در کامپیوتری که با سیستم‌عامل DOS یا ویندوز ۹۵ یا ۹۸ یا Millenium (Me) کار می‌کند، برنامه‌های تبادل اطلاعات با درگاههای کامپیوتر به درستی اجرا می‌شوند. اما در نسخه‌های بعدی ویندوز (NT، 2000، XP، Vista و ...) داستان ارتباط با درگاههای ورودی/خروجی متفاوت است؛ در این سیستم‌عامل‌ها، برنامه‌های کاربر در مد کاربردی[1] و توابع سیستمی (از جمله توابع دسترسی به درگاههای ورودی/خروجی) در مد هسته سیستم‌عامل[2] اجرا می‌شوند. این کار برای بالا بردن امنیت سیستم‌عامل و جلوگیری از حملات مخرب انجام می‌شود و آزادی عملی که کاربران سیستم‌عامل‌های قبل از ویندوز NT برای اجرای توابع سیستمی داشتند را از آنان می‌گیرد. بنابراین اگر برنامه‌ای برای تبادل مستقیم اطلاعات با درگاههای ورودی/خروجی بنویسید (حتی اگر از زبانهای تحت ویندوز مانند Visual C++ و Delphi استفاده کنید)، در ویندوزهای NT به بعد در اجرای صحیح برنامه ناکام خواهید ماند. برای تصحیح این مشکل، باید از برنامه‌هایی مانند PortTalk یا کتابخانه پویای inpout32 استفاده کنید که به صورت واسط بین مد کاربردی و مد هسته عمل می‌کنند؛ به بیان دیگر درخواستهای کاربر برای دسترسی به درگاهها را دریافت و به هسته سیستم‌عامل ارائه کرده و پاسخ را از هسته گرفته و به کاربر می‌دهند.

در ادامه دو روش برای این کار مورد بررسی قرار می‌گیرند. در روش اول، از توابع کتابخانه‌ای آماده که در فایلهای inpout32.dll و inpout32.lib وجود دارند استفاده می‌کنیم و در روش دوم بسته نرم‌افزاری PortTalk را مورد بررسی قرار می‌دهیم. این بسته دارای یک فایل اجرایی به نام AllowIO.exe است که ابزاری ساده برای در دسترس قرار دادن درگاههای ورودی/خروجی عرضه می‌کند. پس از بررسی این ابزار، با الهام از برنامه PortTalk توابعی سیستمی خواهیم نوشت که عملیات دسترسی به درگاه را انجام می‌دهند. با قرار دادن این توابع در یک فایل سرآیند و استفاده از آن در برنامه‌های خود می‌توانید به درگاههای ورودی/خروجی دسترسی داشته باشید.

برای فهم مطالبی که در ادامه می‌آید، نیازی به دانستن مفاهیم پیشرفته برنامه‌نویسی تحت ویندوز نیست و به سادگی می‌توانید از آنها حتی در برنامه‌های کنسول[3] که شباهت زیادی به برنامه‌های تحت DOS دارند، استفاده کنید.

کتابخانه inpout32.dll

فایل inpout32.dll یک کتابخانه پویاست که به کمک توابع آن می‌توان از درگاههای ورودی/خروجی کامپیوتر استفاده کرد. با مراجعه به سایت http://www.logix4u.net فایلهای

---

[1] ***Application Mode***
[2] ***Kernel Mode***
[3] ***Console***

inpout32.dll و inpout32.lib را دانلود کنید. در این کتابخانه دو تابع به نامهای Inp32 و Out32 تعریف شده که الگوی استفاده از آنها دقیقاً مانند توابع inportb و outportb زبان C است که قبلاً معرفی شدند.

از این کتابخانه‌ها می‌توان در تمام زبانهای تحت ویندوز استفاده کرد. در ادامه، مراحل به‌کارگیری این کتابخانه‌ها را در محیط Visual C++ تشریح می‌کنیم.

- ابتدا به کمک بسته Visual Studio یک برنامه دلخواه برای ارتباط با درگاه بنویسید. این برنامه می‌تواند مبتنی بر MFC یا API یا Console باشد. از توابع Inp32 و Out32 برای ارتباط با درگاهها استفاده کنید.
- دو فایل فوق را در شاخه پروژه خود کپی کنید.
- از منوی Project گزینه Setting را انتخاب کنید. به سربرگ Link بروید و نام فایل کتابخانه‌ای inpout32.lib را به قسمت object/library اضافه کنید.
- دو خط زیر را به ابتدای فایل اصلی برنامه خود (بعد از راهنماهای اولیه[1]) اضافه کنید:

```
short _stdcall Inp32(short portaddr);
void  _stdcall Out32(short portaddr, short datum);
```

برای نمونه، برنامه ارسال داده‌ها به کارت XT خروجی را (که قبلاً در زبان C تحت DOS نوشته بودیم) بازنویسی می‌کنیم. با اجرای این دستورات در مد کنسول محیط Visual Studio می‌توانید به نتایجی مشابه قبل دست یابید و نیز آن را به دلخواه خود تغییر دهید؛ مثلاً اگر به جای آدرس 031Bh، از آدرس درگاه موازی (0378h) استفاده کنید و هشت LED به پایه‌های داده این درگاه متصل کنید، چشمک زدن آنها را خواهید دید.

```
#include " stdafx.h "
#include < conio.h >
#include < windows.h >

short _stdcall Inp32(short portaddr);
void _stdcall Out32(short portaddr, short datum);

void main(void){

        short data = 0x55;

        while (!kbhit()){

                Out32(0x031B , data);
                data = ~data;
                Sleep(500);
        }
}
```

[1] ***pre processor directives***

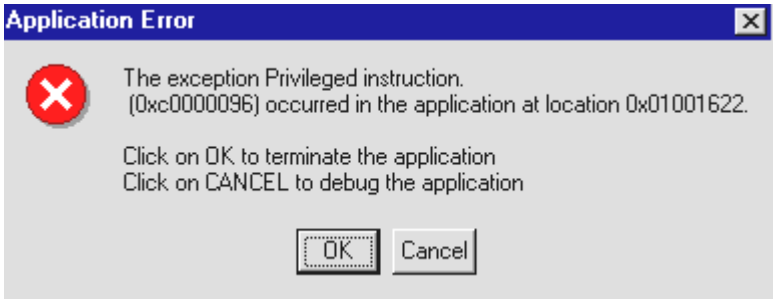
به همین روش می‌توانید از توابع Inp32 و Out32 در برنامه‌های دیگر تحت ویندوز استفاده کنید. برای تمرین، برنامه‌ای بنویسید که خروجی یک A/D که دمای اتاق را به عنوان ورودی از حسگر دما دریافت کرده را روی نمایشگر کامپیوتر نشان دهد.

نرم‌افزار PortTalk

این نرم‌افزار که از طریق سایت http://www.beyondlogic.org قابل دسترسی است، برای استفاده از درگاههای ورودی/خروجی در برنامه‌های تحت ویندوز به کار می‌رود. برای توضیح شیوه کاری این نرم‌افزار، نگاهی دقیقتر به دلایل محدودیت دسترسی به درگاههای ورودی/خروجی کامپیوتر در ویندوزهای NT به بعد می‌اندازیم. برای فهم این مطالب مد محافظت‌شده پردازنده‌های ۸۰۲۸۶ و ۸۰۳۸۶ را تا حدی بشناسید.

ویندوز NT با استفاده از قابلیتهای پردازنده در مد محافظت‌شده[1] محدودیتهایی برای برنامه‌های کاربردی در دسترسی به درگاههای ورودی/خروجی کامپیوتر به وجود می‌آورد. این دسترسی تابع دو مفهوم بیتهای IOPL[2] و نقشه بیتی اجازه دسترسی به درگاههای ورودی/خروجی (IOPM)[3] است.

بیتهای IOPL: دو بیت IOPL در ثبات EFLAGS پردازنده وجود دارند که چهار سطح دسترسی به ورودی/خروجی را ایجاد می‌کنند که سطح صفر بالاترین اولویت است. اگر برنامه‌ای که سطح دسترسی تعریف شده آن پایینتر از سطح تعریف شده در IOPL باشد برای دسترسی به درگاهها تلاش کند، وقفه شماره ۱۳[4] رخ می‌دهد و در ویندوز پیغامی به شکل زیر ظاهر می‌شود.



هر برنامه‌ای که می‌خواهد به درگاههای ورودی/خروجی دسترسی داشته باشد باید سطح دسترسی تعریف شده توسط IOPL را تا سطح اولویت برنامه پایین بیاورد.

---

[1] ***Protected Mode***
[2] ***I/O Privileged Level***
[3] ***I/O Permission bitMap***
[4] ***General Protection Error***

نقشه بیتی دسترسی به درگاه: هر فرآیند[۱] که در ویندوز اجرا می‌شود، دارای یک سگمنت حالت فرآیند (TSS[۲]) است که اطلاعات کامل مربوط به آن فرآیند را نگهداری می‌کند. از TSS برای تعویض بین فرآیندها در سیستمهای چندوظیفه[۳] استفاده می‌شود. هر TSS حاوی یک نقشه بیتی اجازه دسترسی به درگاههای ورودی/خروجی است که IOPM[۴] نامیده می‌شود. به ازای هرکدام از $۲^{۱۶}$ (۶۵۵۳۶) درگاه ورودی/خروجی کامپیوتر (که به کمک ۱۶ بیت آدرس‌دهی می‌شود)، یک بیت در این نقشه وجود دارد که اگر «یک» باشد، دستور دسترسی به درگاه با یک خطای استثنای[۵] «نقض اولویت[۶]» روبرو می‌شود و اگر «صفر» باشد دسترسی به درگاه صورت خواهد گرفت.

در ویندوز NT، تنها دو سطح دسترسی (از چهار سطح دسترسی) به کار می‌رود که سطح صفر (بالاترین اولویت) و سطح سه (پایین‌ترین اولویت) می‌باشند. برنامه‌های کاربر در سطح سه و برنامه‌های راه‌انداز وسایل[۷] و هسته سیستم‌عامل در سطح صفر (معروف به حلقه صفر[۸]) اجرا می‌شوند. به همین دلیل برنامه‌های کاربر از حق کافی برای دسترسی به درگاههای ورودی/خروجی برخوردار نیستند و این دسترسی فقط باید توسط برنامه‌های سطح صفر انجام شود. تمام برنامه‌های مد کاربر باید از طریق یک برنامه راه‌انداز وسیله که در سطح صفر اجرا شده به درگاههای ورودی/خروجی دست یابند.

هنگامی که یک دستور ورودی/خروجی اجرا می‌شود، پردازنده ابتدا بررسی می‌کند آیا فرآیند درخواست‌کننده، مجوز لازم برای دسترسی به درگاههای ورودی/خروجی را دارد یا خیر؟ در صورت مثبت بودن پاسخ، دستور اجرا خواهد شد. در غیر این صورت، نقشه بیتی اجازه دسترسی به درگاهها در TSS آن برنامه بررسی و به کمک آن مشخص می‌شود آیا اجازه دسترسی به درگاه ورودی/خروجی مورد نظر داده شده است یا خیر؟

دو راه برای حل مشکل دسترسی به درگاهها در ویندوزهای NT به بعد وجود دارد. روش اول، نوشتن یک برنامه راه‌انداز وسیله است که در سطح صفر اجرا شده و روند دسترسی به درگاههای ورودی/خروجی را به نفع شما تغییر دهد. تبادل اطلاعات بین برنامه کاربر و برنامه راه‌انداز به کمک توابع کنترلی ورودی/خروجی (IOCTL) انجام شده و برنامه راه‌انداز مسئول اجرای دستورات ورودی/خروجی شما خواهد بود.

---

[1] ***Task - Process***
[2] ***Task State Segment***
[3] ***Multi-Task***
[4] ***Input/Output Permission Map***
[5] ***Privilege Exception***
[6] ***Perivileged Instruction Violation***
[7] ***Device Driver***
[8] ***Ring 0***

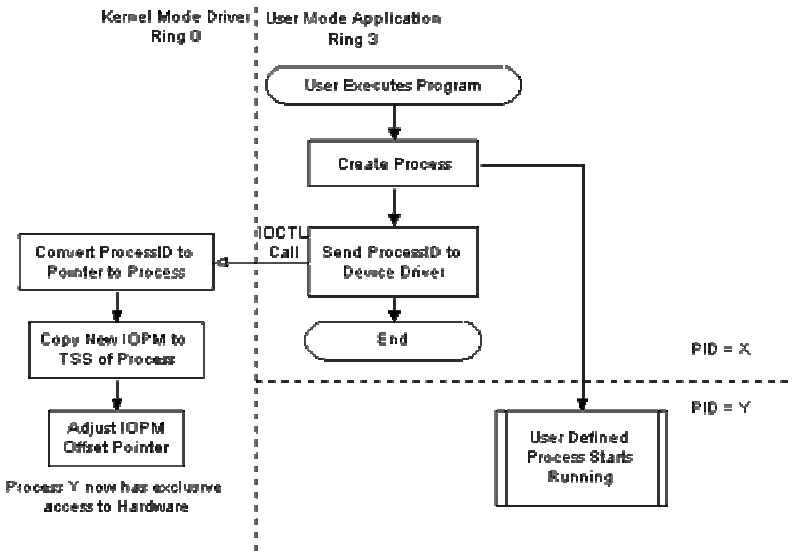
راه دیگر، اصلاح نقشه بیتی دسترسی به درگاهها در TSS فرآیند است تا مجوزهای لازم برای دسترسی به درگاههای مشخص به آن برنامه داده شود. این راه چندان مناسب نیست، اما به شما این امکان را می‌دهد که برنامه‌هایی که از قبل (مثلاً تحت DOS) نوشته‌اید را بدون تغییر و تنها به کمک یک برنامه که مجوزهای دسترسی را تغییر می‌دهد، در ویندوزهای NT به بعد اجرا کنید (برنامه AllowIO.exe که در ادامه بررسی می‌شود همین کار را انجام می‌دهد).

استفاده از برنامه‌های راه‌انداز مانند PortTalk می‌تواند به ناکارآمدی از نظر زمانی بیانجامد. هربار یک تابع IOCTL برای خواندن از درگاه ورودی یا نوشتن در درگاه خروجی اجرا می‌شود، پردازنده باید از سطح سه به سطح صفر برود، تابع را اجرا کند و مجدداً به سطح سه بازگردد. به این جهت باید در نوشتن برنامه‌هایی که نیاز به تبادل سریع اطلاعات با درگاهها ورودی/خروجی دارند، ملاحظات زمانی را مد نظر قرار دهید.

برنامه AllowIO.exe

به کمک این برنامه می‌توانید برنامه‌هایی که در DOS یا ویندوزهای قبل از NT (برنامه‌های ۱۶ بیتی) به درگاهها دسترسی داشته‌اند را بدون مشکل در ویندوزهای NT به بعد اجرا کنید. برنامه‌های ۱۶ بیتی ویندوز یا برنامه‌های DOS، در ویندوزهای NT به بعد به کمک ماشین مجازی[1] (که در ویندوز عمل می‌کند) اجرا می‌شوند. ماشین مجازی، عملیات ورودی/خروجی را متوقف کرده و آنها را به یک مرجع رسیدگی ورودی/خروجی ارجاع می‌دهد. اگر ماشین مجازی DOS (VDM[2]) دارای مجوز لازم برای دسترسی به درگاههای ورودی/خروجی باشد، عملیات ورودی/خروجی را قطع نکرده و از نظر ملاحظات زمانی بهتر عمل می‌کند.

برای تغییر IOPM یک فرآیند ، ابتدا باید شماره شناسایی (ID) فرآیند را بدانید. اگر فرآیند را خودتان ایجاد کنید، به سادگی می‌توانید شماره شناسایی آن را به برنامه راه‌انداز رد کنید. برنامه کاربردی کوچکی می‌تواند این کار را انجام دهد که نام برنامه شما را بگیرد، با اجرای آن یک فرآیند ایجاد کند و شماره



[1] *Virtual Machines*
[2] *Virtual DOS Machine*

شناسایی آن را به شما ارائه کند[1]. شکل مقابل مراحل ایجاد مجوزهای دسترسی به درگاهها را نشان می‌دهد.

هنگامی که اجرای یک برنامه ۳۲ بیتی ویندوز (برنامه‌ای که تحت ویندوزهای NT به بعد نوشته شده) به کمک تابع سیستمی CreateProcess() آغاز می‌شود، شماره شناسایی فرآیند ایجاد شده و به کمک یک تابع IOCTL به برنامه راه‌انداز ارسال می‌شود.
برنامه‌های تحت DOS شماره فرآیند ندارند و تحت ماشین مجازی DOS ویندوز NT (NTVDM) اجرا می‌شوند که یک زیرسیستم حفاظت‌شده برای شبیه‌سازی DOS است.
هنگامی که یک برنامه تحت DOS در ویندوز NT فراخوانی می‌شود، شماره شناسایی NTVDM را به خود می‌گیرد.
هنگامی که برنامه راه‌انداز شماره شناسایی فرآیند را بداند، اشاره‌گر به فرآیند را یافته و مجوزهای دسترسی به ورودی/خروجی (IOPM) را در آن اصلاح می‌کند.

استفاده از برنامه AllowIO.exe بسیار ساده است؛ فرض کنید برنامه‌ای به نام TestPort.exe تحت DOS نوشته‌اید که قبلاً با درگاه موازی به شماره 0378h کار می‌کرده و اکنون در ویندوز NT به مشکل برخورده است. مراحل زیر را دنبال کنید:

- بسته نرم‌افزاری PortTalk را از سایت http://beyondlogic.org دانلود و بازگشایی کنید.
- فایل PortTalk.sys را در شاخه System32\Driver ویندوز خود کپی کنید. این فایل، برنامه راه‌اندازی است که باید در سطح اولویت صفر اجرا شود.
- برنامه porttalk.reg را اجرا کنید تا اصلاحات لازم در رجیستری ویندوز انجام شود.
- سیستم خود را خاموش و مجدداً روشن کنید.
- فایل اجرایی AllowIO.exe را در مسیری که فایل اجرایی برنامه‌تان قرار دارد (مثلاً C:\MyProject)، کپی کنید.
- وارد محیط DOS ویندوز شوید و به مسیر فوق بروید و برنامه خود را به صورت زیر اجرا کنید:

```
C:\MyProject\> AllowIO.exe  TestPort.exe  0x0378
```

با اجرای این دستور، برنامه شما اجرا شده و بدون ایجاد هیچ مشکلی با درگاه موازی ارتباط برقرار می‌کند. عددی که مقابل دستور ذکر می‌شود محدوده درگاههای مورد نیاز را مشخص می‌کند؛ در حالت کلی اگر عدد N را مقابل دستور بیاورید ، برنامه AllowIO.exe دسترسی

[1] راه دیگر استفاده از تکنیکهای Callback است که در اینجا به آن نمی‌پردازیم.

به درگاهها در محدوده آدرس N تا N+8 را فراهم می‌آورد. اگر به جای آدرس درگاه مورد نظر، از a/ استفاده کنید، برنامه شما می‌تواند به همه درگاههای ورودی/خروجی دسترسی داشته باشد. تنها باید مراقب باشید که از این برنامه تحت نظارت کامل خودتان استفاده شود.

در شاخه checked از پوشه PortTalk، نسخه دیگری از برنامه راه‌انداز porttalk.sys وجود دارد که اطلاعات debug در آن نهفته است و به همین لحاظ در اجرا سرعت پایین‌تری دارد. اگر مایل به استفاده از اطلاعات debug (مانند سرریز بافر و ...) هستید می‌توانید به جای برنامه راه‌اندازی که در شاخه اصلی پوشه PortTalk قرار دارد، از این برنامه استفاده کنید. با این کار می‌توانید پیغامهای debug را به کمک نرم‌افزارهای مناسب[1] مشاهده کنید.

استفاده از PortTalk در برنامه‌های تحت ویندوز

با اینکه برنامه AllowIO.exe ابزار لازم برای اجرای برنامه‌های دسترسی به درگاههای ورودی/خروجی را فراهم می‌آورد، اما شیوه فراخوانی آن برای استفاده در برنامه‌هایی که کاربر به کمک محیطهای تحت ویندوز می‌نویسد، مناسب نیست.

در ادامه، توابع سیستمی نرم‌افزار PortTalk را معرفی و به اختصار توضیحاتی راجع به آنها ارائه می‌کنیم. با نوشتن این توابع در یک فایل سرآیند[2] (مثلاً PortTalk.h) و شامل کردن آن در برنامه‌های خود، می‌توانید به راحتی از این توابع برای دسترسی به درگاههای ورودی/خروجی استفاده کنید. این توابع از سرآیندهای winioctl.h و windows.h استفاده می‌کنند. شامل کردن آنها را در سرآیند PortTalk.h فراموش نکنید.

تعاریف زیر را در سرآیند PortTalk.h انجام دهید:

```
#define   PORTTALK_TYPE   40000 /* 32768-65535 are reserved for customers */

// The IOCTL function codes from 0x800 to 0xFFF are for customer use.

#define   IOCTL_IOPM_RESTRICT_ALL_ACCESS \
    CTL_CODE(PORTTALK_TYPE,0x900, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_IOPM_ALLOW_EXCUSIVE_ACCESS \
    CTL_CODE(PORTTALK_TYPE, 0x901, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_SET_IOPM \
    CTL_CODE(PORTTALK_TYPE, 0x902, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_ENABLE_IOPM_ON_PROCESSID \
    CTL_CODE(PORTTALK_TYPE, 0x903, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_READ_PORT_UCHAR \
    CTL_CODE(PORTTALK_TYPE, 0x904, METHOD_BUFFERED, FILE_ANY_ACCESS)

#define IOCTL_WRITE_PORT_UCHAR \
    CTL_CODE(PORTTALK_TYPE, 0x905, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

[1] مانند نرم‌افزار System Internals DebugView که از سایت http://www.sysinternals.com قابل دانلود است.

[2] ***Header***

```
HANDLE   PortTalk_Handle;        /* Handle for PortTalk Driver */
```

از دستگیره[1] PortTalk_Handle برای دسترسی به درگاه استفاده می‌شود.

در ابتدای کار باید نرم‌افزار راه‌انداز PortTalk نصب شده باشد. به شیوه‌ای که قبلاً برای استفاده از ALLOWIO شرح داده شد عمل کنید و توابع زیر را در ابتدای برنامه خود برای نصب و راه‌اندازی PortTalk فراخوانی کنید.

```
void InstallPortTalkDriver(void)
{
    SC_HANDLE  SchSCManager;
    SC_HANDLE  schService;
    DWORD      err;
    CHAR       DriverFileName[80];

/*
Get Current Directory. Assumes PortTalk.SYS driver is in this directory. Doesn't
detect if file exists, nor if file is on removable media - if this is the case then
when windows next boots, the driver will fail to load and a error entry is made in
the event viewer to reflect this.
Get System Directory. This should be something like c:\windows\system32 or
c:\winnt\system32 with a Maximum Character lenght of 20. As we have a buffer of 80
bytes and a string of 24 bytes to append, we can go for a max of 55 bytes.
*/

    if (!GetSystemDirectory(DriverFileName, 55))
        {
             printf("PortTalk: Failed to get System Directory.");
             printf("Is System Directory Path > 55 Characters?\n");
             printf("PortTalk: Please manually copy driver to your");
             printf("system32/driver directory.\n");
        }

    /* Append our Driver Name */
    lstrcat(DriverFileName,"\\Drivers\\PortTalk.sys");
    printf("PortTalk: Copying driver to %s\n",DriverFileName);

    /* Copy Driver to System32/drivers directory.
    This fails if the file doesn't exist. */

    if (!CopyFile("PortTalk.sys", DriverFileName, FALSE))
        {
             printf("PortTalk: Failed to copy driver to");
             printf("%s\n",DriverFileName);
             printf("PortTalk: Please manually copy driver to your");
             printf("system32/driver directory.\n");
        }

    /* Open Handle to Service Control Manager */
    SchSCManager = OpenSCManager (NULL,/*machine (NULL == local)*/
                                  NULL,/*database (NULL == default)*/
                                  SC_MANAGER_ALL_ACCESS);/*access required*/

/*
Create Service/Driver - This adds the appropriate registry keys in
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services - It doesn't care if the
driver exists, or if the path is correct.
*/

    schService = CreateService (SchSCManager,/* SCManager database */
                                "PortTalk",/* name of service */
                                "PortTalk",/* name to display */
                                SERVICE_ALL_ACCESS,/* desired access */
                                SERVICE_KERNEL_DRIVER,/* service type */
```

[1] *Handle*

```
                                SERVICE_DEMAND_START,/* start type */
                                SERVICE_ERROR_NORMAL,/* error control type */
                                "System32\\Drivers\\PortTalk.sys",
                                  /* service's binary */
                                NULL,/* no load ordering group */
                                NULL,/* no tag identifier */
                                NULL,/* no dependencies */
                                 NULL,/* LocalSystem account */
                                NULL/* no password */
                                );

    if (schService == NULL) {
         err = GetLastError();
         if (err == ERROR_SERVICE_EXISTS){
                    printf("PortTalk: Driver already exists.");
                    printf("No action taken.\n");
               }
         else {
                    printf("PortTalk: Unknown error while creating");
                    printf("Service.\n");
               }
    }
    else
              printf("PortTalk: Driver successfully installed.\n");

    /* Close Handle to Service Control Manager */
    CloseServiceHandle (schService);
}

//*************************************************************

unsigned char StartPortTalkDriver(void)
{
    SC_HANDLE  SchSCManager;
    SC_HANDLE  schService;
    BOOL       ret;
    DWORD      err;

    /* Open Handle to Service Control Manager */
    SchSCManager = OpenSCManager (NULL,/* machine (NULL == local)*/
                                  NULL,/*database (NULL == default)*/
                                  SC_MANAGER_ALL_ACCESS);/* access required */

    if (SchSCManager == NULL)
      if (GetLastError() == ERROR_ACCESS_DENIED) {
         /* We do not have enough rights to open the SCM, therefore we must */
         /* be a poor user with only user rights. */
                printf("PortTalk: You do not have rights to access");
               printf("the Service Control Manager and\n");
                printf("PortTalk: the PortTalk driver is not installed");
               printf("or started. Please ask \n");
                printf("PortTalk: your administrator to install the driver");
               printf("on your behalf.\n");
                return(0);
         }

    do {
         /* Open a Handle to the PortTalk Service Database */
         schService = OpenService(SchSCManager,
                                  /* handle to service control manager database */
                                   "PortTalk",
                                  /*pointer to name of service to start */
                                   SERVICE_ALL_ACCESS);
                                  /* type of access to service */

         if (schService == NULL)
            switch (GetLastError()){
                case ERROR_ACCESS_DENIED:
                        printf("PortTalk: You do not have rights ");
                        printf("to the PortTalk service database\n");
                        return(0);
```

```
                case ERROR_INVALID_NAME:
                        printf("PortTalk: The specified service name ");
                        printf("is invalid.\n");
                        return(0);
                case ERROR_SERVICE_DOES_NOT_EXIST:
                        printf("PortTalk: The PortTalk driver does");
                        printf("not exist. Installing driver.\n");
                        printf("PortTalk: This can take up to 30 seconds");
                        printf("on some machines . .\n");

                        InstallPortTalkDriver();
                        break;
            }
          } while (schService == NULL);

/*
Start the PortTalk Driver. Errors will occur here if PortTalk.SYS file doesn't
exist
*/

    ret = StartService (schService,    /* service identifier */
                        0,             /* number of arguments */
                        NULL);         /* pointer to arguments */

    if (ret)
       printf("PortTalk: The PortTalk driver has been successfully started.\n");
    else {
        err = GetLastError();
        if (err == ERROR_SERVICE_ALREADY_RUNNING)
          printf("PortTalk: The PortTalk driver is already running.\n");
        else {
          printf("PortTalk: Unknown error while starting PortTalk");
          printf("driver service.\n");
          printf("PortTalk: Does PortTalk.SYS exist in your");
          printf("\\System32\\Drivers Directory?\n");
          return(0);
        }
    }

    /* Close handle to Service Control Manager */
    CloseServiceHandle (schService);
    return(TRUE);
}
```

تابع OpenPortTalk برای باز کردن درگاه ورودی/خروجی

```
unsigned char OpenPortTalk(void)
{
/* Open PortTalk Driver. If we cannot open it, try installing and starting it */
    PortTalk_Handle = CreateFile("\\\\.\\PortTalk",
                                 GENERIC_READ,
                                 0,
                                 NULL,
                                 OPEN_EXISTING,
                                 FILE_ATTRIBUTE_NORMAL,
                                 NULL);

    if(PortTalk_Handle == INVALID_HANDLE_VALUE) {
            /* Start or Install PortTalk Driver */
            StartPortTalkDriver();
            /* Then try to open once more, before failing */
            PortTalk_Handle = CreateFile("\\\\.\\PortTalk",
                                         GENERIC_READ,
                                         0,
                                         NULL,
                                         OPEN_EXISTING,
                                         FILE_ATTRIBUTE_NORMAL,
                                         NULL);

            if(PortTalk_Handle == INVALID_HANDLE_VALUE) {
```

```
                printf("PortTalk: Couldn't access PortTalk Driver,");
                printf("Please ensure driver is loaded.\n\n");
                return -1;
            }
             else
                return 1;
    }
else
       return 1;
    }
}
```

از تابع سیستمی CreateFile برای باز کردن ارتباط با درگاه استفاده شده است. چنانچه این ارتباط به درستی برقرار نشود، به کمک تابع StartPortTalkDriver ابتدا راه‌انداز نصب شده و مجدداً تلاش برای ارتباط با درگاه آغاز می‌شود.

تابع ClosePortTalk برای بستن درگاه ورودی/خروجی

برای بستن ارتباط از تابع زیر استفاده می‌کنیم:

```
void ClosePortTalk(void)
{
    CloseHandle(PortTalk_Handle);
}
```

توجه کنید که دستگیره PortTalk_Handle را در فایل سرآیند PortTalk.h به صورت عمومی تعریف کرده‌ایم و به همین لحاظ نیازی نیست به صورت پارامتر یا مقدار خروجی با توابع ارتباط برقرار کنند.

تابع outportb برای نوشتن در درگاه خروجی

```
void outportb(unsigned short PortAddress, unsigned char byte)
{
    unsigned int error;
    DWORD BytesReturned;
    unsigned char Buffer[3];
    unsigned short * pBuffer;
    pBuffer = (unsigned short *)&Buffer[0];
    *pBuffer = PortAddress;
    Buffer[2] = byte;

    error = DeviceIoControl(PortTalk_Handle,
                            IOCTL_WRITE_PORT_UCHAR,
                            &Buffer,
                            3,
                            NULL,
                            0,
                            &BytesReturned,
                            NULL);

    if (!error){
              printf("Error occured during outportb while talking to PortTalk");
              printf("driver %d\n",GetLastError());
    }
}
```

تابع outportb برای خواندن از درگاه ورودی

```
unsigned char inportb(unsigned short PortAddress)
{
    unsigned int error;
    DWORD BytesReturned;
```

```
    unsigned char Buffer[3];
    unsigned short * pBuffer;
    pBuffer = (unsigned short *)&Buffer;
    *pBuffer = PortAddress;

    error = DeviceIoControl(PortTalk_Handle,
                            IOCTL_READ_PORT_UCHAR,
                            &Buffer,
                            2,
                            &Buffer,
                            1,
                            &BytesReturned,
                            NULL);

    if (!error){
       printf("Error occured during inportb while talking to PortTalk");
       printf("driver %d\n",GetLastError());
}
    return(Buffer[0]);
}
```

همانطور که می‌بینید، توابع فوق با فرض استفاده از برنامه‌های مد کنسول نوشته شده‌اند. اگر مایل به نوشتن برنامه‌های تحت ویندوز هستید می‌توانید دستورات مد کنسول (مثلاً printf) را با توابع ویندوز (مثلاً MessageBox) عوض کنید.

برنامه ارسال اطلاعات به کارت XT خروجی را بازنویسی می‌کنیم. برنامه مقابل را ببینید.

```
#include "stdafx.h"
#include "PortTalk.h"
#include <conio.h>
#include <windows.h>

int main(int argc, char* argv[])
{
    unsigned char data = 0x55;

    OpenPortTalk();

    while(!kbhit()){
      outportb(0x031B, data);
      data = ~data;
      Sleep(500);
     }

    ClosePortTalk();

      return 0;
}
```

ارتباط با درگاه سریال

ارتباط با درگاه سریال کامپیوتر دارای تشریفات مفصلتری نسبت به درگاههای دیگر است. در این بخش، به نحوه استفاده از توابع ویندوز برای ارتباط با درگاه سریال می‌پردازیم. با تغییرات کمی می‌توانید همین مفاهیم را در مورد ارتباط با درگاههای ورودی/خروجی دیگر به کار ببرید.

همانطور که قبلاً در مبحث ارتباط سریال بیان شد، یک بسته اطلاعاتی در ارتباط سریال غیرهمگام شامل بیت آغاز، بیتهای اطلاعاتی، بیت توازن و بیت توقف است. برای بنا نهادن یک

ارتباط سریال صحیح، باید ساختار بسته اطلاعاتی و مشخصات دیگر ارتباط (مانند سرعت و نحوه کنترل جریان داده) بین فرستنده و گیرنده یکسان باشد.

از دیدگاه نرم‌افزار، در یک ارتباط سریال مراحل زیر وجود دارند:

- گشودن درگاه سریال
- تنظیم ویژگیهای ارتباط سریال
- تنظیم سرریزهای زمانی[1]
- خواندن اطلاعات از درگاه سریال یا نوشتن در آن
- بستن درگاه سریال

برای هرکدام از اعمالی که در بالا ذکر شد، تابعی می‌نویسیم که عملیات مورد نظر را انجام دهد. در توابع ویندوز با درگاههای ورودی/خروجی به صورت فایل برخورد می‌شود. برای باز کردن درگاه از تابع CreateFile، برای خواندن و نوشتن به ترتیب از توابع ReadFile و WriteFile و برای بستن آن از تابع CloseFile استفاده می‌شود.

توابعی که در ادامه بررسی می‌شود را در یک فایل سرآیند به نام COMPort.h بنویسید و در برنامه‌های ارتباطی سریال آن را شامل کنید. فایل windows.h را در این فایل شامل کنید.

تعریف زیر را به صورت عمومی در این فایل انجام دهید:

```
HANDLE hComm;
```

متغیر hComm دستگیره‌ای برای دسترسی به درگاه مورد نظر است که در توابع بعدی مورد استفاده قرار می‌گیرد. به عبارتی می‌توان این دستگیره را نام منطقی فایل در برنامه دانست.

تابع گشودن درگاه سریال

```
BOOL OpenPort(char *Port)
{
        char   portname[20];
        strcpy(portname," //./");
        strcat(portname,Port);

        hComm = CreateFile(portname,
                           GENERIC_READ | GENERIC_WRITE,
                           0,
                           0,
                           OPEN_EXISTING,
                           0,
                           0);
        if (hComm == INVALID_HANDLE_VALUE){
                printf("\nCreateFile Error Code = %d ",GetLastError());
                return FALSE;
        }
        else
                return TRUE;
}
```

[1] *Time-outs*

اگر درگاه به درستی باز شود، خروجی این تابع TRUE و در غیر این صورت FALSE است. پارامتر ورودی این تابع، نام درگاه سریال مورد نظر است؛ مثلاً فراخوانی تابع به صورت OpenPort("COM1");، درگاه COM1 را گشوده و دستگیره آن را در متغیر hComm قرار می‌دهد. از تابع CreateFile برای باز کردن درگاه استفاده شده است. از پارامتر دوم این تابع مشخص است که درگاه هم برای خواندن و هم برای نوشتن گشوده شده است.

برای باز کردن درگاه می‌توان به دو روش «با همپوشانی» (OVERLAPPED) و «بدون همپوشانی» (NON-OVERLAPPED) عمل کرد. در این نوشته (با مقداردهی پارامترهای CreateFile) از روش بدون همپوشانی استفاده شده است. برای اطلاعات بیشتر راجع به این موضوع به مستندات MSDN مراجعه کنید.

اگر در انجام تابع CreateFile خطایی رخ دهد، این تابع به جای برگرداندن یک دستگیره معتبر، وقوع خطایی را گزارش می‌کند. تابع GetLastError، کد آخرین خطایی که رخ داده است را ارائه می‌کند. به کمک مستندات MSDN می‌توانید جزییات این خطا را ببینید. در ادامه از توابعی متعلق به ویندوز استفاده می‌کنیم که هرکدام یک خروجی منطقی دارند که در صورت موفقیت TRUE و در غیر این صورت FALSE می‌باشد. اگر خروجی توابع فوق (که در متغیر RetValue قرار می‌گیرند) FALSE باشد به کمک تابع GetLastError کد خطای رخ داده شده نمایش داده می‌شود.

بعضی از خطاهای مهم عبارتند از:

- **خطای شماره ۲ (ERROR_FILE_NOT_FOUND):** هنگامی رخ می‌دهد که برای باز کردن درگاهی تلاش شود که در سیستم موجود نیست. مثلاً اگر درگاه com3 در سیستم موجود نباشد و با تابع CreateFile سعی کنید آن را باز کنید، این خطا گزارش می‌شود.
- **خطای شماره ۵ (ERROR_ACCESS_DENIED):** اگر برای باز کردن درگاهی تلاش کنید که قبلاً توسط برنامه دیگری باز شده این خطا رخ می‌دهد.
- **خطای شماره ۶ (ERROR_INVALID_HANDLE):** اگر در باز کردن درگاه موفق نباشید و دستگیره برگردانده شده توسط تابع CreateFile نامعتبر باشد، تلاش برای استفاده از این دستگیره در توابع دیگر باعث وقوع این خطا می‌شود.
- **خطای شماره ۹۹۵ (ERROR_OPERATION_ABORTED):** نشانه وقوع خطا در عملیات خواندن یا نوشتن است.

تابع پیکربندی درگاه سریال

```
BOOL ConfigurePort(DWORD BaudRate, BYTE ByteSize,
```

```
                    DWORD fParity, BYTE Parity, BYTE StopBits)
{
    BOOL   RetValue;
    DCB    m_dcb;

    RetValue = GetCommState(hComm, &m_dcb);
    if (RetValue == FALSE)
    {
         printf("\nGetCommState Error Code = %d ",GetLastError());
         CloseHandle(hComm);
        return FALSE;
    }
    m_dcb.BaudRate =BaudRate;
    m_dcb.ByteSize = ByteSize;
    m_dcb.Parity =Parity ;
    m_dcb.StopBits =StopBits;
    m_dcb.fBinary=TRUE;
    m_dcb.fDsrSensitivity=FALSE;
    m_dcb.fParity=fParity;
    m_dcb.fOutX=FALSE;
    m_dcb.fInX= FALSE;
    m_dcb.fNull= FALSE;
    m_dcb.fAbortOnError=TRUE;
    m_dcb.fOutxCtsFlow=FALSE;
    m_dcb.fOutxDsrFlow= FALSE;
    m_dcb.fDtrControl=DTR_CONTROL_DISABLE;
    m_dcb.fDsrSensitivity= FALSE;
    m_dcb.fRtsControl=RTS_CONTROL_DISABLE;
    m_dcb.fOutxCtsFlow= FALSE;
    m_dcb.fOutxCtsFlow= FALSE;

    RetValue = SetCommState(hComm, &m_dcb);
    if(RetValue == FALSE)
    {
         printf("\nSetCommState Error Code = %d",GetLastError());
         CloseHandle(hComm);
         return FALSE;
    }
    return TRUE;
}
```

در ویندوز ساختاری به نام DCB تعریف شده که برای پیکربندی ارتباط سریال از آن استفاده می‌شود. متغیری به نام m_dcb از نوع ساختار DCB تعریف می‌کنیم. تابع GetCommState از توابع ویندوز است که پیکربندی فعلی درگاهی که دستگیره آن ذکر شده را در متغیر m_dcb ذخیره می‌کند. این کار برای حفظ مشخصات پیش‌فرض درگاهها که مایل به تغییر آنها نیستیم مفید است. اکنون برای پیکربندی درگاه مورد نظر باید فیلدهای متغیر ساختاری m_dcb را مقداردهی کنیم. شرح این فیلدها و مقادیری که می‌توانند به خود بگیرند و معانی آنها به تفصیل در مستندات MSDN موجود است. در اینجا تنها به چهار پارامتر ورودی تابع ConfigurePort که چهار فیلد مهم متغیر m_dcb را مقداردهی می‌کنند می‌پردازیم:

- **پارامتر BaudRate**: به کمک این پارامتر می‌توان سرعت ارتباط سریال را تنظیم کرد. این سرعت باید مطابق با سرعت ارتباطی دستگاهی باشد که به درگاه سریال متصل است. مثلاً برای سرعت ۹۶۰۰ بیت در ثانیه باید این پارمتر را برابر 9600 یا CBR_9600 مقداردهی کنید. مقادیر معتبر برای درگاه سریال کامپیوتر عبارتند از:

```
CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600,
CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200,
CBR_128000, CBR_256000
```

- **پارامتر ByteSize**: این پارامتر، تعداد بیتهای داده در بسته اطلاعاتی را نشان می‌دهد که می‌تواند از ۴ تا ۸ باشد.
- **پارامتر fParity**: اگر مقدار این پارامتر TRUE باشد، بررسی توازن انجام خواهد شد.
- **پارامتر Parity**: این پارامتر نوع توازن مورد استفاده را مشخص می‌کند. مقادیر قابل قبول عبارتند از EVENPARITY (توازن زوج)، ODDPARITY (توازن فرد)، NOPARITY (بدون توازن)، MARKPARITY (توازن همیشه «یک»)، SPACEPARITY (توازن همیشه «صفر»)
- **پارامتر StopBits**: برای مشخص کردن تعداد بیتهای توقف در بسته اطلاعاتی از این پارامتر استفاده می‌کنیم. مقادیر معتبر عبارتند از:

```
ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS
```

به نکته مهمی توجه کنید. تابع ConfigurePortاز مکانیسمی برای کنترل جریان داده استفاده نمی‌کند. به بیان دیگر، نه RTS/CTS و نه XON/XOFF را کنترل نمی‌کند. اگر مایل به کنترل جریان داده هستید، با انجام اصلاحات روی سخت‌افزار و نیز فیلدهای دیگر متغیر m_dcb می‌توانید این کار را انجام دهید.

پس از تنظیم فیلدهای متغیر m_dcb، به کمک تابع SetCommState این متغیر را به درگاهی که دستگیره آن در تابع ذکر شده نسبت می‌دهیم.

مثالی از فراخوانی تابع ConfigurePort به صورت زیر است:

```
ConfigurePort(CBR_9600, 8, TRUE, EVENPARITY , ONESTOPBIT )
```

پیکربندی درگاه سریال با سرعت ۹۶۰۰ بیت در ثانیه، ۸ بیت داده، توازن زوج و یک بیت توقف انجام می‌شود. دستگاه مخاطب درگاه سریال نیز باید از همین پیکربندی استفاده کند. اگر خروجی تابع TRUE باشد، پیکربندی به درستی انجام شده است.

تنظیم ملاحظات زمانی

تنظیم مسایل زمانی ارتباط سریال در برنامه‌نویسی ویندوز به کمک توابع GetCommTimeoutsو SetCommTimeouts و ساختاری به نام COMMTIMEOUTS انجام می‌شود.

```
BOOL    SetCommunicationTimeouts(DWORD ReadIntervalTimeout,
                                 DWORD ReadTotalTimeoutMultiplier,
                                 DWORD ReadTotalTimeoutConstant,
                                 DWORD WriteTotalTimeoutMultiplier,
                                 DWORD WriteTotalTimeoutConstant)
{
```

```
    BOOL          RetValue;
    COMMTIMEOUTS  m_CommTimeouts;

    RetValue = GetCommTimeouts (hComm, &m_CommTimeouts);
    if(RetValue == FALSE){
         printf("\nGetCommTimeouts Error Code = %d",GetLastError());
         CloseHandle(hComm);
          return FALSE;
    }

    m_CommTimeouts.ReadIntervalTimeout =ReadIntervalTimeout;
    m_CommTimeouts.ReadTotalTimeoutConstant =ReadTotalTimeoutConstant;
    m_CommTimeouts.ReadTotalTimeoutMultiplier =ReadTotalTimeoutMultiplier;
    m_CommTimeouts.WriteTotalTimeoutConstant = WriteTotalTimeoutConstant;
    m_CommTimeouts.WriteTotalTimeoutMultiplier =WriteTotalTimeoutMultiplier;

    RetValue = SetCommTimeouts (hComm, &m_CommTimeouts);

    if(RetValue == FALSE)
    {
         printf("\nSetCommTimeouts Error Code = %d",GetLastError());
        CloseHandle(hComm);
        return FALSE;
    }
    return TRUE;
}
```

متغیری از نوع ساختار COMMTIMEOUTS به نام m_CommTimeouts تعریف شده که فیلدهای آن برای تنظیم مسایل زمانی ارتباط سریال به کار می‌روند. به کمک تابع GetCommTimeouts، مشخصات فعلی ارتباطی که دستگیره آن مشخص شده در متغیر m_CommTimeouts ذخیره می‌شود و پس از تغییرات لازم به کمک تابع SetCommTimeouts ویژگیهای تنظیم شده را به درگاهی که دستگیره آن را مشخص کرده‌ایم، نسبت می‌دهیم.

پارامترهای ورودی تابع SetCommunicationTimeouts را در زیر بررسی می‌کنیم:

- **پارامتر ReadIntervalTimeout**: ماکزیمم زمان مجاز (برحسب میلی‌ثانیه) بین دریافت دو کاراکتر متوالی را مشخص می‌کند. اگر زمان سپری شده از این مقدار بیشتر شود، تابع ReadFile که برای خواندن درگاه به کار می‌رود، عملیات خواندن را پایان‌یافته فرض کرده و به تابع فراخواننده بازمی‌گردد. اگر مقدار این پارامتر را صفر تعیین کنید، به معنای استفاده نکردن از این مشخصه ارتباط سریال است.
- **پارامتر ReadTotalTimeoutConstant**: یک مقدار ثابت بر حسب میلی ثانیه است که برای محاسبه مجموع مدت زمان بین اعمال خواندن به کار می‌رود. برای هر عمل خواندن، مقدار این ثابت به حاصلضرب تعداد بایتهای درخواستی در پارامتر ReadTotalTimeoutMultiplier اضافه می‌شود و مجموع زمانی که اصولاً عمل خواندن باید به طول بیانجامد را معلوم می‌کند. اگر زمان سپری شده از این مقدار بیشتر شود، تابع ReadFile که برای خواندن درگاه به کار می‌رود، عملیات

خواندن را پایان‌یافته فرض کرده و به تابع فراخواننده بازمی‌گردد. اگر دو پارامتر ذکر شده را صفر کنید، از استفاده از مجموع زمان بین اعمال خواندن صرف‌نظر کرده‌اید.

- **پارامتر ReadTotalTimeoutMultiplier**: ماکزیمم زمان انتقال هر بایت بر حسب میلی‌ثانیه به کمک این پارامتر مشخص می‌شود. تعداد بایتهای درخواستی در عمل خواندن در این پارامتر ضرب و با پارامتر ReadTotalTimeoutConstant جمع می‌شود تا مجموع زمان مجاز عملیات خواندن مشخص شود.

پارامترهای ReadTotalTimeoutConstant و ReadTotalTimeoutMultiplier مانند پارامترهای بالا در عملیات نوشتن عمل می‌کنند.

مثلاً فراخوانی تابع فوق به صورت ;SetCommunicationTimeouts(10,500,0,0,0) مدت زمان ماکزیمم عملیات خواندن برای یک بلوک از داده‌ها را ۵۰۰ میلی‌ثانیه تعیین می‌کند. خروجی TRUE به معنای صحت انجام عملیات پیکربندی زمانی ارتباط سریال است.

تابع نوشتن در درگاه سریال

همانطور که قبلاً ذکر شد، برای نوشتن در درگاه سریال از تابع نوشتن در فایل استفاده می‌شود.

```
BOOL WriteByte(BYTE bybyte)
{
    DWORD iBytesWritten=0;
    BOOL  RetValue;

    RetValue = WriteFile(hComm,&bybyte,1,&iBytesWritten,NULL);

    if(RetValue == FALSE){
        printf("\nWriteByte Error Code = %d",GetLastError());
        return FALSE;
    }
    else
        return TRUE;
}
```

بایتی که می‌خواهیم به خروجی ارسال شود را به عنوان پارامتر ورودی به تابع ارسال می‌کنیم. پارامترهای تابع WriteFile نیز جالب توجه هستند. پارامتر اول دستگیره درگاه مورد نظر، پارامتر دوم اشاره‌گر به ابتدای بافر حاوی اطلاعاتی که می‌خواهیم نوشته شود و پارامتر سوم تعداد بایتهای مورد نظر برای نوشتن است. پارامتر چهارم بعد از اجرای دستور WriteFile تعداد بایتهایی که با موفقیت نوشته شده را در خود نگه می‌دارد. نکته جالب در مورد این تابع این است که لزومی به ارسال بایتهای اطلاعاتی به صورت تک‌تک در برنامه نیست و می‌توان با قرار دادن آنها در یک بافر، کل اطلاعات را با یک دستور WriteFile به درگاه فرستاد (که البته توسط خود سیستم تک‌تک به درگاه ارسال خواهد شد). تابعی که در بالا نوشته شده تنها یک

بایت را که به صورت پارامتر برای آن ارسال شده، در خروجی می‌نویسد. اگر عملیات تابع به صورت صحیح انجام شود، خروجی آن TRUE خواهد بود.

تابع خواندن اطلاعات از درگاه سریال

همانگونه که از بخش ارتباط سریال به خاطر دارید، برای ارسال اطلاعات به درگاه سریال کافی است اطلاعات مورد نظر در این درگاه نوشته شود. اما برای دریافت اطلاعات از درگاه سریال داستان کمی متفاوت است. برای این کار باید ابتدا از دریافت کاراکتر مطمئن شویم و سپس درگاه را بخوانیم.

تابع زیر برای اطمینان از دریافت کاراکتر نوشته شده است. اگر خروجی تابع TRUE باشد یعنی کاراکتر دریافت شده است:

```
BOOL RecvData(void)
{

    DWORD   dwErrorFlags;
    COMSTAT ComStat;
    BOOL  RetValue;

    RetValue = ClearCommError(hComm,&dwErrorFlags,&ComStat);
    if (RetValue == FALSE){
         printf("\nClearCommError Error Code = %d",GetLastError());
         return FALSE;
    }

    if  (ComStat.cbInQue == 0)
        return FALSE;
    else
        return TRUE;
}
```

ساختار COMSTAT در ویندوز می‌تواند مشخصات ارتباطی یک وسیله را نگهداری کند. متغیر ComStat که از این نوع تعریف شده با تابع ClearCommError مقداردهی می‌شود و فیلدهای آن اطلاعاتی مختلفی در مورد ارتباط سریال را نگهداری می‌کنند که ما از فیلد cbInQue استفاده کرده‌ایم. این فیلد حاوی تعداد بایتهایی است که دریافت شده، اما هنوز به کمک تابع ReadFile خوانده نشده است.

حال به تابع خواندن اطلاعات از درگاه سریال می‌پردازیم:

```
BOOL ReadByte(BYTE *resp)
{
    BYTE rx;
    *resp = 0;
    BOOL  RetValue;

    DWORD dwBytesRead = 0;

    RetValue = ReadFile(hComm, &rx, 1, &dwBytesRead, NULL);

    if (RetValue == TRUE)
    {
```

```
        if (dwBytesRead == 1)
        {
            *resp = rx;
            return TRUE;
        }

    }

    printf("\nReadByte Error Code = %d",GetLastError());
    return FALSE;
}
```

پس از آنکه به کمک تابع RecvData از دریافت کاراکتر مطمئن شدیم، به کمک تابع ReadByte آن را می‌خوانیم. این تابع مانند قبل، از تابع خواندن فایل (ReadFile) برای دریافت اطلاعات از درگاه استفاده می‌کند. پارامتر چهارم تابع ReadFile تعداد بایتهای دریافتی را پس از اجرای تابع فوق نشان می‌دهد که اگر مقدار آن برابر یک باشد، یعنی یک بایت دریافت شده و بایت فوق که در پارامتر دوم تابع قرار گرفته مورد استفاده قرار می‌گیرد. خروجی TRUE نشان‌دهنده صحت دریافت اطلاعات است.

بستن درگاه

تابع زیر درگاه ارتباطی را می‌بندد:

```
void ClosePort()
{
    BOOL  RetValue;
    RetValue = CloseHandle(hComm);
    if (RetValue == FALSE)
        printf("\nReadByte Error Code = %d",GetLastError());
}
```

توابع فوق را در سرآیندی به نام ComPort.h نوشته و در برنامه‌های بعدی از آنها استفاده می‌کنیم. مانند قبل، توابع فوق با فرض استفاده از برنامه‌های مد کنسول نوشته شده‌اند. اگر مایل به نوشتن برنامه‌های تحت ویندوز هستید می‌توانید دستورات مد کنسول (مثلاً printf) را با توابع ویندوز (مثلاً MessageBox) عوض کنید.

راه بهتر، نوشتن این توابع به صورت یک کلاس است که استفاده از آنها در برنامه‌های شیءگرا و تغییر و به‌روز کردن آنها ساده‌تر باشد. به کمک برنامه‌نویسی MFC یا API حتی می‌توانید برنامه‌های حرفه‌ای مبتنی بر پیغام[1] بنویسید.

مثالی از ارسال داده‌ها به درگاه سریال کامپیوتر

برنامه زیر را در محیط کنسول Visual Studio بنویسید و اجرا کنید. در سوی مقابل، باید سخت‌افزاری وجود داشته باشد (مثلاً یک میکروکنترلر) که اطلاعات را به صورت سریال (با همین پیکربندی) از درگاه کامپیوتر دریافت کند و روی یکی از درگاههای میکروکنترلر نمایش

[1] ***Message-Based***

دهد. اگر به این درگاه میکروکنترلر هشت عدد LED متصل کنید، باید شمارش اعداد صفر تا ۱۵ را به صورت متوالی روی LEDها مشاهده کنید.

```
#include "stdafx.h"
#include <windows.h>
#include <ConIO.h>
#include "ComPort.h"

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

  OpenPort("COM1");
  ConfigurePort(CBR_9600,
                8,
                TRUE,
                NOPARITY,
                ONESTOPBIT);

SetCommunicationTimeouts(10,0,500,0,0);

//**************************
  BYTE data = 0;

  while (!kbhit()){

      WriteByte(data);
      Sleep(500);

      if (data == 15)
        data = 0;
      else
        data++;
  }
//*************************

  ClosePort();
  return 0;
}
```

برای تست این برنامه راه ساده‌تری نیز وجود دارد. اگر بورد اصلی کامپیوتر شما دارای دو درگاه COM باشد، می‌توانید به کمک یک کابل Null Modem (که در بخش ارتباط سریال از آن سخن گفتیم)، دو درگاه سریال را به هم متصل کنید. اگر در این حالت دو پنجره HyperTerminal باز کنید و یکی را به درگاه COM1 و دیگری را به درگاه COM2 نسبت داده و با پیکربندی یکسان تنظیم کنید، هرآنچه روی یکی از دو پنجره تایپ کنید در پنجره دیگر دیده خواهد شد.

برای تست برنامه بالا، پنجره HyperTerminal که به درگاه COM1 متصل است را ببندید تا تنها پنجره منتسب به درگاه COM2 باز باشد. اگر چنین نکنید، برنامه بالا نمی‌تواند درگاه COM1 را باز کند و خطای شماره ۵ (ERROR_ACCESS_DENIED) رخ می‌دهد.

حال اگر برنامه فوق را اجرا کنید، داده‌هایی که برنامه شما از طریق درگاه COM1 ارسال می‌کند را روی پنجره HyperTerminal متعلق به درگاه COM2 مشاهده خواهید کرد. البته چون Hyperterminal اعدادی که دریافت می‌کند را به فرمت ASCII نمایش می‌دهد، اطلاعات قابل فهمی روی پنجره HyperTerminal نمی‌بینید. برای حل این مشکل، در قسمتی از برنامه بالا که بین دو خط ستاره مشخص شده، عدد 0 را با کاراکتر 'A' و عدد 15 را

با کاراکتر 'Z' جایگزین کنید. با این کار تا زمانی که برنامه ادامه دارد، حروف 'A' تا 'Z' را روی پنجره HyperTerminal خواهید دید.

اگر بورد اصلی کامپیوتر شما تنها یک درگاه COM داشته باشد، به ناچار باید از دو کامپیوتر استفاده کنید؛ روی یکی برنامه خودتان و روی دیگری برنامه HyperTerminal را اجرا کنید. در هر دو طرف از درگاه COM1 استفاده کنید.

نکته جالب دیگر راجع به برنامه بالا این است که همانطور که در توضیح دستور WriteFile گفته شد، نیازی به ارسال داده‌ها به صورت تک‌به‌تک نیست و می‌توان کل آنها را با هم ارسال کرد. برنامه زیر، نسخه اصلاح شده برنامه بالا است:

```
#include "stdafx.h"
#include <windows.h>
#include <ConIO.h>
#include "ComPort.h"

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

  OpenPort("COM1");
  ConfigurePort(CBR_9600,
                8,
                TRUE,
                NOPARITY,
                ONESTOPBIT);
  SetCommunicationTimeouts(10,0,500,0,0);

  BYTE data[16] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

  DWORD num;

  WriteFile(hComm,data,16,&num,NULL);

  printf("\n Numberof written bytes = %d\n",num);
  ClosePort();
  return 0;
}
```

دستور WriteFile، ۱۶ بایت (پارامتر سوم) از اطلاعات آرایه data (پارامتر دوم) را به درگاهی که دستگیره آن در پارامتر اول مشخص شده ارسال می‌کند. سیستم‌عامل پس از ارسال داده‌ها، تعداد واقعی بایتهایی که توانسته ارسال کند را در متغیر num ذخیره می‌کند. با اجرای این برنامه، هر ۱۶ بایت اطلاعات پشت سر هم ارسال می‌شوند. طبیعی است که سخت‌افزار مقابل هم باید طوری برنامه‌ریزی شود که بتواند این رشته بایت (که بدون استفاده از روشهای کنترل جریان داده ارسال می‌شوند) را دریافت کند. برای تست این برنامه به شیوه‌ای که قبلاً گفته شد، می‌توانید دو درگاه COM را به هم متصل کنید. اگر آرایه data را با مقادیر

کاراکتری مقداردهی کنید، کل رشته ارسالی را در پنجره HyperTerminal گیرنده مشاهده خواهید کرد.

مثالی از دریافت داده‌ها از درگاه سریال کامپیوتر

برنامه مقابل را در محیط کنسول Visual Studio بنویسید و اجرا کنید. در سوی مقابل، باید سخت‌افزاری وجود داشته باشد (مثلاً یک میکروکنترلر) که اطلاعاتی را به صورت سریال (با همین پیکربندی) بایت به بایت برای کامپیوتر ارسال کند. برنامه مقابل منتظر می‌ماند تا یک بایت اطلاعات دریافت کند و سپس آن را به فرمت شانزده‌تایی نمایش می‌دهد. اجرای برنامه با فشار یک کلید به پایان می‌رسد.

```
#include "stdafx.h"
#include <ConIO.h>
#include "ComPort.h"

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

  OpenPort("com1");
  ConfigurePort(CBR_9600,
                8,
                TRUE,
                NOPARITY,
                ONESTOPBIT);
  SetCommunicationTimeouts(10,0,500,0,0);

  BYTE data = 0;

  while (1){

      while (!RecvData() && !kbhit());
       if (kbhit())
            break;
       else{
            ReadByte(&data);
            printf("%X",data);
          }
  }
  ClosePort();
  return 0;
}
```

می‌توانید به شیوه قبل با اتصال دو درگاه COM کامپیوتر به هم، این برنامه را تست کنید. در این حال، برنامه بالا هرچه را در برنامه HyperTerminal تایپ کنید، در دستگاه شانزده‌تایی نمایش خواهد داد.

```
#include "stdafx.h"
#include <ConIO.h>
#include "ComPort.h"

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

  OpenPort("com1");
  ConfigurePort(CBR_9600,
                8,
                TRUE,
                NOPARITY,
                ONESTOPBIT);
  SetCommunicationTimeouts(10,0,500,0,0);

  BYTE data[16];
  while(!RecvData()); //Wait for reception
  DWORD num = 0;
  ReadFile(hComm,data,16,&num,NULL);
  for (int i = 0;i < num;i++)
       printf("%c",data[i]);
  getch();
  ClosePort();
  return 0;
}
```

حال برنامه مقابل را ببینید. در این برنامه، ابتدا منتظر می‌مانیم تا اطلاعاتی دریافت شود و سپس به کمک تابع

ReadFile، ۱۶ بایت اطلاعات را به صورت متوالی خواهیم خواند. اگر این برنامه و برنامه ارسال داده‌ها به کمک WriteFile (که قبلاً گفته شد) به شیوه اتصال درگاههای COM تست شود، شانزده بایتی که برنامه فرستنده ارسال می‌کند توسط برنامه گیرنده دریافت و نمایش داده خواهد شد. فراموش نکنید که ابتدا برنامه گیرنده و سپس برنامه فرستنده را اجرا کنید.

منابع

- http://www.beyondlogic.org/porttalk/porttalk.htm
- Microsoft Windows NT Device Driver Kit
- Microsoft Win32 SDK
- Intel Architecture Developer's Manual - Basic Architecture, Order Number 243190.
- Intel Architecture Developer's Manual - Instruction Set Reference Manual, Order Number 243191.
- Intel Architecture Developer's Manual - System Programming Guide, Order Number 243192.
- http://www.codeproject.com/KB/system/cserialcom.aspx