimeout** 为 0，则该值不起作用。如果值为 **MAXDWORD**，并且 **ReadTotalTimeoutConstant** 和 **ReadTotalTimeoutMultiplier** 两个值都为 0，则指定读操作携带已经收到的字符立即返回，即使没有收到任何字符。

**ReadTotalTimeoutMultiplier**: 读取每字符间的超时。指定以毫秒为单位的累积值。用于计算读操作时的超时总数。对于每次读操作，该值与所要读的字节数相乘。

**ReadTotalTimeoutConstant**: 一次读取串口数据的固定超时。所以在一次读取串口的操作中，其超时为 **ReadTotalTimeoutMultiplier** 乘以读取的字节数再加上 **ReadTotalTimeoutConstant**。将 **ReadIntervalTimeout** 设置为 **MAXDWORD**，并将 **ReadTotalTimeoutMultiplier** 和 **ReadTotalTimeoutConstant** 设置为 0，表示读取操作将立即返回存放在输入缓冲区的字符。

**WriteTotalTimeoutMultiplier**: 写入每字符间的超时。

**WriteTotalTimeoutConstant**: 一次写入串口数据的固定超时。所以在一次写入串口的操作中，其超时为 **WriteTotalTimeoutMultiplier** 乘以写入的字节数再加上 **WriteTotalTimeoutConstant**。

示例如下：

```
//Set read timeout
COMMTIMEOUTS timeouts;
GetCommTimeouts(hCom_1, &timeouts);
timeouts.ReadIntervalTimeout = 0;
timeouts.ReadTotalTimeoutMultiplier = 0;
timeouts.ReadTotalTimeoutConstant = 6000;
timeouts.WriteTotalTimeoutMultiplier = 0;
timeouts.WriteTotalTimeoutConstant = 0;
SetCommTimeouts(hCom_1, &timeouts);

//Set read/write buffer size
static const int nBufSize = 32768;
if(!SetupComm(hCom_1, nBufSize, nBufSize))
{
    printf("SetupComm() failed, comm port closed...\n");
}
```

```
    CloseHandle(hCom_1);  
  
    return -1;  
}
```

## 3.3 参数设置

Windows 提供了专门的数据结构 DCB 和专门的接口函数 `GetCommState()` / `SetCommState()`，用于设置串口参数。

DCB 是一个用于串口通信设备设置的数据结构。其中串口基本设置的变量如下：

**BaudRate:** 波特率设置，可设置的最大值为 256000；

**Parity:** 校验位设置，可设置为无校验（NOPARITY）、奇校验（ODDPARITY）、偶校验（EVENPARITY）、1 校验（MARKPARITY）、0 校验（SPACEPARITY）。

**ByteSize:** 传输单个字节的比特数。

**StopBits:** 停止位设置，可设置为 1 位停止位（ONESTOPBIT）、1.5 停止位（ONE5STOPBITS）、2 停止位（TWOSTOPBITS）。

示例如下：

```
//Set comm port configurations  
DCB dcb;  
if (!GetCommState(hCom_1, &dcb))  
{  
    printf("GetCommState() failed, comm port closed...\n");  
    CloseHandle(hCom_1);  
    return -1;  
}  
  
//Configuration settings  
int nBaud = 115200;  
dcb.DCBlength = sizeof(DCB);  
dcb.BaudRate = nBaud; //波特率  
dcb.Parity = NOPARITY; //校验方式  
dcb.ByteSize = 8; //字节位宽
```

```
    dcb.StopBits = TWOSTOPBITS; //停止位

    if (!SetCommState(hCom_1, &dcb))
    {
        printf("SetCommState() failed\n");
        CloseHandle(hCom_1);
        return -1;
    }
```

## 3.4 清除串口缓冲

Windows 提供 **PurgeComm()** 函数用于停止读写操作和清除读写缓冲区。第一次读取串口数据、写串口数据之前，或串口长时间未使用、串口出现错误等情况下，应当先清空读写缓冲区。示例如下：

```
    DWORD dwError;
    COMSTAT cs;
    if (!ClearCommError(hCom_1, &dwError, &cs))
    {
        printf("ClearCommError() failed\n");
        CloseHandle(hCom_1);
        return -1;
    }
```



## 3.5 清除串口错误

Windows 提供 `ClearCommError()` 函数来清除通信中的错误，以及获得当前通信的状态。在读写操作之前，可调用 `ClearCommError()` 来清除错误，获得缓冲区内数据大小。示例如下：

```
DWORD dwError;  
  
COMSTAT cs;  
  
if(!ClearCommError(hCom_1, &dwError, &cs))  
{  
  
    printf("ClearCommError() failed\n");  
  
    CloseHandle(hCom_1);  
  
    return -1;  
  
}
```

## 3.6 通过虚拟串口设备发送数据（同步）

可使用 Windows 提供的函数 `WriteFile()` 发送数据，执行成功返回 `TRUE`，失败则返回 `FALSE`。

同步发送数据时，应将函数参数中 `OVERLAPPED` 数据结构地址设置为 `NULL`。

示例如下：

```
BOOL bErrorFlag = FALSE;  
  
char DataBuffer[] = "This is some test data to write to the virtual  
com port.\r\n";  
  
DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);  
  
DWORD dwBytesWritten = 0;  
  
bErrorFlag =  
WriteFile (  
  
    hCom_1,          // 串口句柄  
  
    DataBuffer,     // 保存要发送数据的地址  
  
    dwBytesToWrite, // 要发送的数据字节数  
  
    &dwBytesWritten, // 保存实际发送数据字节数的变量地址  
  
    NULL           // OVERLAPPED 数据结构地址，同步发送时为 NULL  
  
);
```

## 3.7 通过虚拟串口设备接收数据（同步）

可使用 Windows 提供的函数 `ReadFile()`接收数据, 执行成功返回 `TRUE`, 失败则返回 `FALSE`。

同步接收数据时, 应将函数参数中的 `OVERLAPPED` 数据结构地址设置为 `NULL`。

示例如下:

```
char buf[101];

BOOL bErrorFlag = FALSE;

DWORD nLenOut = 0;

bErrorFlag =

ReadFile (

    hCom_1,    // 串口句柄

    buf,       // 接收到数据的保存地址

    100,       // 要接收的数据字节数

    &nLenOut,   // 保存实际接收到的数据的字节数的变量地址

    NULL      // OVERLAPPED 数据结构地址, 同步接收时为 NULL

);

if(bErrorFlag)

{

    if(nLenOut)

    { // 成功

    }

    else

    { // 超时

    }

}

else

{ // 失败

}
```

## 3.8 异步读写串口

异步读写串口的方式与同步读写相类似, 区别在于需要以重叠方式打开串口。如果重叠操作不能立即完成, 则 `WaitCommEvent()` 返回 `FALSE`, `GetLastError()` 会返回 `ERROR_IO_PENDING`, 表示操作正在后台进行, 在 `WaitCommEvent` 返回之前, 参数重叠结构中的 `hEvent` 成员会被设置为无信号状态。如果当事件发生或错误发生时, 其被设置为有信号状态, 应用程序可以调用等待函数 (`WaitForSingleObject`、`WaitForSingleObjectEx` 等) 来判断时间对象的状态, 而 `WaitCommEvent()` 的参数 `lpEvtMask` 会保存具体发生的事件。

有两种方法可以等待或判断重叠操作是否完成:

- 一种方法是使用 `WaitForSingleObject()` 来等待读写函数中 `OVERLAPPED` 类型的参数的 `hEvent` 成员: 当调用 `ReadFile`、`WriteFile` 函数时, 该成员会自动被置为无信号状态; 当重叠操作完成后, 该成员变量会自动被置为有信号状态。
- 另一种方法是调用 `GetOverlappedResult()` 获得重叠操作的状态, 来判断重叠操作是否完成。

下面是一个完整的虚拟串口异步读写案例:

```
//头文件
#include <stdio.h>
#include <windows.h>

DWORD WINAPI ThreadRxMsg(LPVOID lpParameter);
void getComName(char* ComNamePtr, unsigned int ComNum);
void printBuf(char *buf, unsigned int bufSize);

char ComName[13];
HANDLE hCom_1;

int main(int argc, char* argv[])
{
    DWORD dwError;
    unsigned int ComNum = 26;
```

```
getComName (ComName, ComNum);

// 以 OVERLAPPED 方式打开虚拟串口设备, 串口名为 COM26
hCom_1 =
CreateFile (
    ComName,
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    FILE_FLAG_OVERLAPPED,
    0
);

if(hCom_1 == INVALID_HANDLE_VALUE)
{
    int a = GetLastError();
    printf("Error : %d\n", a);
    return -1;
}

// 超时设置
COMMTIMEOUTS timeouts;
GetCommTimeouts (hCom_1, &timeouts);
timeouts.ReadIntervalTimeout = 0;
timeouts.ReadTotalTimeoutMultiplier = 0;
timeouts.ReadTotalTimeoutConstant = 6000;
timeouts.WriteTotalTimeoutMultiplier = 0;
timeouts.WriteTotalTimeoutConstant = 0;
SetCommTimeouts (hCom_1, &timeouts);
```

```
// 设置读写缓冲区大小

static const int nBufSize = 32768;
if (!SetupComm(hCom_1, nBufSize, nBufSize))
{
    printf("SetupComm() failed, comm port closed...\n");
    CloseHandle(hCom_1);
    return -1;
}

// 设置串口参数
DCB dcb;
if (!GetCommState(hCom_1, &dcb))
{
    printf("GetCommState() failed, comm port closed...\n");
    CloseHandle(hCom_1);
    return -1;
}

int nBaud = 115200;
dcb.DCBlength = sizeof(DCB);
dcb.BaudRate = nBaud;           //波特率
dcb.Parity = NOPARITY;        //校验方式: 无校验
dcb.ByteSize = 8;             //数据位: 8
dcb.StopBits = TWOSTOPBITS;   //停止位: 1
if (!SetCommState(hCom_1, &dcb))
{
    printf("SetCommState() failed\n");
    CloseHandle(hCom_1);
    return -1;
}
```

```
// 新建一个 OVERLAPPED 结构, 用于异步发送的控制
OVERLAPPED wrOverlapped;
memset(&wrOverlapped, 0, sizeof(wrOverlapped));
if (wrOverlapped.hEvent != NULL)
{
    ResetEvent(wrOverlapped.hEvent);
    wrOverlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
}

char DataBuffer[] = "This is some test data to write to the
virtual com port.\r\n";

DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);
DWORD dwBytesWritten = 0;
// 清除串口错误和串口发送缓冲区
if (ClearCommError(hCom_1, &dwError, NULL))
{
    PurgeComm(hCom_1, PURGE_TXABORT | PURGE_TXCLEAR);
}

// 使用 WriteFile 发送数据, 并循环判断是否完成发送
if
(!WriteFile(hCom_1, DataBuffer, dwBytesToWrite, &dwBytesWritten, &wrOverlapped))
{
    if (GetLastError() == ERROR_IO_PENDING)
    {
        while
(!GetOverlappedResult(hCom_1, &wrOverlapped, &dwBytesWritten, FALSE))
        {
            if (GetLastError() == ERROR_IO_INCOMPLETE)
            {
```

```
        //Not finished, continue and get the result again
        continue;
    }
    else
    {
        printf("Send Error Occured\n");
        ClearCommError(hCom_1, &dwError, NULL);
        break;
    }
}
printf("Data send finished, Bytes Written: %d\n",
dwBytesWritten);
}
}

// 清除读缓冲区
PurgeComm(hCom_1, PURGE_RXCLEAR | PURGE_RXABORT);

//clear error
COMSTAT cs;
if(!ClearCommError(hCom_1, &dwError, &cs))
{
    printf("ClearCommError() failed\n");
    CloseHandle(hCom_1);
    return -1;
}

SetCommMask(hCom_1, EV_RXCHAR); //设置事件为收到数据
//新建一个辅助线程用于接收数据
HANDLE hThread1 =
CreateThread(NULL, 0, ThreadRxMsg, NULL, 0, NULL);
```

```
while(1)
{
    // to do
}

CloseHandle(hCom_1);

return 0;
}

// 辅助线程，用于异步接收串口数据
DWORD WINAPI ThreadRxMsg(LPVOID lpParameter)
{
    while(1)
    {
        // 用于控制异步接收的 OVERLAPPED 结构
        OVERLAPPED osWait;
        memset(&osWait,0,sizeof(OVERLAPPED));
        osWait.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
        DWORD dwEvtMask;
        if (WaitCommEvent(hCom_1,&dwEvtMask,&osWait))
        {
            //若接收缓冲区非空，则对缓冲区内数据进行处理
            if(dwEvtMask & EV_RXCHAR)
            {
                DWORD dwError;
                COMSTAT cs;
                if(!ClearCommError(hCom_1,&dwError,&cs))
                {
                    printf("ClearCommError() failed\n");
                    CloseHandle(hCom_1);
                }
            }
        }
    }
}
```



```
        return -1;
    }

    char buf[101] = {0};
    DWORD nLenOut = 0;
    DWORD dwTrans = 0;
    OVERLAPPED osRead;
    memset(&osRead, 0, sizeof(OVERLAPPED));
    osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    BOOL bReadStatus =
ReadFile(hCom_1, buf, cs.cbInQue, &nLenOut, &osRead);
    if(!bReadStatus)
    {
        if(GetLastError() == ERROR_IO_PENDING)
        {
            printf("GetLastError() == ERROR_IO_PENDING\n");
            // to do
        }
    }
    else
    {
        printBuf(buf, nLenOut);
    }
}
else
{
    if(GetLastError() == ERROR_IO_PENDING)
    {
        WaitForSingleObject(osWait.hEvent, INFINITE);
        if(dwEvtMask & EV_RXCHAR)
```

```
{
    DWORD dwError;
    COMSTAT cs;
    if(!ClearCommError(hCom_1, &dwError, &cs))
    {
        printf("ClearCommError() failed\n");
        CloseHandle(hCom_1);
        return -1;
    }

    char buf[101] = {0};
    DWORD nLenOut = 0;
    DWORD dwTrans = 0;
    OVERLAPPED osRead;
    memset(&osRead, 0, sizeof(OVERLAPPED));
    osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    BOOL bReadStatus =
ReadFile(hCom_1, buf, cs.cbInQue, &nLenOut, &osRead);
    if(!bReadStatus)
    {
        if(GetLastError() == ERROR_IO_PENDING)
        {
            printf("GetLastError() == ERROR_IO_PENDING\n");
            // to do
        }
    }
    else
    {
        printBuf(buf, nLenOut);
    }
}
```

```
    }  
    }  
}  
  
return 1;  
}
```

// 根据串口编号生成串口名称，当编号大于等于 10 时，串口名称应为“\\\\.\\COMXX”的格式。

```
void getComName(char* ComNamePtr, unsigned int ComNum)  
{  
    if(ComNum > 9)  
    {  
        sprintf(ComNamePtr, "\\\\.\\COM%d", ComNum);  
    }  
    else  
    {  
        sprintf(ComNamePtr, "COM%d", ComNum);  
    }  
  
    return;  
}
```

// 打印接收到的串口数据

```
void printBuf(char *buf, unsigned int bufSize)  
{  
    for(int i = 0; i < bufSize; i++)  
    {  
        #ifdef PRINT_HEX  
            printf("%02x ", buf[i]);  
        #else  
            printf("%c ", buf[i]);  
        #endif  
    }  
}
```

```
        printf("%c", buf[i]);  
    #endif  
}  
  
return;  
}
```

Windows 虚拟串口的函数接口，详情请参阅微软官方文档 [Serial Communications Functions](#).

## 术语、缩略语

表 A-1 中列出了本手册中出现的相关术语、缩略语及相关释义。

**表 A-1 术语、缩略语**

术语、缩略语	全称	含义
USB	Universal Serial Bus	通用串行总线
UART	Universal Asynchronous Receiver/Transmitter	通用异步收发器

## 技术支持与反馈

高云半导体提供全方位技术支持，在使用过程中如有任何疑问或建议，可直接与公司联系：

网址：[www.gowinsemi.com.cn](http://www.gowinsemi.com.cn)

E-mail：[support@gowinsemi.com](mailto:support@gowinsemi.com)

Tel: +86 755 8262 0391

