

Structured Exception Handling **(SEH)**

- Jeffrey Richter, Christophe Nasarre, **Windows via C/C++**, Fifth Edition, Microsoft Press, 2008 [**cap. 23, 24, 25**]
- Microsoft, **Microsoft Developer's Network** (MSDN)

Structured Exception Handling (SEH)

- Permite a escrita de código com a atenção centrada na solução sem preocupações de testar erros
- Se algo correr mal durante a execução o sistema determina o problema e notifica o programa
- Separar o código da lógica da aplicação do código de tratamento de erros
- SEH é específico do sistema operativo Windows
- SEH é uma facilidade do sistema operativo e disponível em qualquer linguagem de programação
- SEH é composto por **duas capacidades** :

- **Tratamento de terminação (*Termination handling*)**

```
__try {  
    // Bloco protegido (Guarded body)  
}  
__finally {  
    // Handler de terminação (Termination Handler)  
}
```

- **Tratamento de exceção (*Exception Handling*)**

```
__try {  
    // Bloco protegido (Guarded body)  
}  
__except( <exception filter expression> ) {  
    // Handler de exceção (Exception Handler)  
}
```

Nota: Não confunda o SEH com o tratamento de exceções da linguagem C++ (try, catch e throw).

SEH: Tratamento de Terminação (*Termination handling*)

```
__try {  
    // Bloco protegido (Guarded body)  
}  
__finally {  
    // Handler de terminação (Termination Handler)  
}
```

- O sistema operativo e o compilador garantem que o bloco de código indicado no `__finally` (*handler* de terminação) é executado independentemente da forma como terminou o código do `try` (bloco protegido)
- Assim, o *handler* de terminação pode ser executado em cada uma das situações:
 - Execução sem problemas do bloco protegido, seguindo-lhe a execução do *handler* de terminação.
 - Saída prematura do bloco protegido (através *goto*, *longjump*, *continue*, *break*, *return*, etc.) forçando a execução do *handler* de terminação.
 - Devido a um *unwind* global após a ocorrência de uma exceção durante a execução do bloco protegido.
- O *handler* de terminação executa-se em consequência de uma destas três situações. Existe a função `AbnormalTermination()` que permite determinar se o *handler* de terminação está a ser executado por uma saída prematura do bloco protegido ou não. Esta função apenas pode ser usada no *handler* de terminação.

SEH: Exemplo Tratamento de Terminação

Exemplifica-se a seguir uma possível utilização dos blocos de tratamento de terminação em que, independentemente do que aconteça no bloco de código protegido (`return`, `break`, `goto`, ou exceção com, por exemplo, acesso inválido a uma posição de memória) o código definido no bloco `finally` é sempre executado (note que se for utilizada a função `ExitProcess` tal não acontece).

```
void main ()
{
    HANDLE hsem = CreateSemaphore(NULL, 1, 1, "SemExemploSEH");

    __try {
        WaitForSingleObject(hsem, INFINITE);
        //...
        int *p = NULL;
        *p = 9;
        //...
        return p;
    }
    __finally {
        ReleaseSemaphore(hsem, 1, NULL);
    }
}
```

SEH: Exemplo de ficheiro mapeado

```
BOOL printFile(const char *fileName) {
    HANDLE hFile = CreateFile(
        fileName,           // file name
        GENERIC_READ,       // access mode
        0,                  // share
        NULL,               // Security
        OPEN_EXISTING,      // how to create
        FILE_ATTRIBUTE_NORMAL, // file attributes
        NULL                // handle to template file
    );
    if (hFile == INVALID_HANDLE_VALUE) {
        ReportErrorSystem("Erro na abertura do ficheiro <%s>:", fileName);
        return FALSE;
    }

    HANDLE fileMapHandle = CreateFileMapping(
        hFile,              // handle to file
        NULL,               // security
        PAGE_READONLY,      // protection
        0,                  // high-order DWORD of size
        0,                  // low-order DWORD of size
        NULL                // object name
    );
    if (fileMapHandle == NULL) {
        ReportErrorSystem("Erro na abertura do FileMap");
        CloseHandle(hFile);
        return FALSE;
    }
}
```

```
char *lpMapAddress = (char *)MapViewOfFile(
    fileMapHandle,          // Handle to mapping object
    FILE_MAP_READ,         // Read/write permission
    0,                     // dwFileOffsetHigh
    0,                     // dwFileOffsetLow
    0);                    // dwNumberOfBytesToMap
if (lpMapAddress == NULL) {
    ReportErrorSystem("Erro no MapViewOfFile.");
    CloseHandle(hFile); CloseHandle(fileMapHandle);
    return FALSE;
}
DWORD fileSize = GetFileSize(hFile, NULL);
if (fileSize == INVALID_FILE_SIZE) {
    ReportErrorSystem("Erro no GetFileSize.");
    UnmapViewOfFile(lpMapAddress);
    CloseHandle(hFile); CloseHandle(fileMapHandle);
    return FALSE;
}
for (DWORD i = 0; i < fileSize; ++i) putchar(lpMapAddress[i]);
// libertar recursos
UnmapViewOfFile(lpMapAddress);
CloseHandle(fileMapHandle);
CloseHandle(hFile);
return TRUE;
} // end printFile
```

SEH: Utilizando o bloco `__try __finally` (I)

```
BOOL printFileSEH1(const char *fileName) {  
    // Iniciar todas as variáveis indicando a situação de erro  
    HANDLE hFile          = INVALID_HANDLE_VALUE;  
    HANDLE fileMapHandle = NULL;  
    char *lpMapAddress    = NULL;  
  
    __try {  
        hFile = CreateFile(  
            fileName,                // file name  
            GENERIC_READ,            // access mode  
            0,                       // share  
            NULL,                   // Security  
            OPEN_EXISTING,           // how to create  
            FILE_ATTRIBUTE_NORMAL,    // file attributes  
            NULL                     // handle to template file  
        );  
        if (hFile == INVALID_HANDLE_VALUE) {  
            ReportErrorSystem("Erro na abertura do ficheiro");  
            return FALSE;  
        }  
        fileMapHandle = CreateFileMapping(  
            hFile,                  // handle to file  
            NULL,                  // security  
            PAGE_READONLY,         // protection  
            0,                    // high-order DWORD of size  
            0,                    // low-order DWORD of size  
            NULL                    // object name  
        );  
    }
```

```
    if (fileMapHandle == NULL) {  
        ReportErrorSystem("Erro na abertura do FileMap");  
        return FALSE;  
    }  
  
    lpMapAddress = (char *)MapViewOfFile(  
        fileMapHandle,              // Handle to mapping object  
        FILE_MAP_READ,             // Read/write permission  
        0,                         // dwFileOffsetHigh  
        0,                         // dwFileOffsetLow  
        0                          // dwNumberOfBytesToMap  
    );  
    if (lpMapAddress == NULL) {  
        ReportErrorSystem("Erro no MapViewOfFile.");  
        return FALSE;  
    }  
  
    DWORD fileSize = GetFileSize(hFile, NULL);  
    if (fileSize == INVALID_FILE_SIZE) {  
        ReportErrorSystem("Erro no GetFileSize.");  
        return FALSE;  
    }  
  
    for (DWORD i = 0; i < fileSize; ++i)  
        putchar(lpMapAddress[i]);  
  
    return TRUE;  
} // end __try
```

SEH: Utilizando o bloco `__try __finally` (II)

```
__finally {  
    // libertar recursos  
    if ( lpMapAddress != NULL )  
        if ( !UnmapViewOfFile(lpMapAddress) )  
            ReportErrorSystem("Erro no UnmapViewOfFile.");  
  
    if ( fileMapHandle != NULL )  
        if ( !CloseHandle(fileMapHandle) )  
            ReportErrorSystem("Erro no CloseHandle(fileMapHandle).");  
  
    if ( hFile != INVALID_HANDLE_VALUE )  
        if ( !CloseHandle(hFile) )  
            ReportErrorSystem("Erro no CloseHandle(hFile).");  
} // end __finally  
  
} // end printFileSEH1
```

SEH: Utilizando o bloco __try __finally e __leave (I)

```
BOOL printFileSEH2(const char *fileName) {
    // Iniciar todas as variáveis indicando a situação de erro
    HANDLE hFile          = INVALID_HANDLE_VALUE;
    HANDLE fileMapHandle = NULL;
    char    *lpMapAddress = NULL;
    BOOL    retValue      = FALSE;
    __try {
        hFile = CreateFile(
            fileName,           // file name
            GENERIC_READ,      // access mode
            0,                  // share mode
            NULL,               // Security
            OPEN_EXISTING,     // how to create
            FILE_ATTRIBUTE_NORMAL, // file attributes
            NULL                // handle to template file
        );
        if (hFile == INVALID_HANDLE_VALUE) {
            ReportErrorSystem("Erro na abertura do ficheiro");
            __leave;
        }
        fileMapHandle = CreateFileMapping(
            hFile,              // handle to file
            NULL,               // security
            PAGE_READONLY,     // protection
            0,                  // high-order
            0,                  // low-order
            NULL                // object name
        );
    }
```

```
    if (fileMapHandle == NULL) {
        ReportErrorSystem("Erro na abertura do FileMap");
        __leave;
    }
    lpMapAddress = (char *)MapViewOfFile(
        fileMapHandle,          // Handle to mapping object
        FILE_MAP_READ,         // Read/write permission
        0,                      // dwFileOffsetHigh
        0,                      // dwFileOffsetLow
        0                       // dwNumberOfBytesToMap
    );
    if (lpMapAddress == NULL) {
        ReportErrorSystem("Erro no MapViewOfFile.");
        __leave;
    }

    DWORD fileSize = GetFileSize(hFile, NULL);
    if (fileSize == INVALID_FILE_SIZE) {
        ReportErrorSystem("Erro no GetFileSize.");
        __leave;
    }

    for (DWORD i = 0; i < fileSize; ++i)
        putchar(lpMapAddress[i]);

    retValue = TRUE;
} // end __try
```


SEH: Utilizando o bloco `__try __finally e __leave` (II)

```
__finally {  
  
    // libertar recursos  
    if ( lpMapAddress != NULL )  
        if ( !UnmapViewOfFile(lpMapAddress) )  
            ReportErrorSystem("Erro no UnmapViewOfFile.");  
  
    if ( fileMapHandle != NULL )  
        if ( !CloseHandle(fileMapHandle) )  
            ReportErrorSystem("Erro no CloseHandle(fileMapHandle).");  
  
    if ( hFile != INVALID_HANDLE_VALUE )  
        if ( !CloseHandle(hFile) )  
            ReportErrorSystem("Erro no CloseHandle(hFile).");  
  
} // end __finally  
  
return retValue;  
  
} // end printFileSEH2
```

SEH: Tratamento de Excepção (*Exception Handling*)

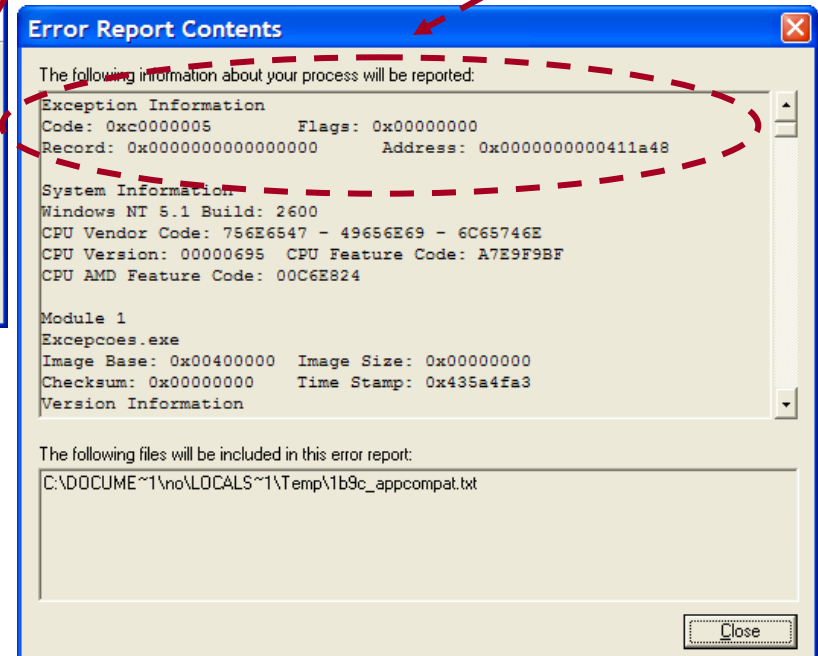
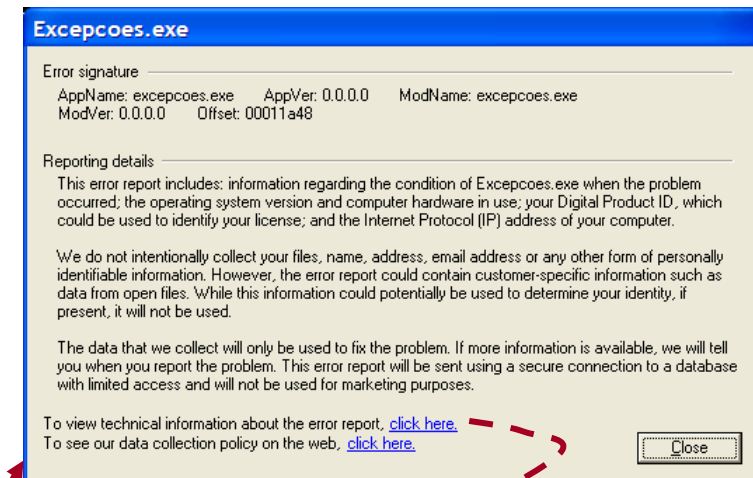
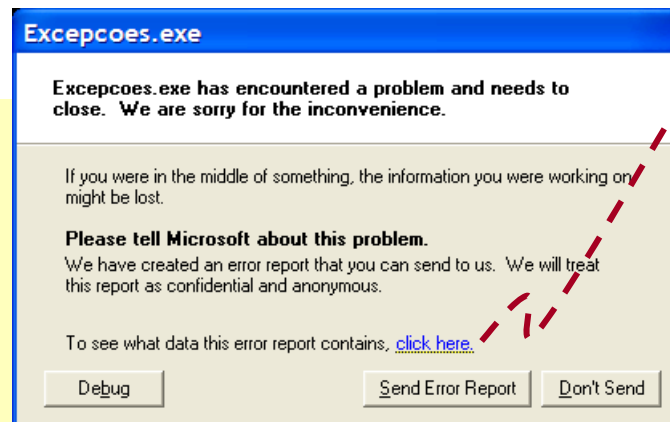
```
__try {  
    // Bloco protegido (Guarded body)  
}  
__except( <exception filter expression> ) {  
    // Handler de excepção(Exception Handler)  
}
```

- Os *handlers* de excepção e os filtros são executados directamente pelo sistema operativo no contexto da tarefa onde esta ocorreu
- O filtro de excepção é uma expressão da qual deve resultar um dos seguintes três valores:
 - **EXCEPTION_EXECUTE_HANDLER**
 - **EXCEPTION_CONTINUE_SEARCH**
 - **EXCEPTION_CONTINUE_EXECUTION**
- Estes valores alteram a execução da tarefa
- A expressão do filtro da excepção pode ser uma função que devolva um dos três valores

SEH: Exemplo Tratamento de Excepção (I)

Se considerarmos este programa (excepcoes.cpp) que acede à posição de memória 0x00000000 verificamos que o sistema detecta este acesso indevido á memória e, nos sistemas Windows XP (por omissão), aparece-nos uma janela que permite obter informações sobre a excepção e eventualmente enviar um relatório para um servidor do fabricante

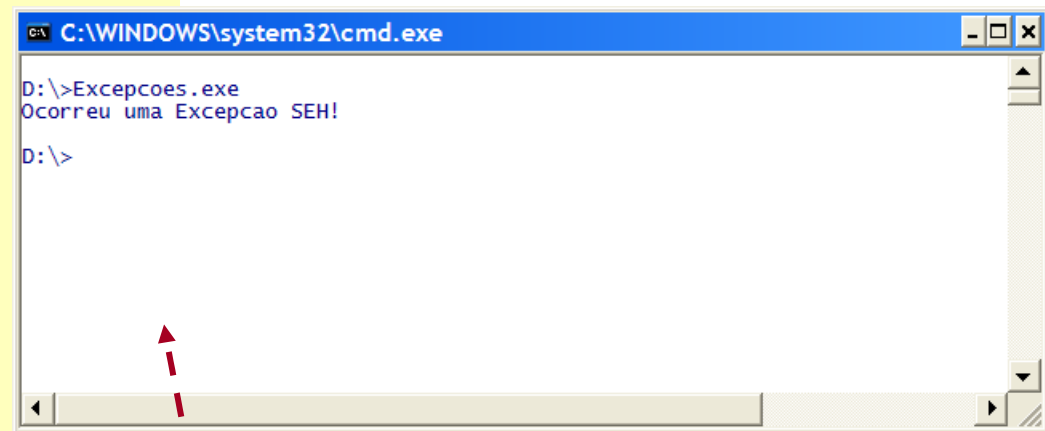
```
void main ()  
{  
    // ...  
  
    int *p = NULL;  
    *p = 9;  
    // ...  
}
```



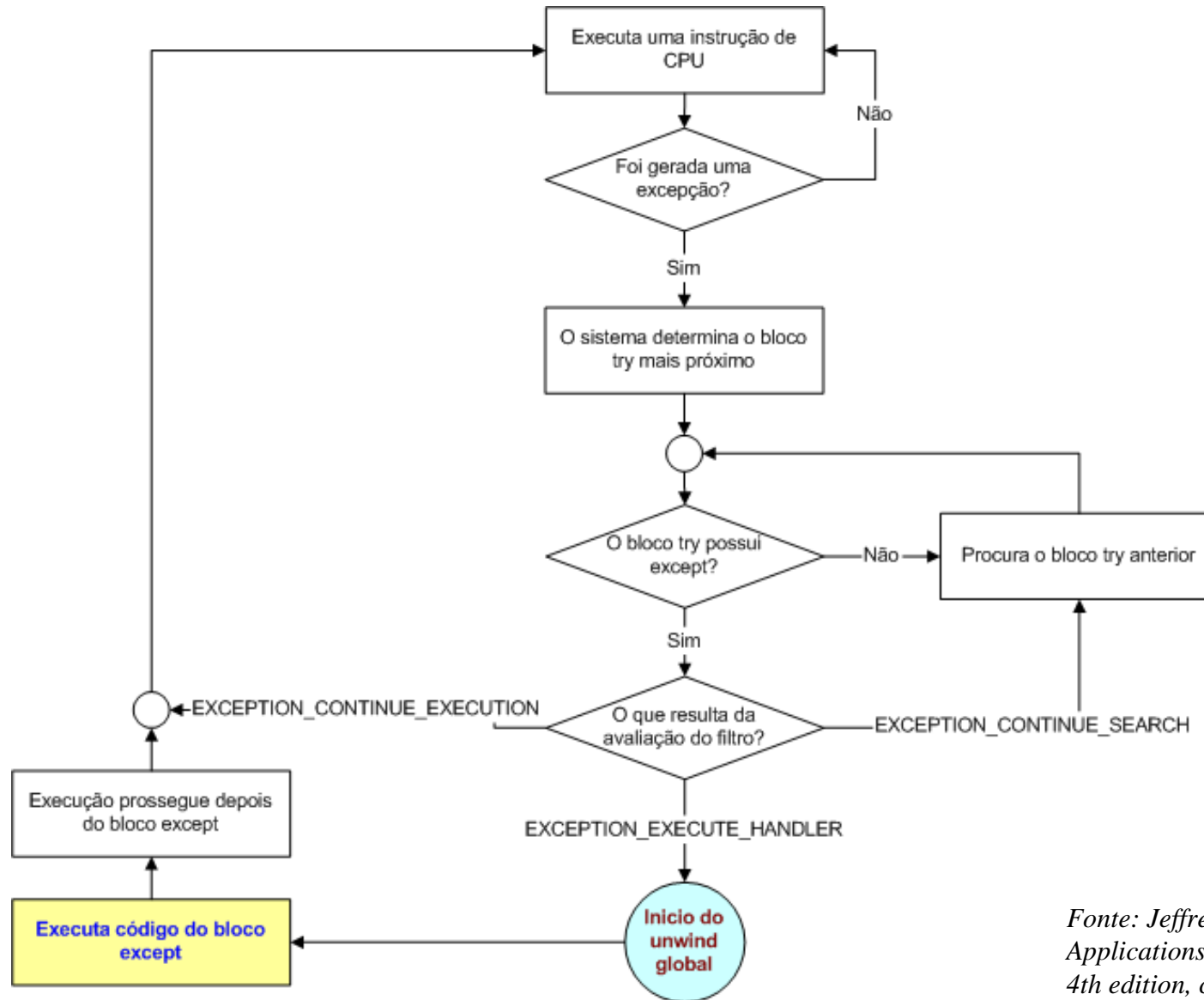
SEH: Exemplo Tratamento de Exceção (II)

Podemos realizar uma nova versão para no caso da ocorrência de uma exceção SEH tratá-la de modo adequado.

```
void main ()  
{  
    __try {  
  
        // ...  
  
        int *p = NULL;  
        *p = 9;  
  
        // ...  
    }  
    __except ( EXCEPTION_EXECUTE_HANDLER ) {  
        printf("Ocorreu uma Excepcao SEH!\n");  
    }  
}
```

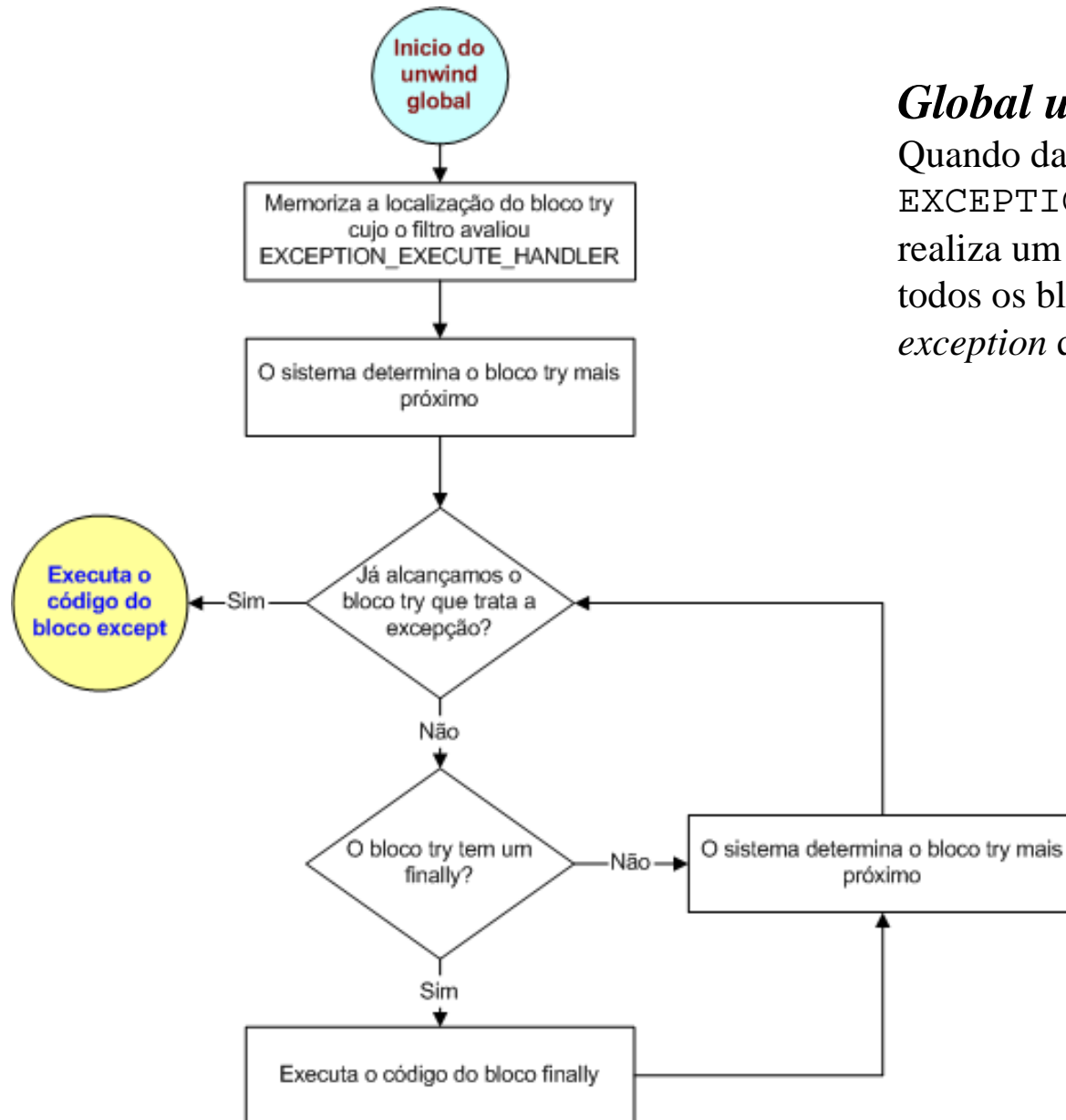


SEH: Como é processada uma exceção pelo sistema



Fonte: Jeffrey Richter, *Programming Applications for Microsoft Windows*, 4th edition, cap. 23, 24 e 25

SEH: Como é realizado o *unwind* global pelo sistema



Global unwind

Quando da avaliação do filtro resulta o valor `EXCEPTION_EXECUTE_HANDLER`, o sistema realiza um *unwind* global. Este *unwind* provoca que todos os blocos *finally*, existentes dentro do bloco *try exception* que vai tratar a exceção, sejam executados

Fonte: Jeffrey Richter, *Programming Applications for Microsoft Windows*, 4th edition, cap. 23, 24 e 25

SEH: Função `GetExceptionCode()`

```
DWORD GetExceptionCode(void);
```

- Os filtros de exceção podem analisar a situação que levou à exceção antes de decidir o valor a devolver:

EXCEPTION_EXECUTE_HANDLER ou

EXCEPTION_CONTINUE_SEARCH ou

EXCEPTION_CONTINUE_EXECUTION

- A função **GetExceptionCode** devolve um valor identificando o tipo de exceção que ocorreu (definidos no ficheiro `WinBase.h`)
- Esta função apenas pode ser chamada na expressão do filtro de exceção (entre os parêntesis a seguir ao `__except`) ou dentro do *handler* de exceção.

SEH: Tipo de exceções devolvido pela função `GetExceptionCode`

Relacionadas com a memória

EXCEPTION_ACCESS_VIOLATION
EXCEPTION_DATATYPE_MISALIGNMENT
EXCEPTION_ARRAY_BOUNDS_EXCEEDED
EXCEPTION_IN_PAGE_ERROR
EXCEPTION_GUARD_PAGE
EXCEPTION_STACK_OVERFLOW
EXCEPTION_ILLEGAL_INSTRUCTION
EXCEPTION_PRIV_INSTRUCTION

Relacionadas as exceções

EXCEPTION_INVALID_DISPOSITION
EXCEPTION_NONCONTINUABLE_EXCEPTION

Relacionadas com *debugging*

EXCEPTION_BREAKPOINT
EXCEPTION_SINGLE_STEP
EXCEPTION_INVALID_HANDLE

Relacionadas com operações de inteiros

EXCEPTION_INT_DIVIDE_BY_ZERO
EXCEPTION_INT_OVERFLOW

Relacionadas com operações de reais

EXCEPTION_FLT_DENORMAL_OPERAND
EXCEPTION_FLT_DIVIDE_BY_ZERO
EXCEPTION_FLT_INEXACT_RESULT
EXCEPTION_FLT_INVALID_OPERATION
EXCEPTION_FLT_OVERFLOW
EXCEPTION_FLT_STACK_CHECK
EXCEPTION_FLT_UNDERFLOW

NOTA: consultar MSDN para lista de tipos de exceções que podem ser devolvidas pela função `GetExceptionCode`

SEH: Regra de Composição dos Códigos de Erro

Bits	31-30	29	28	27-16	15-0
Contents	Severity	Microsoft/customer	Reserved	Facility code	Exception code
Meaning	0=Success 1=Informational 2=Warning 3=Error	0=Microsoft-defined code 1=customer-defined code	Must be 0	Microsoft-defined (see table below)	Microsoft/customer-defined

Facility Code	Value	Facility Code	Value
FACILITY_NULL	0	FACILITY_CONTROL	10
FACILITY_RPC	1	FACILITY_CERT	11
FACILITY_DISPATCH	2	FACILITY_INTERNET	12
FACILITY_STORAGE	3	FACILITY_MEDIASERVER	13
FACILITY_ITF	4	FACILITY_MSMQ	14
FACILITY_WIN32	7	FACILITY_SETUPAPI	15
FACILITY_WINDOWS	8	FACILITY_SCARD	16
FACILITY_SECURITY	9	FACILITY_COMPLUS	17

Regra de composição dos códigos de erro definidos pela Microsoft (ficheiro winerror.h)

Fonte: Jeffrey Richter, Programming Applications for Microsoft Windows, 4th edition, cap. 23, 24, 25

SEH: Exemplo utilizando a função `GetExceptionCode()`

Podemos reformular o exemplo anterior de forma a tratarmos apenas a exceção de acesso inválido á memória (`EXCEPTION_ACCESS_VIOLATION`) todas as outras são enviadas para os outros blocos `__try` `__except` existentes

```
void main ()
{
    __try {

        // ...

        int *p = NULL;
        *p = 9;

        // ...

    }
    __except ( GetExceptionCode() == EXCEPTION_ACCESS_VIOLATION ?
              EXCEPTION_EXECUTE_HANDLER :
              EXCEPTION_CONTINUE_SEARCH )
    {
        printf("Ocorreu uma Excepcao EXCEPTION_ACCESS_VIOLATION!\n");
    }
}
```

SEH: Função `GetExceptionInformation()`

```
LPEXCEPTION_POINTERS GetExceptionInformation (VOID);
```

- A função **GetExceptionInformation()** devolve a descrição da exceção e a informação sobre o estado da máquina que existia no contexto da tarefa onde ocorreu a exceção
- Devolve um apontador para um estrutura **EXCEPTION_POINTERS**

```
typedef struct _EXCEPTION_POINTERS {  
    PEXCEPTION_RECORD ExceptionRecord;  
    PCONTEXT ContextRecord;  
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

- Esta função só pode ser chamada dentro da expressão do filtro de um *handler* de exceção porque **EXCEPTION_POINTERS**, **EXCEPTION_RECORD** e **CONTEXT** só são válidas no processamento do filtro de exceção

SEH: Estrutura EXCEPTION_POINTERS

A estrutura **EXCEPTION_POINTERS** contém apontadores para outras duas estruturas:

- **EXCEPTION_RECORD**

Estrutura contendo a descrição da exceção:

```
typedef struct _EXCEPTION_RECORD {  
    DWORD ExceptionCode;  
    DWORD ExceptionFlags;  
    struct _EXCEPTION_RECORD* ExceptionRecord;  
    PVOID ExceptionAddress;  
    DWORD NumberParameters;  
    ULONG_PTR ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];  
}EXCEPTION_RECORD, *PEXCEPTION_RECORD;
```

- **CONTEXT**

Estrutura contendo a informação do estado da máquina, isto é informação específica dos registos do processador. O ficheiro `WinNT.h` contém a definição desta estrutura para cada arquitectura de processador.

SEH: Exemplo utilizando a função `GetExceptionInformation()` (I)

Podemos chamar uma função na expressão de filtro do `__except` que receba por parâmetro um apontador para a estrutura **EXCEPTION_POINTERS** obtido da chamada à função `GetExceptionInformation()`

```
void main ()
{
    __try {

        // ...

        int *p = NULL;
        *p = 9;

        // ...
    }
    __except ( mySEHFilter(GetExceptionInformation()) )
    {
        printf("Ocorreu uma Excepcao SEH!\n");
    }
}
```

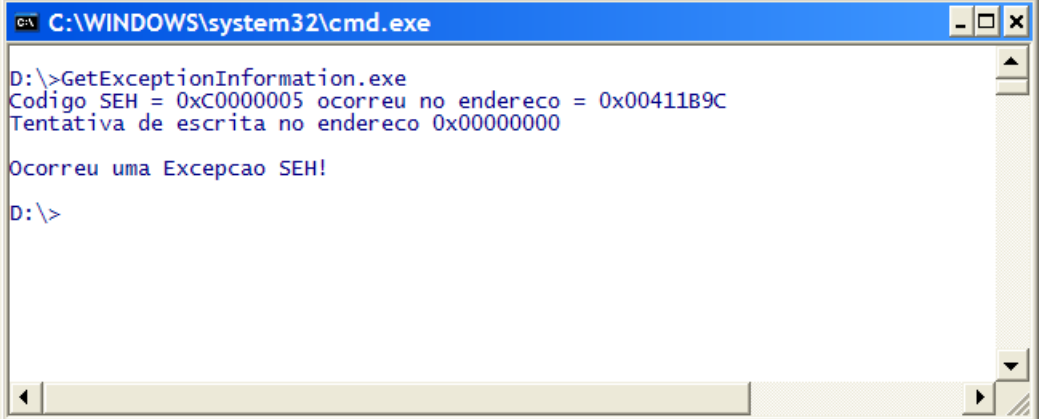
SEH: Exemplo utilizando a função `GetExceptionInformation()` (II)

```
LONG mySEHFilter ( LPEXCEPTION_POINTERS pExceptionPointers )
{
    PEXCEPTION_RECORD exceptionRecord = pExceptionPointers->ExceptionRecord;

    printf("Codigo SEH = 0x%X ocorreu no endereco = 0x%p\n",
           exceptionRecord->ExceptionCode, exceptionRecord->ExceptionAddress);

    switch( exceptionRecord->ExceptionCode )
    {
        case EXCEPTION_ACCESS_VIOLATION:
            printf("Tentativa de %s no endereco 0x%p\n",
                   exceptionRecord->ExceptionInformation[0] ? "escrita": "leitura",
                   exceptionRecord->ExceptionInformation[1]);
            break;
    }
    printf("\n");
    return EXCEPTION_EXECUTE_HANDLER;
}
```

Note: que esta função de filtro está a imprimir a informação sobre qualquer excepção SEH e devolve sempre `EXCEPTION_EXECUTE_HANDLER`.



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The output of the program is as follows:

```
D:\>GetExceptionInformation.exe
Codigo SEH = 0xC0000005 ocorreu no endereco = 0x00411B9C
Tentativa de escrita no endereco 0x00000000

Ocorreu uma Excepcao SEH!

D:\>
```

SEH: Exemplo utilizando a função `GetExceptionInformation()` (III)

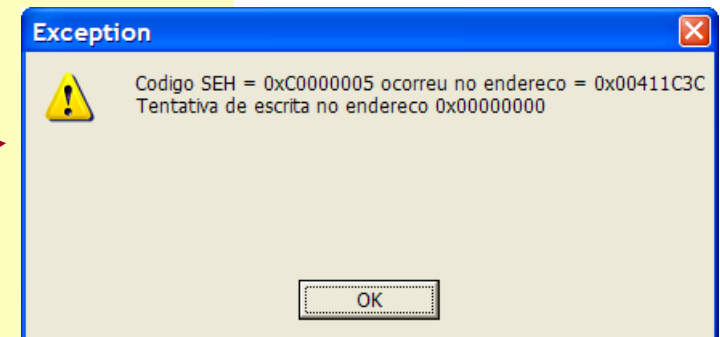
```
LONG mySEHFilter ( LPEXCEPTION_POINTERS pExceptionPointers )
{
    PEXCEPTION_RECORD exceptionRecord = pExceptionPointers->ExceptionRecord;
    char buf[1024];
    int nChwr;
    nChwr = sprintf(buf, "Codigo SEH = 0x%X ocorreu no endereco = 0x%p\n",
                    exceptionRecord->ExceptionCode, exceptionRecord->ExceptionAddress);

    switch( exceptionRecord->ExceptionCode )
    {
        case EXCEPTION_ACCESS_VIOLATION:
            nChwr += sprintf(buf+nChwr, "Tentativa de %s no endereco 0x%p\n",
                            exceptionRecord->ExceptionInformation[0] ? "escrita": "leitura",
                            exceptionRecord->ExceptionInformation[1]);

            break;
    }
    sprintf(buf+nChwr, "\n\n\n\n\n\n\n");

    MessageBox(NULL, buf, "Exception", MB_OK | MB_ICONEXCLAMATION);

    return EXCEPTION_EXECUTE_HANDLER;
}
```



SEH: E quando acontece uma exceção ao nível do Sistema Operativo...

Durante a execução de código em modo de sistema (*kernel mode*) se ocorrer uma exceção não tratada significa um erro grave e nesta situação não é seguro o sistema continuar em execução. Neste caso o sistema apresenta uma ecrã azul (O conhecido *Blue Screen of Death*) onde são apresentados quais os *device drivers* carregados e qual o módulo que contém o código que originou a exceção.

```
*** STOP: 0x00000019 (0x00000000,0xC00E0FF0,0xFFFFFD4,0xC0000000)
BAD_POOL_HEADER

CPUID: GenuineIntel 5.2.0 irql:1f SYSVER 0xf0000565

Dll Base DateStamp - Name
80100000 3202c07e - ntoskrnl.exe
80001000 31ed06b4 - atapi.sys
802c6000 31ed06bf - aic78xx.sys
802d1000 31ec6c7a - CLASS2.SYS
fc698000 31ec6c7d - Floppy.SYS
fc90a000 31ec6df7 - Fs_Rec.SYS
fc864000 31ed868b - KSecDD.SYS
fc648000 31ec6c90 - i8042prt.sys
fc874000 31ec6c94 - kbdclass.sys
fcffa000 31ec6c62 - mma_mll.sys
fc708000 31ec6ccb - Msfs.SYS
fcfb0000 31eed262 - NDIS.SYS
fcfa4000 31f91a51 - mma.dll
fc80c000 31ec6e6c - TDI.SYS
fcacf000 31f130a7 - tcpip.sys
fc550000 31601a30 - el59x.sys
fc718000 31ec6e7a - netbios.sys
fc870000 31ec6c9b - Parallel.SYS
fc5b0000 31ec6cb1 - Serial.SYS
fcab0000 31f7a1ba - mup.sys

Dll Base DateStamp - Name
80010000 31ee6c52 - hal.dll
80006000 31ec6c74 - SCSIPTORT.SYS
802c4000 31ed237c - Disk.sys
8037c000 31ee48a7 - Ntfs.sys
fc6a8000 31ec6ca1 - Cdrom.SYS
fc9c9000 31ec6c99 - Null.SYS
fc9ca000 31ec6c78 - Beep.SYS
fc86c000 31ec6c97 - mouclass.sys
fc6f0000 31f50722 - VIDEOPORT.SYS
fc890000 31ec6c6d - vga.sys
fc4b0000 31ec6cc7 - Npfs.SYS
a0000000 31f954f7 - win32k.sys
fec31000 31eedd07 - Fastfat.SYS
fcdf0000 31ed0754 - nbf.sys
fcab3000 31f50a65 - netbt.sys
fc560000 31f8f864 - afd.sys
fc858000 31ec6c9b - Parport.sys
fc954000 31ec6c9d - ParWdm.SYS
fcad4000 31f5003b - nls.sys
fc9da000 32031abe - srv.sys

Address dword dump Build [13811] - Name
fec32d04 80143e00 80143e00 80144000 ffd5f000 00070b02 - KSecDD.SYS
801471c8 80144000 80144000 ffd5f000 00000001 - ntoskrnl.exe
801471dc 80122000 f0003fe0 f030eeee e133c4b4 e133cd40 - ntoskrnl.exe
80147304 803023f0 0000023c 00000034 00000000 00000000 - ntoskrnl.exe

Restart and set the recovery options in the system control panel
or the /CRASHDEBUG system start option.
```

```
*** STOP: 0x0000000A (0x00000000,0x00000002,0x00000000,8038c240)
IRQL_NOT_LESS_OR_EQUAL*** Address 8038c240 has base at 8038c000 - Ntfs.SYS

CPUID: GenuineIntel 6.3.3 irql:1f SYSVER 0xf0000565

Dll Base DateStamp - Name
80100000 336546bf - ntoskrnl.exe
80000100 334d3a53 - atapi.sys
802aa000 33013a6b - epst.mpd
802b9000 336015af - CLASS2.SYS
802bd000 33d844be - Siwvid.sys
f9318000 31ec6c8d - Floppy.SYS
f9468000 31ed868b - KSecDD.SYS
f9358000 335bc82a - i8042prt.sys
f947c000 31ec6c94 - kbdclass.sys
f9370000 33248011 - VIDEOPORT.SYS
f9490000 31ec6c6d - vga.sys
f90f0000 332480d0 - Npfs.SYS
a0000000 335157ac - win32k.sys
fe0c9000 335bd30e - Fastfat.SYS
fe108000 31ec6c9b - Parallel.SYS
f9050000 332480ab - Serial.SYS

Dll Base DateStamp - Name
80010000 33247f88 - hal.dll
80007000 33248043 - SCSIPTORT.SYS
802b5000 336016a2 - Disk.sys
8038c000 3356d637 - Ntfs.sys
803e4000 33d84553 - NTice.sys
f95c9000 31ec6c99 - Null.SYS
f95ca000 335e60cf - Beep.SYS
f9474000 3324806f - mouclass.sys
f95cb000 3373c39d - ctrl2cap.SYS
fe9d7000 3370e7b9 - ati.sys
f93b0000 332480dd - Msfs.SYS
fe957000 3356da41 - NDIS.SYS
fe914000 334eal44 - ati.dll
fe110000 31ec7c9b - Parport.SYS
f95b4000 31ec6c9d - ParWdm.SYS

Address dword dump Build [1314] - Name
801afc24 80149905 80149905 ff8e6b8c 80129c2c ff8e6b94 8025c000 - Ntfs.SYS
801afc2c 80129c2c 80129c2c ff8e6b94 00000000 ff8e6b94 80100000 - ntoskrnl.exe
801afc34 801240f2 80124f02 ff8e6df4 ff8e6f60 ff8e6c58 80100000 - ntoskrnl.exe
801afc54 80124f16 80124f16 ff8e6f60 ff8e6c3c 8015ac7e 80100000 - ntoskrnl.exe
801afc64 8015ac7e 8015ac7e ff8e6df4 ff8e6f60 ff8e6c58 80100000 - ntoskrnl.exe
801afc70 80129bda 80129bda 00000000 80088000 80106fc0 80100000 - ntoskrnl.exe

Restart and set the recovery options in the system control panel
or the /CRASHDEBUG system start option. If this message reappears,
contact your system administrator or technical support group.
```


SEH: Exceções de *software*

```
void RaiseException(  DWORD           dwExceptionCode,  
                    DWORD           dwExceptionFlags,  
                    DWORD           nNumberOfArguments,  
                    const ULONG_PTR* lpArguments );
```

- É possível gerar exceções por software que são tratadas da mesma maneira que as exceções de hardware
- Para gerar uma exceção de software é utilizada a função **RaiseException**()

SEH: Exceções não tratadas (*unhandled exception*) (I)

- Se o nosso código nunca tratar uma exceção SEH (i.e. todos os filtros de exceção devolvem `EXCEPTION_CONTINUE_SEARCH`) estamos perante uma exceção não tratada (*unhandled exception*)
- O sistema neste caso garante o tratamento destas exceções assegurando-se que o código de todas as tarefas se encontra protegido por um bloco `__try __except`
 - Todas as tarefas do sistema começam a sua execução numa função definida da DLL `Kernel32`
 - Existem duas funções semelhantes:
 - **BaseProcessStart** uma é utilizada para a tarefa primária do processo e
 - **BaseThreadStart** outra para as restantes tarefas
- O sistema disponibiliza uma função de filtro, *UnhandledExceptionFilter*, que é chamada no filtro `__try __except` das funções `BaseProcessStart`, `BaseThreadStart`. No entanto, esta função pode, também, ser chamada pelas aplicações. É esta função que é responsável pela apresentação da janela informando que uma tarefa de um processo provocou uma exceção
- No caso das aplicações C/C++ o *run time* envolve o código das *tarefas* num outro bloco `__try __except` para o tratamento das exceções não tratadas

SEH: Exceções não tratadas (*unhandled exception*) (II)

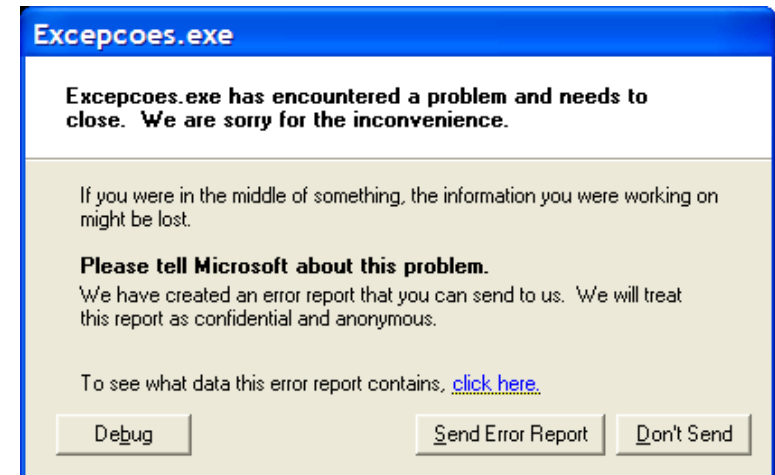
```
VOID BaseProcessStart (PPROCESS_START_ROUTINE pfnStartAddr)
{
    __try {
        ExitThread( (pfnStartAddr)() );
    }
    __except ( UnhandledExceptionFilter(GetExceptionInformation()) ) {
        ExitProcess( GetExceptionCode() );
    }
    // NOTA: A execução do código nunca chega aqui
}
```

```
VOID BaseThreadStart (PTHREAD_START_ROUTINE pfnStartAddr, PVOID pvParam)
{
    __try {
        ExitThread( (pfnStartAddr)(pvParam) );
    }
    __except ( UnhandledExceptionFilter( GetExceptionInformation() ) ) {
        ExitProcess( GetExceptionCode() );
    }
    // NOTA: A execução do código nunca chega aqui
}
```

SEH: Exceções não tratadas (*unhandled exception*) (III)

```
LONG UnhandledExceptionFilter(struct_EXCEPTION_POINTERS* ExceptionInfo );
```

- Esta função é responsável pela apresentação da janela informando que uma tarefa de um processo provocou uma exceção.
- Se for escolhida o botão **Debug** a função tenta carregar o *debug* e associá-lo ao processo
- Se escolhermos uma das outras opções () a função devolve EXCEPTION_EXECUTE_HANDLER levando a que seja realizado um *unwind global* garantindo-se, assim, a execução de todos os blocos finally. De seguida executa-se o handler de exceção da função **BaseProcessStart** ou da função **BaseThreadStart** que chamam a função **ExitProcess** conduzindo à terminação do processo. É por esta razão que as nossas aplicações terminam quando provocam uma exceção.



SEH: Exceções não tratadas (*unhandled exception*) (IV)

```
UINT SetErrorMode( UINT uMode );
```

- Pode-se evitar que a função **UnhandledExceptionFilter** apresente a janela de diálogo chamado a função **SetErrorMode** e passando como parâmetro o valor **NOGPFALTERRORBOX**
- Assim, quando a função **UnhandledExceptionFilter** é chamada verifica que foi pedido para que não tenha lugar a apresentação da janela de diálogo e a função termina devolvendo **EXCEPTION_EXECUTE_HANDLER**
- Note que desta forma **não existe qualquer aviso** da razão pela qual a aplicação terminou

SEH: Exceções não tratadas (*unhandled exception*) (V)

```
LPTOP_LEVEL_EXCEPTION_FILTER  
SetUnhandledExceptionFilter (LPTOP_LEVEL_EXCEPTION_FILTER  
                             lpTopLevelExceptionFilter );
```

- Pode-se utilizar esta função para definir uma função de filtro a ser utilizada quando ocorre uma exceção não tratada
- A função tem um protótipo igual ao da função **UnhandledExceptionFilter** ou seja: LONG UnhandledExceptionFilter (PEXCEPTION_POINTERS pExceptionInfo);
- A função deve terminar devolvendo um dos três valores:
 - **EXCEPTION_EXECUTE_HANDLER**
 - **EXCEPTION_CONTINUE_SEARCH**
 - **EXCEPTION_CONTINUE_EXECUTION**
- Quando acontece uma exceção não tratada a função **UnhandledExceptionFilter** é chamada e vai verificar se foi definida uma função de filtro, através da função **SetUnhandledExceptionFilter**, e em caso afirmativo procede á sua evocação

SEH: Integração com as excepções C++

Integração das SEH com as excepções da linguagem C++

- A função `_set_se_translator` regista, para o *thread* que a chama, uma função que vai ser chamada quando ocorre uma excepção SEH, isto é, avisamos o *run time* C/C++ para chamar a função, indicada na chamada de `_set_se_translator`, sempre que ocorra uma excepção SEH.
- Esta função pode agora gerar uma excepção C++ (*throw <type>*); A classe do objecto excepção pode ser decidida com base no código obtido da excepção SEH que ocorreu.

```
typedef void (*_se_translator_function)(unsigned int, struct _EXCEPTION_POINTERS*);  
  
_se_translator_function _set_se_translator( _se_translator_function se_trans_func );
```

SEH: Integração com as excepções C++ - Exemplo (I)

```
class SystemException {
    EXCEPTION_RECORD m_er;          // CPU independent exception information
    CONTEXT          m_context;     // CPU dependent exception information
public:
    // Chamar esta função por cada thread antes de usar try/catch;
    // faz o registo da função de conversão de Excepções do S.O. (SEH) para um objecto C++;
    static void MapSystemException() { _set_se_translator(TranslateSEH); }

    // Construtor do objecto de excepção só activado por TranslateSEH!
    SystemException(PEXCEPTION_POINTERS pep) { m_er = *pep->ExceptionRecord; m_context = *pep->ContextRecord; }

    // Construtor de Cópia
    SystemException(SystemException& e) { m_er = e.m_er; m_context = e.m_context; }

    //Coerção para obter o código da excepção
    operator DWORD() { return(m_er.ExceptionCode); }
    // É chamada automaticamente pelo run-time do C++ quando ocorre uma excepção; Origina a criação do objecto C++
    static void _cdecl TranslateSEH(UINT dwEC, PEXCEPTION_POINTERS pep);
};

//Função registada para converter excepções SEH em objectos da classe SystemException
void _cdecl SystemException::TranslateSEH(UINT dwEC, PEXCEPTION_POINTERS pep) {
    throw SystemException(pep);
}
```


SEH: Integração com as exceções C++ - Exemplo (II)

```
int main()
{
    SystemException::MapSystemException(); // Activar a conversão SEH em exceções C++

    try
    {
        int *p = NULL;
        *p = 9;
    }
    catch ( SystemException e )
    {
        printf("%x \n",DWORD(e));
    }
}
```