CURSO DE ESPECIALIZAÇÃO EM AUTOMAÇÃO INDUSTRIAL

Fundamentos e Aplicações de Sistemas de Automação

Módulo 5:
Entrada e Saída

"The world is moving so fast these days that the man who says it can't be done is generally interrupted by someone doing it" (Elbert Hubbard)

Professor: Constantino Seixas Filho

sábado, 9 de outubro de 2004

1

Síncrono x Assíncrono

- No I/O síncrono nós disparamos uma operação e ficamos esperando pelo resultado.
- No I/O assíncrono várias operações podem ser disparadas e o resultado só é recebido quando habilitarmos esta opção no futuro.

2

Create File

```
HANDLE CreateFile
LPCTSTR lpFileName,                          // Nome do arquivo
DWORD dwDesiredAccess,                       // Tipo de acesso.
DWORD dwShareMode,                           // Compartilhamento
LPSECURITY_ATTRIBUTES lpSecurityAttributes,  // Atributos de segurança
DWORD dwCreationDistribution,                // Criação ou abertura
DWORD dwFlagsAndAttributes,                  // Atributos e Flags
HANDLE hTemplateFile                         // Gabarito de atributos
);
```

Retorno:

Status	Interpretação
Handle para o Arquivo criado	Sucesso
INVALID_HANDLE_VALUE	Falha

3

CreateFile

lpFileName	Nome do arquivo a ser aberto ou criado
dwDesiredAccess	GENERIC_READ: abre apenas para leitura GENERIC WRITE: abre apenas para escrita GERNERIC_READ|GENERIC_WRITE: abre para leitura e escrita
dwShareMode	Indica como o arquivo será compartilhado: 0: Nenhuma outra abertura pode ser efetuada FILE_SHARE_READ: só aberturas para leitura serão permitidas FILE_SHARE_WRITE: só aberturas para escrita serão permitidas FILE_SHARE_READ|FILE_SHARE_WRITE: quaisquer operações de aberturas serão permitidas
lpSecurityAttributes	Apontador para descritor de segurança para o handle do novo arquivo (contém a ACL do novo handle). Se NULL, será utilizada estrutura default e o handle para o arquivo não será herdável.
dwCreationDistribution	CREATE_NEW: Tenta ciar um novo arquivo. Retorna erro se já existir CREATE_ALWAYS: Cria novo arquivo em qualquer situação. OPEN_EXISTING: Abre arquivo. Se não existir retorna erro. OPEN_ALWAYS: Tenta abrir um arquivo. Se não existir é criado. TRUNCATE_EXISTING: Abre arquivo existente truncando tamanho para 0 bytes. Se não existir: erro.
dwFlagsAndAttributes	Atributos: Usados na criação do arquivo: FILE_ATTRIBUTE_HIDDEN: Arquivo oculto. Não aparece em diretórios. FILE_ATTRIBUTE_SYSTEM: O arquivo é usado apenas pelo S.O. FILE_ATTRIBUTE_READONLY: Só pode ser lido. FILE_ATTRIBUTE_TEMPORARY: arquivo será mantido na memória. FILE_ATTRIBUTE_NORMAL: deve ser usado sozinho. Flags: Alteram o modo como as operações de escrita e leitura são realizadas: FILE_FLAG_RANDOM_ACCESS: otimiza acesso randômico. FILE_FLAG_SEQUENTIAL_SCAN: otimiza acesso seqüencial. FILE_FLAG_DELETE_ON_CLOSE: O arquivo será deletado quando for fechado. **FILE_FLAG_OVERLAPPED: habilita operações a**ssíncronas sobre o arquivo.
hTemplateFile	Especifica handle para arquivo que será usado como gabarito de atributos ou NULL.

4

Outras funções

```
BOOL DeleteFile(
LPCTSTR lpFileName);          // Nome do arquivo
```

Retorno:

Status	Interpretação
TRUE	Sucesso
FALSE	Falha

Outras:

- **CopyFile**
- **MoveFile**
- **Etc.**

5

ReadFile

```
BOOL ReadFile(
HANDLE hFile,                 // Handle para o arquivo
LPVOID lpBuffer,              // Buffer para onde os dados serão lidos.
DWORD nNumberOfBytesToRead,   // Quantidade de bytes a serem lidos.
LPDWORD lpNumberOfBytesRead,  // Número de bytes lidos.
LPOVERLAPPED lpOverlapped);   // Apontador para estrutura OVERLAPPED.  NULL para síncrono
```

Retorno:

Status	Interpretação	
!=0	Sucesso	
0	Falha. Faça Erro = GetLastError()	
	Analise Erro	
	ERROR_IO_PENDING	Pedido foi enfileirado
	ERROR_HANDLE_EOF	Ponteiro além do EOF
	ERROR_INVALID_USER_BUFFER	Excesso de pedidos
	ERROR_NOT_ENOUGH_MEMORY	Excesso de pedidos
	ERROR_NOT_ENOUGH_QUOTA	Excesso de pedidos
	ERROR_MORE_DATA	Operação com Pipes[1]: Msg maior que número de bytes especificado na leitura
	ERROR_INSUFICIENT_BUFFER	Leitura de Mailslot[1] com buffer muito pequeno
	ERROR_BROKEN_PIPE	Tentativa de leitura em um Pipe[1] anônimo de escrita que foi fechado.

6

WriteFile

```
BOOL WriteFile(
HANDLE hFile,                    // Handle para o arquivo
LPVOID lpBuffer,                 // Buffer com dados a serem escritos.
DWORD nNumberOfBytesToWrite,     // Quantidade de bytes a serem escritos.
LPDWORD lpNumberOfBytesWritten,  // Número de bytes escritos.
LPOVERLAPPED lpOverlapped        // Apontador para estrutura OVERLAPPED.
                                 // NULL para operações síncronas.
);
```

Retorno:

Status	Interpretação
!=0	Sucesso
0	Falha

7

SetFilePointer

```
DWORD SetFilePointer(
HANDLE hFile,                    // Handle para o arquivo
LONG lDistanceToMove,            // Deslocamento (palavra de menor ordem).
PLONG lpDistanceToMoveHigh,      // Deslocamento (palavra de maior ordem).
                                 // Retorna palavra de maior ordem da nova posição do apontador do arquivo.
DWORD dwMoveMethod               // Ponto de referência para posicionamento: FILE_BEGIN
                                 // FILE_END
                                 // FILE_CURRENT
);
```

Retorno:

Status	Interpretação
Palavra de menor ordem da nova posição do apontador do arquivo.	Sucesso
0xFFFFFFFF	Falha

- **Tanto valores positivos como negativos de deslocamento podem ser passados em DistanceToMove.**

8

FlushFileBuffers

A função *FlushFileBuffers()* é usada para forçar a escrita de dados do buffer no disco

```
DWORD FlushFileBuffers(
HANDLE hFile,                    //  Handle para o arquivo
);
```

Retorno:

Status	Interpretação
<>0	Sucesso
0	Falha

9

Acesso a arquivos com Exclusão Mútua

```
BOOL LockFile(
HANDLE hFile,                    //  Handle para o arquivo
DWORD dwFileOffsetLow,           //  Posição da região a trancar
DWORD dwFileOffsetHigh,          //
DWORD nNumberOfBytesToLockLow,   //  Número de bytes a trancar
DWORD nNumberOFBytesToLockHigh); //              .
```

```
BOOL UnlockFile(
HANDLE hFile,                    //  Handle para o arquivo
DWORD dwFileOffsetLow,           //  Posição da região a destrancar
DWORD dwFileOffsetHigh,          //
DWORD nNumberOfBytesToLockLow,   //  Número de bytes a destrancar
DWORD nNumberOFBytesToLockHigh); //              .
```

Retorno:

Status	Interpretação
TRUE	Sucesso
FALSE	Falha

10

Acesso assíncrono a arquivos

1. **Ativar a flag FILE_FLAG_OVERLAPED na criação do arquivo**
2. **Definir a estrutura OVERLAPPED na emissão de ReadFile ou WriteFile**

```
typedef struct _OVERLAPPED {
        DWORD  Internal;
        DWORD  InternalHigh;
        DWORD  Offset;
        DWORD  OffsetHigh;
        HANDLE hEvent;
} OVERLAPPED;
```

Internal	Reservado
InternalHigh	Reservado. Representa o tamanho do bloco de dados transferido se *GetOverlappedResult()* retorna TRUE.
Offset	Offset em bytes do início do arquivo para leitura e escrita
OffsetHigh	Offset: palavra alta
hEvent	Handle para objeto do tipo Evento com reset manual a ser sinalizado ao final da operação assíncrona. Mesmo que a leitura aconteça imediatamente, sem enfileiramento, o evento será sinalizado.

11

Uso de overlap com sinalização de handles para arquivos

```
hFile= CreateFile(..., FILE_IO_OVERLAPPED, ...);
ReadFile(hFile, ...);
< Use o tempo livre para realizar outras tarefas>
WaitForSingleObject(hFile, INFINITE);
```

12

Uso de overlap com sinalização de Eventos

1. **Abrir arquivo com flag FILE_FLAG_OVERLAPPED ativada.**
2. **Inicializar estrutura OVERLAPPED em uma variável global. Esta estrutura identifica a operação de IO em execução. Indique os eventos a serem sinalizados.**
3. **Realizar a operação de leitura ou escrita passando o endereço da estrutura como parâmetro para *ReadFile()* ou *WriteFile()*.**
4. **Realizar alguma função útil**
5. **Esperar pelo término das operações solicitadas através das funções *Wait…()*.**
6. **Chamar a função *GetOverlappedResult*() para conformar o resultado das operações assíncronas.**

GetOverlappedResult

```
BOOL GetOverlappedResult(
HANDLE hFile,                          //  Handle para o objeto
LPOVERLAPPED lpOverlapped,             //  Apontador para variável do tipo OVERLAPPED
DPDWORD lpNumberOfBytesTransfered,     //  Apontador para retornar número de bytes transferidos.
BOOL bWait                             //  TRUE: espera pelo fim da operação.
);
```

Retorno:

Status	Interpretação
TRUE	Sucesso
FALSE	Falha

Uso de overlap com sinalização de Eventos - Exemplo

```
#include "GetOSVersion.h"

typedef struct {
     char nome[30];
     char telefone[16];
} Cadastro;

#define ACESSOS_SIMULTANEOS      4
OVERLAPPED overlap[ACESSOS_SIMULTANEOS];

int main()
{
     HANDLE hFile;
     HANDLE hEvents[ACESSOS_SIMULTANEOS];
     Cadastro MyBuffer[ACESSOS_SIMULTANEOS];
     DWORD BytesRead, error;
     BOOL Status;
     int leitura;                    // índice da leitura
     DWORD ret;
     int i;
     int nProc;                      // Número de processadores
     char strVer[30];                // Versão do sistema operacional
```

15

Uso de overlap com sinalização de Eventos - Exemplo

```
     // Testa se o sistema operacional é o NT
     Status = GetOSVersion(strVer, &nProc);
     printf("Versao do SO = %s  Processadores = %d\n", strVer, nProc);
     if (!Status) {                  // Não é NT. Não é NT ...
             printf("Falha: Sistema operacional NAO E O NT\n");
             return 0;
     } // if

     / Cria eventos com reset Manual
     for (i=0; i< ACESSOS_SIMULTANEOS; ++i) {
             hEvents[i]= CreateEvent(NULL, TRUE, FALSE, NULL);
             CheckForError(hEvent[i]);
     } //for

     // Abre arquivo
     hFile= CreateFile("c:\\Livro\\Arquivos\\MyClients.arq",
                       GENERIC_READ,
                       FILE_SHARE_READ|FILE_SHARE_WRITE,    // abre: leitura e escrita
                       NULL,                                // atributos de segurança
                       OPEN_EXISTING,                       // abre arquivo já existente
                       FILE_FLAG_OVERLAPPED,                // acesso assíncrono
                       NULL);                               // Template para atributos e flags
     CheckForError(hFile != INVALID_HANDLE_VALUE);
```

16

Uso de overlap com sinalização de Eventos - Exemplo

```
for (leitura=0; leitura<ACESSOS_SIMULTANEOS; ++leitura) {
        // Monta estrutura overlap para cada leitura
        overlap[leitura].OffsetHigh = 0;
        overlap[leitura].Offset= leitura * sizeof(Cadastro);
        overlap[leitura].hEvent= hEvent[leitura];
        // Solicita leitura do dado - Não funciona no Windows 95 e 98
        Status = ReadFile(hFile, &MyBuffer[leitura], sizeof(Cadastro), &BytesRead, &overlap[leitura]);
        if (Status) {
        // printf("\nLeitura %d realizada sem overlap", leitura);
        // printf("\nleitura %d lidos: %d bytes\nNome=%s Telefone= %s\n",
        // leitura,BytesRead,MyBuffer[leitura].nome,
        // MyBuffer[leitura].telefone);
        }  // if
        else {
                error = GetLastError();
                if (error == ERROR_IO_PENDING) // IO Assíncrono enfileirado
                    printf("Leitura %d enfileirada\n", leitura);
                    else printf("Erro fatal\n");
        } // else
}  // for

// Espera por todos os eventos
ret = WaitForMultipleObjects(ACESSOS_SIMULTANEOS, hEvents, TRUE, INFINITE);
CheckForError((ret >= WAIT_OBJECT_0) && (ret < WAIT_OBJECT_0 + ACESSOS_SIMULTANEOS));
```

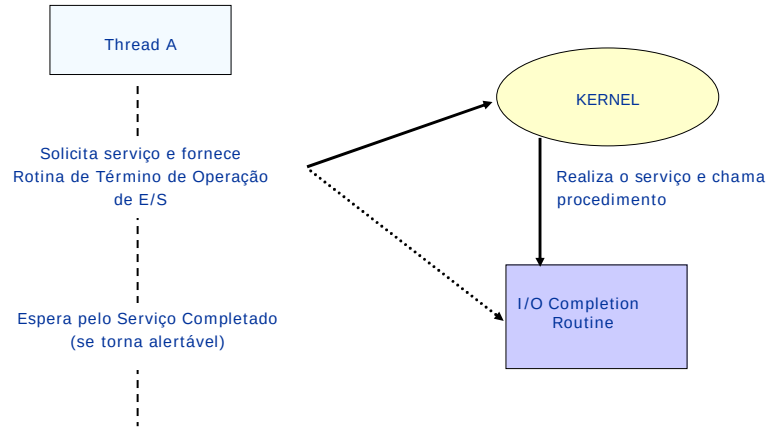
7

Uso de overlap com sinalização de Eventos - Exemplo

```
// Mostra resultado
for (i=0; i<ACESSOS_SIMULTANEOS; ++i)
{
        DWORD BytesRead;
        Status = GetOverlappedResult(hFile, &overlap[i], &BytesRead,FALSE);
        // Caso pegue evento a evento o índice será ret - WAIT_OBJECT_0
        printf("\nleitura %d concluida: %d bytes\nNome=%s Telefone= %s\n",
                i, BytesRead, MyBuffer[i].nome, MyBuffer[i].telefone);
        CloseHandle(hEvent[i]);
} // for
CloseHandle(hFile);
_getch();           // espera usuário para sair
return EXIT_SUCCESS;
}  // main
```

18

Asynchronous Procedure Call



19

WaitForSingleObjectEx

```
DWORD WaitForSingleObjectEx(
HANDLE hHandle,          //  Handle para um objeto do kernel
DWORD dwMilliseconds,    //  Tempo máximo que desejamos esperar
BOOL bAlertable          //  TRUE: se torna alertável
);
```

Retorno:

Status	Interpretação
WAIT_OBJECT_0	Objeto foi sinalizado
WAIT_TIMEOUT	Ocorreu timeout
WAIT_ABANDONED	Uma thread proprietária de um Mutex realiza ExitThread() sem liberá-lo.
WAIT_FAILED	Função falhou
WAIT_IO_COMPLETION	Uma Rotina de Término de E/S foi enfileirada para execução.

Esta função irá retornar em três situações:

- **Quando o objeto for sinalizado**
- **Quando o intervalo de *timeout* expirar**
- **Quando a Rotina de Término de E/S for acionada ou estiver enfileirada na thread**

20

Rotina de Término de Operação em arquivo

```
VOID WINAPI FileIOCompletionRoutine(
DWORD dwErrorCode,                  // Código de conclusão
                                    //  0 : Sucesso
                                    //  ERROR_HANDLE_EOF:  leitura ultrapassou EOF
DWORD dwNumberOfBytesTransfered,    // Número de bytes transferidos
LPOVERLAPPED lpOverlapped);         // Apontador para estrutura com informação de E/S
```

21

ReadFileEx

```
BOOL ReadFileEx(
HANDLE hFile,                                   // Handle para o arquivo
LPVOID lpBuffer,                                // Buffer para onde os dados serão lidos.
DWORD nNumberOfBytesToRead,                     // Quantidade de bytes a serem lidos.
LPOVERLAPPED lpOverlapped                       // Apontador para estrutura OVERLAPPED.
LPLPOVERLAPPED_COMPLETION_ROUTINE               // Endereço da Rotina de término de I/O
lpCompletionRoutine);
```

Retorno:

Status	Interpretação
!=0	Operação de E/S está pendente
0	Falha. Faça Erro = GetLastError() A análise é semelhante à da função *ReadFile()*.

22

Uso de overlap com APCs

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define _CHECKERROR1            // Ativa função CheckForError
#include "CheckForError.h"
#include "GetOSVersion.h"

typedef struct {
     char nome[30];
     char telefone[16];
} Cadastro;

#define ACESSOS_SIMULTANEOS  3
long nNumSolicitacoes;          // Número de operações enfileiradas
OVERLAPPED overlap[ACESSOS_SIMULTANEOS];
Cadastro MyBuffer[ACESSOS_SIMULTANEOS]; // Buffer para leitura
HANDLE hFile;
HANDLE hEventIODone;            // Sinaliza final de todas as operações de E/S
```

23

Uso de overlap com APCs

```
VOID WINAPI FileIOCompletionRoutine(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered,
     LPOVERLAPPED lpOverlapped)
{    int iIndice;

     iIndice = (int) lpOverlapped->hEvent;
     if (dwErrorCode ==0) { // Sucesso
       printf("Leitura %d completada. %d bytes lidos\n", iIndice, dwNumberOfBytesTransfered);
       printf("Nome = %s  Telefone = %s\n", MyBuffer[iIndice].nome, MyBuffer[iIndice].telefone);
     }  // if
     else printf("Leitura %d retornou erro %d\n", iIndice, dwErrorCode);
     if (InterlockedDecrement(&nNumSolicitacoes) == 0)  SetEvent(hEventIODone);
} //FileIOCompletionRoutine
```

24

Uso de overlap com APCs

```
VOID SolicitaLeitura(int iIndice, DWORD dwTamRegistro)
{
    BOOL bStatus;
    DWORD dwError;
    // Monta estrutura overlap para cada leitura
    overlap[iIndice].OffsetHigh = 0;
    overlap[iIndice].Offset= iIndice * dwTamRegistro;
    // Guarda o índice ao invés de hEvent: será útil quando a leitura completar
    overlap[iIndice].hEvent= (HANDLE) iIndice;

    // Solicita leitura do dado - Não funciona no Windows 95 e 98
    bStatus = ReadFileEx(hFile, &MyBuffer[iIndice], dwTamRegistro, &overlap[iIndice], FileIOCompletionRoutine);

    InterlockedIncrement(&nNumSolicitacoes);
     // Incrementa número de solicitações realizadas

    if (bStatus) // Operação ficou pendente
            printf("Leitura %d ficou pendente ...\n", iIndice);
    else {
            dwError = GetLastError();
            printf("Erro fatal %d na Rotina Solicita Leitura\n", dwError);
    }
} // SolicitaLeitura
```

25

Uso de overlap com APCs

```
int main()
{   int iPedido;        // Ordem do pedido de leitura
    int nProc;          // Número de processadores
    char strVer[30];    // Versão do sistema operacional
    BOOL bRet;

    // Define o título da janela
    SetConsoleTitle("Programa 5.4 - APC");
    bRet = GetOSVersion(strVer, &nProc);
    printf("Versao do SO = %s  Processadores = %d\n", strVer, nProc);
    if (!bRet) {        // Não é NT. Não é NT ...
            printf("Falha: Sistema operacional NAO E O NT\n");
            printf("\nAperte uma tecla...\n");
            _getch();   // Espera usuário
            return 0;   } // if

    hEventIODone= CreateEvent(NULL, TRUE, FALSE,NULL); // Cria evento com reset Manual
    CheckForError(hEventIODone);
    hFile= CreateFile("c:\\Livro\\Programas\\Arquivos\\MyClients.arq", // Abre arquivo
            GENERIC_READ, FILE_SHARE_READ|FILE_SHARE_WRITE,   // abre para leitura e escrita
            NULL,                                   // Atributos de segurança
            OPEN_EXISTING,                          // abre arquivo já existente
            FILE_FLAG_OVERLAPPED,                   // acesso assíncrono
            NULL);                                  // Template para atributos e flags
    CheckForError(hFile != INVALID_HANDLE_VALUE);
```

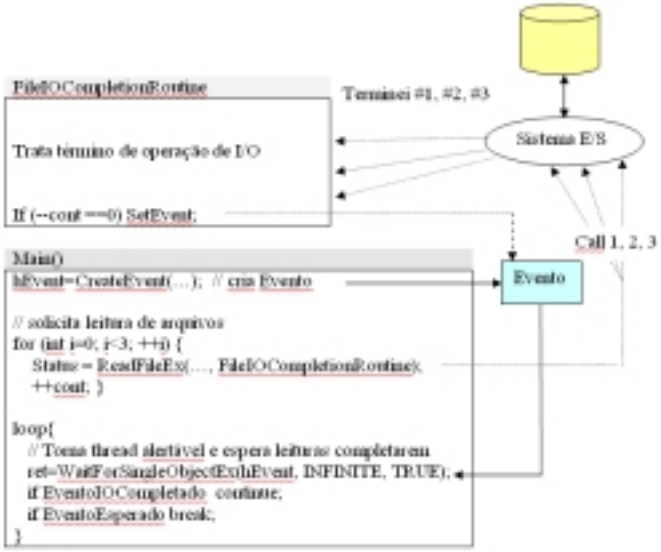
26

Uso de overlap com APCs

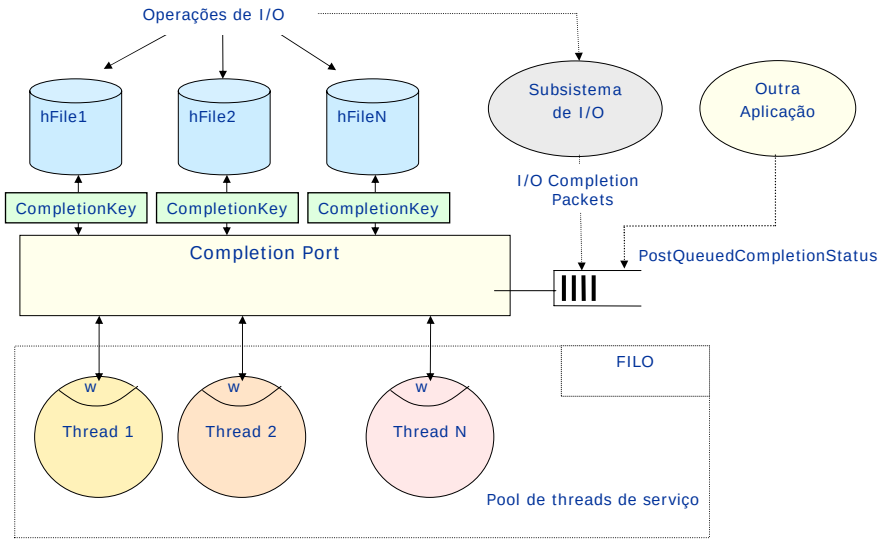
```
    for (iPedido=0; iPedido < ACESSOS_SIMULTANEOS; ++iPedido)
            SolicitaLeitura(iPedido, sizeof(Cadastro));

    // Espera pelos eventos
    for (; ; ) {
        DWORD bRet;
        bRet = WaitForSingleObjectEx(hEventIODone,INFINITE, TRUE);
        if (bRet == WAIT_IO_COMPLETION)
            {           printf("IOCompletion\n");
                        continue;  // Uma ou mais APCs terminaram
            }
            if (bRet == WAIT_OBJECT_0) {
                        printf( "Todas as leituras efetuadas\n" );
                        break;      // Evento sinalizado
            }
    }
    CloseHandle(hFile);
    printf("\nAperte uma tecla...\n");
    _getch();           // Espera usuário
    return EXIT_SUCCESS;
}  // main
```


Uso de overlap com APCs



Portas de Conclusão de E/S (I/O Completion Ports)



29

Completion Ports

Passos da operação:

- **Criar uma *Completion Port***
- **Associar Arquivos à *Completion Port*.**
- **Criar um conjunto de threads de serviço exclusivas para este fim.**
- **Colocar cada thread à espera na *Completion Port*.**
- **Emitir solicitações de leitura e escrita dos arquivos com *overlap*.**
- **Tratar as notificações nas threads de serviço.**

30

CreateIoCompletionPort

```
HANDLE CreateIoCompletionPort(
HANDLE FileHandle,                  // Handle para o arquivo
HANDLE ExistingCompletionPort,      // Handle para a Completion Port
DWORD CompletionKey,                // Chave de identificação dos pacotes, definido pelo usuário por arquivo.
DWORD NumberOfConcurrentThreads     // Número de threads concorrentes
);
```

Retorno:

Status	Interpretação
NULL	Falha
!=NULL	Handle para a *Completion Port* criada ou aberta.

31

GetQueuedCompletionStatus

```
HANDLE GetQueuedCompletionStatus(
HANDLE CompletionPort               // Handle para completion port
LPWORD lpNumberOfBytesTransfered,   // Apontador para receber número de bytes transferidos
LPDWORD lpCompletionKey,            // Apontador para receber chave
LPOVERLAPPED *lpOverlapped          // Endereço para receber apontador para estrutura overlapped
DWORD dwMilliseconds                // Timeout
);
```

Retorno:

Status	Interpretação
TRUE	Sucesso
FALSE	Falha. Se um packet para operação que falhou é desempilhado, então lpOverlapped aponta pacote com descrição da operação que falhou. Se chamada falhou então lpOverlapped = NULL Faça Erro = *GetLastError()* Se houve timeout: Erro = WAIT_TIMEOUT

32

Realizando operações de I/O sem notificação

- Mesmo que o handle do arquivo esteja associado a uma *completion port*, com uma estrutura OVERLAPPED válida, é possível impedir que uma *completion port* seja notificada. Para isto devemos especificar um handle de evento hEvent válido na estrutura OVERLAPPED e ativar o seu bit menos significativo
- overlap.hEvent = (HANDLE) ((DWORD)hEvent | 0x0001);
- Isto deve ser feito para cada operação de leitura ou escrita sem notificação

33

O pulo do gato: como recuperar os resultados após a operação

```
typedef struct {
    OVERLAPPED  Overlap;
    LONG        lOpIndex;
    Cadastro    Buffer;
} IoTransaction;
```

34

Exemplo – Completion Ports

```
#define _CHECKERROR1            // Ativa função CheckForError
#include "CheckForError.h"
#include "GetOSVersion.h"

// Protótipos de funções
BOOL GetOSVersion(char *, int *);
DWORD WINAPI IOServiceFunc(LPVOID);
typedef struct {
    char nome[30];
    char telefone[16];
} Cadastro;

#define  TAM_ARQUIVO                4
#define  ACESSOS_SIMULTANEOS        6
#define  NUM_ARQ                    2
#define  NUM_THREADS                3

// Define estrutura IoTransaction a ser usado em toda transação
typedef struct {
    OVERLAPPED      Overlap;
    LONG            lOpIndex;
    Cadastro        Buffer;
} IoTransaction;
```

35

Exemplo – Completion Ports

```
HANDLE hFile[NUM_ARQ];                  // Handle para arquivos
HANDLE hCPort;                          // Handle para I/O Completion Port
LONG lOperacao;                         // Conta número de operações pendentes
HANDLE hEventos[NUM_THREADS];           // Eventos para dizer que a thread já está pronta
HANDLE hDone;                           //  Todas as operações foram concluídas

VOID SolicitaLeitura(LONG lIndice, int nArquivo, int Registro, DWORD dwTamRegistro)
{
  BOOL bStatus;
  DWORD dwBytesLidos;           // Número de bytes lidos
  IoTransaction *pIO;           // Apontador para IO packet
  printf("\n-->Solicitando leitura: Indice: %02d nArquivo %02d Registro %02d", lIndice, nArquivo, Registro);
  pIO = (IoTransaction *)calloc(1, sizeof(IoTransaction));
  if (pIO != NULL) {    // Monta estrutura overlap para cada leitura
    pIO->Overlap.OffsetHigh = 0;
    pIO->Overlap.Offset= Registro * dwTamRegistro;
    pIO->Overlap.hEvent= NULL;
    pIO->lOpIndex = lIndice;
    // Solicita leitura do dado - Não funciona no Windows 95
    bStatus = ReadFile(hFile[nArquivo], (void *)&pIO->Buffer, dwTamRegistro, &dwBytesLidos, &pIO-
    >Overlap);
    CheckForError(bStatus);
    InterlockedIncrement(&lOperacao); }  // Incrementa operações pendentes
else printf("\nOverflow de memoria");
} // SolicitaLeitura
```

36

Exemplo – Completion Ports

```
int main()
{
    LONG            lPedido;            // Ordem do pedido de leitura
    int             nProc;              // Número de processadores
    char            strVer[30];         // Versão do sistema operacional
    BOOL            bRet;
    int             nArquivo=0;         // Índice do arquivo a ser lido
    HANDLE          hThreads[NUM_THREADS];
    DWORD           dwThreadId;
    int             nLoop;
    DWORD           dwRet;
    DWORD           dwExitCode;

    bRet = GetOSVersion(strVer, &nProc);
    printf("\nVersao do SO = %s  Processadores = %d", strVer, nProc);
    if (!bRet) {        // Não é NT. Não é NT ...
        printf("\nFalha: Sistema operacional NAO E O NT");
        printf("\nAperte uma tecla...\n");
        _getch();   return 0;
    } // if
    hDone= CreateEvent(NULL, TRUE, FALSE, NULL);  // Operação terminou
    hCPort = CreateIoCompletionPort(            // Cria Completion Port
        INVALID_HANDLE_VALUE,
        NULL,                                   // Completion port existente
        0,                                      // Chave
        0);                                     // Número de threads = Número de processadores
    CheckForError(hCPort);
```


Exemplo – Completion Ports

```
// Abre arquivos
hFile[0]= CreateFile("c:\\Livro\\Programas\\Arquivos\\MyClients.arq",
        GENERIC_READ,
        FILE_SHARE_READ,    // compartilha para leitura
        NULL,               // atributos de segurança
        OPEN_EXISTING,      // abre arquivo já existente
        FILE_FLAG_OVERLAPPED, // acesso assíncrono
        NULL);              // Template para atributos e flags
CheckForError(hFile[0] != INVALID_HANDLE_VALUE);
hFile[1]= CreateFile("c:\\Livro\\Programas\\Arquivos\\MyClients2.arq",
        GENERIC_READ,
        FILE_SHARE_READ,    // compartilha para leitura
        NULL,               // atributos de segurança
        OPEN_EXISTING,      // abre arquivo já existente
        FILE_FLAG_OVERLAPPED, // acesso assíncrono
        NULL);              // Template para atributos e flags
CheckForError(hFile[1] != INVALID_HANDLE_VALUE);
// Associa arquivo com Completion Port: uma chamada para cada arquivo
for (nLoop=0; nLoop<NUM_ARQ; ++nLoop)
    hCPort = CreateIoCompletionPort(
        hFile[nLoop],       // Associa arquivo
        hCPort,             // Completion port existente
        nLoop,              // Chave: Identificação do arquivo
        0);                 // Número de threads = Número de processadores
```

Exemplo – Completion Ports

```
for (nLoop = 0; nLoop < NUM_THREADS; ++nLoop) {       // Cria pool de threads
    hThreads[nLoop] = (HANDLE) _beginthreadex(
                        NULL,       0,     (CAST_FUNCTION)IOServiceFunc,
                        (LPVOID)nLoop,                  // Identificação da thread
                        0,   (CAST_LPDWORD)&dwThreadId);
    CheckForError(hThreads[nLoop]);
    if (hThreads[nLoop]) printf("\nThread %d criada com Id= %0x ", nLoop, dwThreadId);
    hEventos[nLoop]= CreateEvent(NULL, TRUE, FALSE, NULL);
    CheckForError(hEventos[nLoop]);
} // for
// Espera threads iniciarem
dwRet = WaitForMultipleObjects(NUM_THREADS, hEventos, TRUE, INFINITE);
for (lPedido=0; lPedido<ACESSOS_SIMULTANEOS; ++lPedido)
    SolicitaLeitura(lPedido, nArquivo++ % NUM_ARQ, lPedido% TAM_ARQUIVO, sizeof(Cadastro));
    WaitForSingleObject(hDone, INFINITE); // Espera leitura terminar
    // Pede que as threads de trabalho terminem
    for (nLoop=0; nLoop<NUM_THREADS; ++nLoop) PostQueuedCompletionStatus(hCPort, 0, -1, NULL);
dwRet=WaitForMultipleObjects(NUM_THREADS,hThreads,TRUE, INFINITE);
CheckForError((dwRet >= WAIT_OBJECT_0) && (dwRet < WAIT_OBJECT_0 + NUM_THREADS));
for (nLoop=0; nLoop<3; ++nLoop) {
    GetExitCodeThread(hThreads[nLoop], &dwExitCode);
    printf("thread %d terminou: codigo=%d\n",nLoop,dwExitCode);
    CloseHandle(hThreads[nLoop]);              // apaga referência ao objeto
    CloseHandle(hEventos[nLoop]);
}  // for
```

39

Exemplo – Completion Ports

```
    for (nLoop=0; nLoop<NUM_ARQ; ++nLoop)
        CloseHandle(hFile[nLoop]);
    CloseHandle(hDone);
    CloseHandle(hCPort);
    printf("\nAperte uma tecla...\n");
    _getch();
    return EXIT_SUCCESS;
}  // main
```

40

Exemplo – Completion Ports

```
DWORD WINAPI IOServiceFunc(LPVOID index)
{    BOOL bRet;
     DWORD dwBytesLidos;
     DWORD dwCompletionKey;
     IoTransaction *lpIoTransaction;
     LONG lOperacaoLocal;
     SetEvent(hEventos[(int) index]); // Avisa que já está pronta

     for ( ; ; ) {
          bRet = GetQueuedCompletionStatus(          // Coloca Threads a espera
                    hCPort,                          // Completion Port
                    &dwBytesLidos,                   // Número de bytes lidos
                    &dwCompletionKey,                // Chave
                    (OVERLAPPED **)&lpIoTransaction, // Endereço do apontador e não apontador
                    INFINITE);                       // Timeout
          if (bRet) {  // Sucesso
            if (dwCompletionKey == -1) break;  // flag para abortar thread
            lpIoTransaction->lOpIndex; // indice da operacao efetuada
            printf("\nThread %d: Operacao %d no arquivo %d completada %d bytes lidos", (int)index,
                     lpIoTransaction->lOpIndex,  dwCompletionKey, dwBytesLidos);
            printf("\nNome = %s  Telefone = %s", lpIoTransaction->Buffer.nome,
                     lpIoTransaction->Buffer.telefone);
            free(lpIoTransaction);                                  // Libera memória alocada
            lOperacaoLocal = InterlockedDecrement(&lOperacao);      // Decrementa operações pendentes
            if (lOperacaoLocal == 0) SetEvent(hDone); } // If       // Terminou
```

41

Exemplo – Completion Ports

```
          else { // Erro
            if (lpIoTransaction == NULL) // Erro na chamada da função
                     printf("\nFuncao nao completou a chamada");
             else // Packet com falha de operação foi recebido
                     printf("\nFalha de leitura foi notificada");
            CheckForError(FALSE); // Imprime msg de erro: GetLastError();
           } // else
          // A tarefa é muito pequena. Da um tempo ou a thread 2 vai monopolizar
          Sleep(100);
     } // for
     _endthreadex((DWORD) index);
     return(0);
} // IOServiceFunc
```

42

Change Notification

- **No Windows NT existe um objeto do kernel denominado *Change Notification* que é sinalizado toda vez que um diretório ou árvore de diretórios é modificada**
- **Pode-se ficar a espera de um evento de modificação de diretório através das instruções *Wait...()* de forma similar ao utilizado com outros objetos de sincronização**
- **Este mecanismo é particularmente útil quando deseja-se realizar a comunicação de dois aplicativos através de arquivos**
- **O processo receptor perceberá que um novo arquivo contendo os parâmetros de interesse foi carregado e efetuará a leitura dos novos parâmetros por evento**

Change Notification

```
HANDLE FindFirstChangeNotification(
LPCTSTR lpPathName,                 // Ponteiro para nome do diretório a ser examinado
BOOL bWatchSubtree,                 //  Flag para monitoração de diretório ou árvore de diretório
DWORD dwNotifyFilter                // Condição de filtro para esperar
);
```

lpPathName	Ponteiro para um string terminado em '\0' que especifica o caminho do diretório a ser monitorado.
bWatchSubtree	Especifica se a função irá monitorar o diretório ou a árvore de diretório. TRUE: monitora árvore de diretório FALSE: monitora apenas o diretório
dwNotifyFilter	Condição a ser aguardada: FILE_NOTIFY_CHANGE_FILE_NAME: modificação do nome do arquivo: renomear, criar, ou apagar arquivo. FILE_NOTIFY_CHANGE_DIR_NAME: modificação do nome do diretório: renomear, criar, ou apagar diretório. FILE_NOTIFY_CHANGE_ATTRIBUTES: modificação do atributo no diretório ou árvore de diretórios examinada. FILE_NOTIFY_CHANGE_SIZE: modificação do tamanho de arquivo na árvore ou diretório. FILE_NOTIFY_CHANGE_LAST_WRITE modificação da data da última escrita no arquivo. FILE_NOTIFY_CHANGE_SECURITY: modificação do descritor de segurança no diretório ou árvore de diretórios.

Retorno:

Status	Interpretação
Handle para objeto	Sucesso
INVALID_HANDLE_VALUE	Falha.Chame *GetLastError()*

Change Notification

```
HANDLE FindNextChangeNotification(
HANDLE hChangeHandle          // handle para a change notification a ser sinalizada. Este handle
                              // deve ter sido retornado pela função FindFirstChangeNotification.
);
```

```
HANDLE FindCloseChangeNotification(
HANDLE hChangeHandle          // handle para a change notification a ser fechada.
);
```

Retorno:

Status	Interpretação
<> 0	Sucesso
0	Falha. Chame *GetLastError()* para obter mais informações.

45

Change Notification - Exemplo

```
Thread escritora criada com Id= ffc972ff
Alarmes da planta
HH:MM:SS        Código de parada
07:10:01                41
07:10:02                67
07:10:03                34
07:10:04                0
07:10:05                69
07:10:06                24
07:10:07                78
```

Saída do Programa de Teste

46

Change Notification - Exemplo

```
#define WIN32_LEAN_AND_MEAN
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <process.h>            // _beginthreadex() e _endthreadex()
#include <conio.h>9
#define _CHECKERROR1            // Ativa função CheckForError
#include "CheckForError.h"

// Casting para terceiro e sexto parâmetros da função _beginthreadex
typedef unsigned (WINAPI *CAST_FUNCTION)(LPVOID);
typedef unsigned *CAST_LPDWORD;

// Protótipos de funções
DWORD WINAPI WriterFunc(LPVOID);

typedef struct {
    int iHora;
    int iMinuto;
    int iSegundo;
    int iCodigoEvento;
} Alarme;
```

47

Change Notification - Exemplo

```
int main()
{
    HANDLE hFile;               // Handle para arquivo
    int Arquivo=0;              // Indice do arquivo a ser lido
    HANDLE hThread;
    DWORD dwThreadId;
    HANDLE hcn;
    DWORD ret;
    BOOL Status;
    Alarme MyBuffer;
    DWORD dwBytesLidos;         // Número de bytes lidos
    long lDistanceToMoveHigh=0L;
    long lFilePos = 0L;

    hThread= (HANDLE) _beginthreadex(       // Cria thread para gerar registros no arquivo
            NULL, 0,  (CAST_FUNCTION)WriterFunc, (LPVOID)0, 0, (CAST_LPDWORD)&dwThreadId );
    CheckForError(hThread);
    if (hThread)  printf("Thread escritora criada com Id= %0x \n", dwThreadId);

    hcn = FindFirstChangeNotification(
            "c:\\Livro\\Programas\\Arquivos",
            FALSE,                                  // Monitorar apenas o diretório corrente
            FILE_NOTIFY_CHANGE_LAST_WRITE);         // Filtros
    CheckForError(hcn);
```

48

Change Notification - Exemplo

```
printf("\n Alarmes da planta");
printf("\n HH:MM:SS   Codigo de parada\n");
for ( ; ; ) {
      ret = WaitForSingleObject(hcn, INFINITE);
      CheckForError(ret == WAIT_OBJECT_0);
      hFile = CreateFile("c:\\Livro\\Programas\\Arquivos\\MyAlarms.arq",
            GENERIC_READ,
            FILE_SHARE_READ|FILE_SHARE_WRITE,          // Abre para leitura e escrita
            NULL,                                      // Atributos de segurança
            OPEN_EXISTING,                             // Abre novo arquivo em qualquer situação
            FILE_ATTRIBUTE_NORMAL,                      // Acesso síncrono
            NULL);                                     // Template para atributos e flags
      CheckForError(hFile != INVALID_HANDLE_VALUE);
      SetFilePointer(hFile, iFilePos, &lDistanceToMoveHigh, FILE_BEGIN);
      lFilePos+=sizeof(Alarme);
      Status = ReadFile(hFile, &MyBuffer, sizeof(Alarme), &dwBytesLidos, NULL);
      CloseHandle(hFile);
      printf(" %02d:%02d:%02d   %d\n", MyBuffer.iHora, MyBuffer.iMinuto, MyBuffer.iSegundo,
      MyBuffer.iCodigoEvento);
      if (MyBuffer.iCodigoEvento == -1) break;      // Testa se fim de festa
      FindNextChangeNotification(hcn);
} // for
FindCloseChangeNotification(hcn);
CloseHandle(hThread);
_getch();
return EXIT_SUCCESS;
}  // main
```


Change Notification - Exemplo

```
DWORD WINAPI WriterFunc(LPVOID index)
{
      HANDLE hFile;                          // Handle para arquivo
      DWORD dwBytesEscritos;
      Alarme NovoAlarme;
      DWORD dwHora;                          // Hora em segundos
      long lDistanceToMoveHigh=0l;
      const int iRegMax = 10;
      dwHora = 7 * 60 * 60 + 10 *60 + 0;     // Hora inicial = 7:10:00

      for (int nLoop=0; nLoop<iRegMax ; ++nLoop) {// Gera registro
            dwHora++;
            NovoAlarme.iHora      = dwHora / 3600;
            NovoAlarme.iMinuto    = (dwHora % 3600) / 60;
            NovoAlarme.iSegundo = (dwHora % 3600) % 60;
            // Insere flag no último registro gerado
            NovoAlarme.iCodigoEvento = (nLoop==iRegMax-1)? -1:rand()%100;
            hFile = CreateFile("c:\\Livro\\Programas\\Arquivos\\MyAlarms.arq", // Abre arquivo
                  GENERIC_WRITE,
                  FILE_SHARE_READ|FILE_SHARE_WRITE,      //  leitura/ escrita
                  NULL,                                  // Atributos de segurança
                  (nLoop == 0)? CREATE_ALWAYS:OPEN_EXISTING, // Se primeira vez cria senão abre
                  FILE_ATTRIBUTE_NORMAL,          // Acesso síncrono
                  NULL);                          // Template para atributos e flags
            CheckForError(hFile != INVALID_HANDLE_VALUE);
```

Change Notification - Exemplo

```
            // Escreve no final do arquivo: Append
            SetFilePointer(hFile, 0L, &lDistanceToMoveHigh, FILE_END);
            WriteFile(hFile, &NovoAlarme, sizeof(Alarme), &dwBytesEscritos, NULL);
            CloseHandle(hFile);
            Sleep(1000);            // gera um registro por segundo
        } // for
        _endthreadex((DWORD) index);
        return(0);
} // WriterFunc
```


Muito Obrigado

Perguntas?

Constantino Seixas Filho

constantino.seixas@task.com.br