PROGRAMAR EM *ASSEMBLY*

(PARTE VI)

São muitos os leitores que nos têm «corajosamente» companhando, ao longo de mais de um ano, nesta viagem pelo mundo dos PC e do *Assembly*. Chegamos agora à última etapa do percurso, e tal como aconteceria num cruzeiro turístico (perdoem-nos a comparação) as opiniões dos excursionistas são muito variadas.

Enquanto alguns dos viajantes afirmam, com uma réstea de tristeza, que não conseguiram ver «nada de nada» até este momento, alguns outros dizem, com um arzinho de frustração, que o que viram não foi «nada de novo», e temos também aqueles que pensando que iam ver uma coisa acabaram afinal por ver outra, para não falar dos que já fizeram melhores viagens e, ainda, daqueles poucos, os previligiados, que sentem que se tornaram especialistas no percurso ouvindo apenas o discurso do guia.

A diversidade de opiniões, moldada por diferentes graus de sensibilização e permeabilidade à mesma informação, é efectivamente muito característica, não só dos excursionistas, mas também, e porque não generalizar, dos seres humanos em geral.

Porém, foi com satisfação que podemos constatar, aqui e ali, que alguns leitores conseguiram mesmo iniciar-se no *Assembly* apenas com a informação dada até este momento neste curso.

Muito em particular a lição Nº IV e o pequeno programa RELOGIO.ASM que o acompanhava, parecem ter sido extraordinariamente motivadores e um verdadeiro motor de arranque para alguns leitores que começaram a sentir que, afinal, eram também capazes de fazer algo em linguagem *Assembly*.

Um dos leitores afirmou mesmo que, após estudar o programa RELOGIO.ASM, teve a mesma sensação que tivera muitos anos atrás, na escola primária, quando sentiu pela primeira vez que sabia ler...

Muito interessante é afinal tudo isto, mas passemos à nossa lição final, que aborda um tema «extraordinariamente misterioso», os *Device Drivers*.

E tão misteriosos são estes programas que nem existe uma tradução satisfatória em língua portuguesa para o respectivo nome (pelo menos uma tradução que não toque erradamente campaínhas e repique sinos dentro do nosso imaginário).

Como complemento a esta lição, incluímos na disqueta que acompanha a revista um programa e o respectivo código fonte, PAUSA.EXE e PAUSA.ASM respectivamente, que nos ilustram o segredo de produção daqueles programas «híbridos» que são simultâneamente, *Device Drivers* e programas de estrutura .EXE.

Bem, é que os programas híbridos estão a ficar na moda, quem não conhece alguns «híbridos» como o SMARTDRV.EXE ou o STACKER.EXE ?

Por conseguinte, o nosso programa PAUSA.EXE, para além de ser muito instrutivo, também é util, mas para saber mais sobre ele, nada como ler todos os comentários que escrevemos no ficheiro PAUSA.ASM.

Mas não cobrimos todo o terreno dos «híbridos», pois vamos tratar aqui daqueles outros que são simultaneamente *Devices Drivers* e ficheiros de estrutura .COM, mas alegrar-nos-ia muito saber que algum leitor descobriu por si o pequeno truque. Alguém quer dar uma sugestão ?

OS *DEVICE DRIVERS*

Dito o mais simplesmente possível, *Device Drivers* são programas que o leitor «obriga» o DOS a executar colocando os respectivos nomes no conhecido ficheiro de texto de nome CONFIG.SYS antecedidos da palavra «DEVICE=».

Quando o DOS termina a sua instalação em memória, após o arranque do computador, a primeira actividade que se propõe sempre realizar é ler o conteúdo do ficheiro CONFIG.SYS linha a linha.

E, durante esse processo, quando o DOS encontra uma linha que se inicie com «DEVICE=» (claro que, mais recentemente, também poderia ser «DEVICEHIGH=») assume que o nome a seguir ao símbolo «=» é o nome de um *Device Driver* e fá-lo executar de um modo muito especial.

E por a execução de um *Device Driver* se processar diferentemente da de um programa de estrutura .EXE ou de estrutura .COM. daí o termos considerado muito interessante incluir este tópico avançado (embora a sua abordagem tenha de ser necessariamente um pouco elementar) nesta última lição sobre *Assembly*.

PARA QUE SERVEM OS *DEVICE DRIVERS*?

Bem, a utilização de *Device Drivers* pode servir um de quatro objectivos:

1 – Auxiliar a instalação de novo equipamento no computador.
Como se sabe, o DOS vem de origem preparado para integrar no Sistema vários periféricos tais como discos rígidos, *drives* de disquetes, impressoras, monitores e teclados. Contudo, é sempre possível imaginar que se venha a ter a necessidade de instalar um novo tipo de equipamento, que para poder funcionar convenientemente em boa harmonia quer com o DOS quer com o BIOS, necessite de adicionar ao Sistema algum *software* adicional. Pode ser por exemplo um rato, uma placa de som, ou um *tape streamer*.
E um dos métodos mais utilizados para resolver o problema é precisamente recorrendo a um Device Driver.
Quem não conhece o DRIVER.SYS ou o MOUSE.SYS?
O primeiro permite adicionar ao Sistema uma *drive* não convencional e o segundo deverá fazer funcionar o nosso rato (se for compatível com os «ratinhos» da Microsoft, bem entendido).
O DOS (desde a versão 2.0) dispõe de um *interface* normalizado capaz de integrar este tipo de programas desde que eles se proponham actuar segundo um conjunto muito certinho de regras e aceitem um determinado modelo de comportamento de entre vários à escolha.
E para isso o *Device Driver* dispõe de um *Header*, logo no seu ínicio, através do qual informa sempre préviamente o DOS qual o modelo que quer seguir. Mais especificamente, essa informação consta de um número de 16 bits existente no *Header* e que é designado por Atributo.

Veja no quadro I a estrutura do *Header* dos *Device Drivers*.

Quadro I - Device Driver Head

Nome	Descrição	Offset	Comprimento
LinkSEg	Endereço do segmento do próximo *Device Driver* na cadeia. Se se tratar do último o valor será 0FFFFh (ou simplesmente -1).	0	word
LinkOff	Endereço do Offset do próximo *Device Driver* na cadeia. Se se tratar do último o valor será 0FFFFh (ou simplesmente -1).	2	word
Atributo	Ver Quadros II-A e II-B	4	word
Strategy	Endereço do Offset da rotina «Interrupt»	8	word
Nome	Nome do *Device Driver*	10	8 bytes

2 – Outros vezes não se utilizam *Device Drivers* para instalar novos equipamentos mas sim para modificar a operação de equipamento já existente. O conhecidíssimo ANSI.SYS é um exemplo disso. Ele actua modificando a operação do teclado e a saída de caracteres para o monitor.
E também o SMARTDRV.EXE que acelera o acesso ao disco rígido e às *drives* de disqueta através da criação de um *buffer* em memória.

Quadro II – A – Atributo

Função	Valores	Bit
Tipo de *Device*	1=«Character Device»	15
Suporta *strings* de controlo	0=Não 1=Sim	14
Suporta «*output until busy*»	0=Não 1=Sim	13
RESERVADO	Deve ser 0	5, 8, 9, 10, 12
Suporta «*Open/Close*»	0=Não 1=Sim	11
Suporta «*IOCTL queries*»	0=Não 1=Sim	7
Suporta funções IOCTL e/ou mapeamento lógico de drives	0=Não 1=Sim	6
Suporta «*Fast Character Output*»	0=Não 1=Sim	4
«*Clock Device*»	0=Não 1=Sim	3
«*Nul Device*»	0=Não 1=Sim	2
«*Console Output Device*»	0=Não 1=Sim	1
«*Console Input Device*»	0=Não 1=Sim	0

Quadro II – B – Atributo

Função	Valores	Bit
Tipo de *Device*	0=«*Block Device*»	15
Suporta *strings* de controlo	0=Não 1=Sim	14
Usa FAT ID	0=Não 1=Sim	13
RESERVADO	Deve ser 0	0, 2, 3, 4, 5, 8, 9, 10, 12
Suporta «*Open/Close*»	0=Não 1=Sim	11
Suporta «*IOCTL queries*»	0=Não 1=Sim	7
Suporta funções IOCTL e/ou mapeamento lógico de drives	0=Não 1=Sim	6
Suporta sectores com endereço de 32 bits	0=Não 1=Sim	1

3 – Muitas vezes recorre-se a *Device Drivers* para emular ou simular equipamento que na realidade não existe. É o caso de uma *Drive* em RAM, como o faz o programa RAMDRV.SYS. Ou o caso de uma *Drive* maior como o faz o famoso STACKER.EXE. É também o caso do EMM386.SYS que emula memória expandida a partir de memória estendida.

4 – Finalmente, utilizam-se ainda *Device Drivers* para implementar padrões de actuação que se pensa ser útil que sejam seguidos por todos os programas que façam uso de certas características do Sistema. Por exemplo, o acesso à memória estendida pode ser processado de muitas maneiras. O *Device Driver* HIMEM.SYS implementa uma delas, a especificação XMS (*Extended Memory Specification*) a qual é seguida estritamente pelo MS-DOS.

COMO EXECUTAM OS *DEVICE DRIVERS?*

Os *Device Drivers* executam de um modo que «não lembrava nem ao diabo»!

Para se tentar compreender, vejamos:

Os *Device Drivers* consistem estruturalmente de três partes: o *Header*, a rotina *Strategy*, e a rotina *Interrupt*.

O *Header* (ver quadro I) é uma estrutura de 18 bytes que contém o Atributo (ver quadro II-A e II-B) do *Device Driver*, e os endereços das rotinas *Strategy* e *Interrupt*.

O DOS irá comunicar com o *Device Driver* através de uma estrutura que cria propositadamente para o efeito e que é designada por *Request Header*.

Entre outras coisas (ver também o código fonte do nosso programa PAUSA.ASM) o DOS irá usar o *Request Header* para passar para o *Device Driver* o número da função que pretende que seja executada. Com o DOS 5.0 está prevista a existência (nem todas foram ainda implementadas) de um total de 25 funções normalizadas (ver quadro III).

O DOS também utilizará o *Request Header* para passar endereços de *buffers* ao Device Driver. Por sua vez, o *Device Driver* utilizará o *Request Header* para devolver ao DOS códigos que o informarão da situação e, eventualmente, de quaisquer erros que tenham acontecido. Deste modo o DOS procede assim:

a) Constrói o *Request Header*
b) Chama a rotina *Strategy* e passa-lhe o endereço do *Request Header* .
c) Chama a rotina *Interrupt*.

A rotina *Interrupt* examina o *Request Header*, vê que função o DOS lhe pede que execute e procede em conformidade. No final a rotina *Interrupt* introduz no *Request Header* um código que informe o DOS se tudo correu bem, ou o que correu mal, e devolve de seguida o controle ao DOS.

No nosso programa exemplo (PAUSA.EXE) apenas é executada a função número 0 (INIT). O PAUSA.EXE não permanecerá residente e, por conseguinte, nenhuma outra função fará sentido.

CLASSIFICAÇÃO DOS DEVICE DRIVERS

Existem dois tipos fundamentais de *Device Drivers*:

– Os que se destinam a equipamentos ou dispositivos do tipo em que a informação flui em série, byte a byte ou carácter a carácter, tal como o teclado, a impressora e o modem.
– E os que se destinam a equipamentos ou dispositivos nos quais a informação é tratada por bloco de dados previamente identificados por um endereço de bloco.

Quadro III – Funções (Request Header)

Função	Definição
00H	Init
01H	Media Check *(Block Devices)*
02H	Build BPB *(Block Devices)*
03H	IOCTL Input
04H	Input from Device
05H	Non-Destructive Read
06H	Input Status
07H	Flush Input *(Character Devices)*
08H	Write Output to Device
09H	Write Output with Verify *(Block Devices)*
0AH	Output Status *(Character Devices)*
0BH	Flush Output *(Character Devices)*
0CH	IOCTL Output
0DH	Open Device
0EH	Close Device
0FH	Removable Media *(Block Devices)*
10H	Output Until Busy
11H – 12H	Reservado
13H	Generic IOCTL Request
14H – 16H	Reservado
17H	Get Drive Map *(Block Devices)*
18H	Set Drive Map *(Block Devices)*
19H	IOCTL Query

Os primeiros designam-se por *Character Devices* e os segundos por *Block Devices*.

Os *Character Devices* são identificados por terem o 16º bit (que é o bit 15, já que estes se numeram a partir de 0) do Atributo nulo e os *Block Device* por terem esse mesmo bit com o valor 1. Os outros bits do Atributo fornecerão outro tipo de informação ao DOS, a qual está esquematizada nas tabelas II-A e II-B.

QUE MAIS HÁ A SABER ?

Muita coisa ficou ainda por dizer, e não só sobre os *Device Drivers*.

A metodologia que adoptámos no decorrer deste curso foi associar constantemente a linguagem *Assembly* ao mundo que a circunda, isto é o PC, o DOS e o BIOS.

Como o leitor se poderá finalmente ter já apercebido, a linguagem *Assembly* em si mesma é extremamente simples.

O nosso programa exemplo de hoje não utilizou mais nenhuma mnemónica *Assembly* que não tivesse já sido utilizada nos programas exemplo das lições anteriores.

Dificil para o programador de *Assembly* é, efectivamente, ter um conhecimento prático bastante sólido do ambiente em que se desenvolve a própria programação em *Assembly*.

O tema particularmente difícil, diga-se de passagem, que afloramos hoje (embora muito superficialmente) pretendeu, acima de tudo, mostrar ao leitor (menos conhecedor e experiente) o tipo de interessantes desafios que o candidato a programador em linguagem *Assembly* pode ter «corajosamente» de enfrentar.

José Páscoa