Arquitetura de Computadores 2016/17

Mini-Projeto v2

Prazo de entrega: 23h59 de 22 de maio de 2017

Tópicos: *Manipulação direta dos dispositivos de Entrada/Saída (IN/OUT) usando interrupções. Exemplo da porta série para troca de ficheiros entre computadores.*

Introdução

Na utilização dos periféricos de um computador pode ser necessário testar primeiro o estado desses dispositivos antes de prosseguir com a sua operação. Por exemplo, é muito comum ter de verificar se um periférico está desocupado antes de o mandar executar novas operações. Para tal, pode-se aguardar num ciclo onde se testa repetidamente o seu estado, até que este esteja no estado desejável (espera ativa). Esta abordagem tem vários inconvenientes, incluindo ficar bloqueado nessa espera com desperdício de tempo e processamento do CPU. Assim, todas arquiteturas costumam suportar a possibilidade do CPU ser notificado, em qualquer altura, de determinados acontecimentos, como por exemplo, os dispositivos indicarem a alteração no seu estado para que determinado código seja executado. Tal é o mecanismo de interrupções, que se pretendo utilizar neste trabalho, em particular as respeitantes ao controlador da porta série (UART).

Este trabalho baseia-se na ficha prática 7, devendo ter presente o seu enunciado e todos os ensinamentos que obteve na sua resolução. Como exercício, pretende-se neste mini-projeto, realizar a troca de ficheiros (com qualquer conteúdo, texto ou *binários*) entre dois computadores ligados via porta série (RS-232). Neste trabalho todo o desenvolvimento será efetuado em ambiente simulado, recorrendo à aplicação *qemu*, executando máquinas virtuais com um sistema semelhante ao MS-DOS. Os programas a desenvolver serão implementados em C, variante TurboC, recorrendo às suas funções de biblioteca para as operações de IN/OUT, para a manipulação do vetor de interrupções e para a implementação da rotina de atendimento das interrupções.

Trabalho a realizar

Pretende-se desenvolver dois programas dedicados à transferência de ficheiros pela porta série (COM1). Um dos programas será usado para enviar um ficheiro (ler o ficheiro do disco e enviá-lo pela porta série); o outro para receber os dados que chegam pela porta série e guardá-los em ficheiro no disco (lembre-se que o emissor e o recetor estão a correr em máquinas virtuais distintas). Para o correto funcionamento da transferência deve ser acordado um protocolo entre ambos os programas que permita a cada um destes confirmar que o outro está pronto, que a transferência decorre até todo o conteúdo do ficheiro ser transferido e só terminam após a sua completa transferência (p.e. o programa que recebe tem de saber quando terminam os dados do ficheiro para terminar o programa). Pretende-se também que o programa que recebe leia o teclado e, de cada vez que o utilizador introduza ENTER (ou RETURN), afixe no ecrã o número total de bytes a transferir e quantos é que já foram transferidos até ao momento.

Como protocolo para o correto início da transferência sugere-se o seguinte:

1. O emissor, para aguardar pelo receptor e indicar o tamanho do ficheiro, envia uma string com a palavra “send”, seguida de um espaço e ainda o número de bytes a enviar. Exemplo “send 1045”. Deve depois aguardar por uma resposta do receptor. O programa receptor deve ser executado antes do emissor.

2. O receptor aguarda pela recepção da string anterior, obtém o número de bytes e confirma estar pronto, enviando ao emissor a string: “receive”, seguida de um espaço e do número de bytes que obteve. Exemplo “receive 1045”. O emissor, recebendo esta string e confirmando que o número de bytes está correto, pode enviar todo o ficheiro.

Recomenda-se que desenvolva o seu trabalho pela ordem dos objetivos apresentados a seguir.

1. Enviar um ficheiro pela porta série COM1.

Faça um programa que dado o nome de um ficheiro, indicado na linha de comando, lê os bytes do ficheiro e os envia, um a um, pela porta série. Utilize a função **`sendSerial`** que implementou na ficha 7. Pode testar, esta parte, com um ficheiro de texto, como fez na ficha 7. Pode depois acrescentar o protocolo inicial que verifica se o receptor existe e que passa a dimensão do ficheiro. Veja a função de biblioteca C **`fread`**, para ler um bloco de bytes e **`stat`** para obter informação sobre o ficheiro, incluindo a sua dimensão (ver Anexo).

Note que em DOS e Windows o fim de linha é codificado com dois bytes e não apenas com '\n' como previsto na linguagem C. Assim, neste trabalho, na abertura de um ficheiro, devemos indicar que não queremos interpretar os fins de linha, indicando `fopen( nomef1, "rb")` e `fopen( nomef2, "wb")`, para que o tamanho do ficheiro em bytes seja igual ao número de caracteres lidos e escritos.

A transferência entre computadores virtuais vai decorrer rapidamente, tornando-se difícil ter tempo para premir a tecla ENTER do lado do receptor e ver os bytes transferidos. Coloque na função sendSerial (ou equivalente) uma pequena espera usando `delay(50)`, para que a comunicação seja mais lenta.

2. Receber ficheiro pela porta série COM1.

Implemente agora um programa para receber um ficheiro pela porta série. Este deve aguardar pelo emissor e seguir o protocolo antes descrito. Os bytes recebidos devem ser guardados no ficheiro de nome indicado na linha de comando quando o programa for executado. Como neste programa é pedido que afixe o número de bytes recebidos de cada vez que o utilizador der ENTER, vamos necessitar de separar a recepção dos bytes da parte responsável por ler o teclado e por afixar esse número no ecrã. A abordagem exigida é que o programa, imediatamente antes de indicar a confirmação ao emissor, programe o mecanismo de interrupções para que a recepção seja efectuada autonomamente pela rotina de atendimento de interrupções respectiva. Todos os caracteres que cheguem pela porta série são recebidos por essa rotina e guardados num *buffer* para que não se percam.

De seguida, o programa principal fica num ciclo testando sucessivamente o teclado e o buffer e deve detetar as seguintes situações:

1. O utilizador premiu uma tecla e essa corresponde à mudança de linha ('\n'): o programa deve escrever no ecrã o número de bytes recebidos e o número total a receber. Esta abordagem deve permitir também terminar o receptor em qualquer altura premindo Ctrl+C.
2. Verifica-se que existem bytes já recebidos no buffer: deve retirar um byte e escreve-lo no ficheiro respectivo.

Este ciclo repete-se até que se verifique que todos os bytes foram escritos no ficheiro e que está na altura de terminar o programa. Veja um exemplo de implementação de um *buffer* nos slides da aula 16 (e disponível no CLIP).

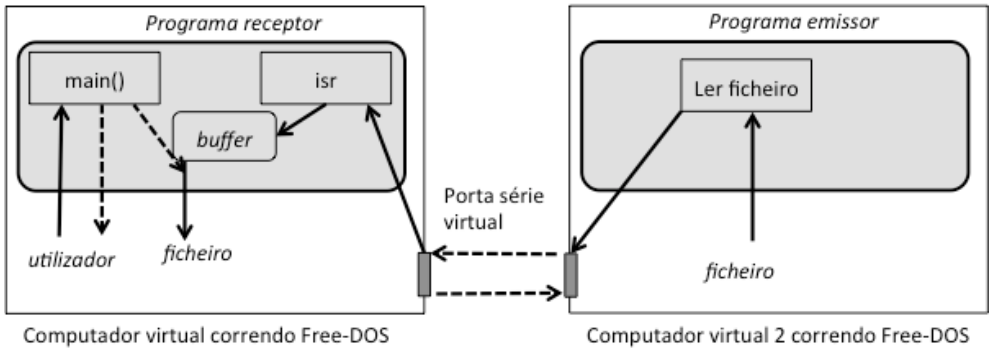
Na figura seguinte ilustra-se a arquitetura destes programas durante uma transferência, sendo que **`isr`** representa a rotina de atendimento de interrupções (ou *interrupt service routine*).

3. *Transferência* entre dois "computadores diferentes".

Copie a imagem do FreeDOS (p.e. `cp fdos.img fdos2.img`) para representar um segundo computador. Deve lançar em simultâneo duas instâncias do *qemu*, cada uma com a sua imagem, para simular dois computadores a executar em simultâneo. Deve usar os seguintes comandos (em terminais diferentes) para que as portas série sejam simuladas ligadas uma à outra:

```
qemu-system-i386 -serial udp::9000@:9001  fdos.img
qemu-system-i386 -serial udp::9001@:9000  fdos2.img
```

Tenha atenção aos números usados em cada comando. Substitua *fdos.img e fdos2.img* pelos nomes das suas imagens. Ambos vão usar e partilhar um canal de comunicação, que simula as portas série ligadas entre si. Durante o funcionamento de ambos os programas, os bytes enviados pelo emissor que executa num "computador", serão recebidos pelo receptor que executa no outro "computador".



Lembre-se que quando altera o seu programa numa imagem, não o está a alterar na outra e deve por isso, manter uma disciplina rígida de programar sempre usando a imagem `fdos.img` e, para voltar a testar, repetir a cópia dessa imagem para `fdos2.img`, de forma a usar sempre a última versão do seu programa em ambos os computadores virtuais.

Anexo

Biblioteca do TurboC

Na biblioteca do TurboC, através do ficheiro de cabeçalhos `<dos.h>`, são disponibilizadas as seguintes funções para aceder, a partir do C, às instruções máquina **`cli`** e **`sti`**, dispensando o uso de *assembly* no seu programa:

`void` `enable(` **`void`** `);` - função do TurboC para ligar as interrupções no CPU (sti);

`void` `disable(` **`void`** `);` - função do TurboC para desligar as interrupções no CPU (cli);

Estão também disponíveis funções para alterar o vetor de interrupção, permitindo indicar a respetiva função de tratamento das interrupções:

`void` `setvect(` **`vec, func`** `);` - Para colocar na entrada *vec* do vetor de interrupções o endereço da uma rotina de tratamento de interrupções *func* (este é um apontador para uma função);

`fun_ptr` `getvect(` **`vec`** `);` - Para consultar a entrada *vec* do vetor de interrupções devolvendo o endereço lá instalado (devolve um apontador para função).

Segue-se o exemplo de como declarar uma função que será usada como rotina de atendimento de interrupções e de como a colocar no vetor de interrupções (neste código é usada a posição 12 do vetor como exemplo). Tenha particular atenção à declaração da função a usar como rotina de atendimento de interrupções. Esta não pode ser uma função C normal, pois não será chamada com `call`, mas sim pelo mecanismo *hardware* de interrupções e também não pode terminar com a instrução `ret`, mas sim com `iret`. A declaração de uma função em TurboC com a palavra "**`interrupt`**" garante que o compilador gera o código necessário para esta situação, em vez duma função normal, garantindo também que todos os registos são preservados.

```
#include <dos.h>
. . .
void interrupt my_isr(){
. . . /* código do atendimento de interrupção */
```

```
    outportb(0x20, 0x20);
}


. . .

int main(int argc, char *argv[]) {
. . .
    oldisr = getvect( 12 );
    setvect( 12, my_isr );
. . .
}
```

Para compilar o seu programa no TurboC, garanta que desligou as seguintes opções:

- Em "Options / Compiler / Optimization" tem "Use register variables" a "Off";
- Em "Options / Linker" tem "Stack warning" também a "Off".

É conveniente usar uma pilha (stack) maior que a default:

- Em "Options / Compiler / Model" escolha "Large";

Informação adicional:

O compilador TurboC é já antigo e não reconhece algumas das facilidades mais recentes da linguagem C. Por exemplo:

- não se pode usar **`//`** como indicação de comentário (apenas **`/* */`**)
- as variáveis têm de ser declaradas logo depois das **`{`** e não no meio do código
- também não se pode declarar a variável na inicialização do ciclo **`for`**

No editor do TurboC pode sempre obter uma breve descrição das funções colocando o cursor sobre o nome da função e premindo Ctrl+F1.

Para suportar a compilação e ligação de vários ficheiros C num único programa, deve indicar num ficheiro de texto com extensão `.prj` os nomes de todos os ficheiros `.c` que fazem parte do programa. Dentro do TurboC deve depois indicar esse ficheiro `.prj` no "Project name".
Recomenda-se que altere as opções necessárias e indique o nome do projeto e, depois, grave todas estas configurações usando "Save options" para o ficheiro `tcconfig.tc`, na sua diretoria de trabalho. Assim, de cada vez que executar o **`tc`** nessa diretoria, essas configurações serão automaticamente carregadas.

Para ler e escrever na imagem do Free-DOS, a partir do Linux, pode montar esta como se fosse um disco externo usando o botão direito e escolhendo "Disk Image Mounter" (ou algo semelhante). Pode acontecer que a imagem fique sempre em modo *read-only*, i.e., só consegue ler os ficheiros do Free-DOS mas não consegue criar novos ficheiros, nem copiar ficheiros para essa imagem. Neste caso execute a "montagem" à mão, a partir da linha de comando, num terminal:

```
gnome-disk-image-mounter  -w  fdos.img
```

se necessário instale este comando fazendo: `sudo apt-get install gnome-disk-utility`

Outras funções C que podem ser úteis:

`int` **`kbhit`**`( )` – devolve um valor diferente de zero se alguma tecla do teclado for premida (incluir `<conio.h>`)

`void` **`sleep`**`( unsigned seconds )` – suspende o programa pelo tempo indicado

`int` **`getchar`**`( )` – lê o próximo carácter do *stdin*

`int` **`sscanf`**`( char *str, char *format,...)` - *scanner* para interpretar e ler, os valores na *string* `str` de acordo com a *string* de formatação e tipos em `format`.

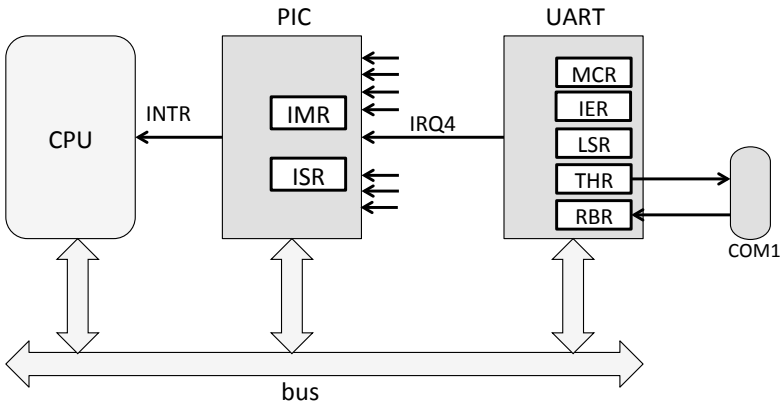
`int` **`sprintf`**`( char *str, char *format,...)` – equivalente ao `printf` mas escrevendo a *string* formatada na *string* indicada por `str`.

Para obter o tamanho de um ficheiro com o nome em `nomef`, use o seguinte código:

```
struct stat info;
stat( nomef, &info );
size = info.st_size;
```

Uso de interrupções da porta série

O controlador da porta série, já visto na aula prática 8, dispõe de mais alguns registos para ser programado para gerar interrupções em determinadas situações. Estas interrupções são sinais electicos que ligam a UART ao controlador de interrupções (PIC) que por sua vez os propaga ao CPU para este desencadear o tratamento de interrupção, ou seja, a execução do código que está indicado no respectivo vetor de interrupções. Este PIC dispõe também dos seus registos de IO onde se indica que interrupções podem ser passadas ao CPU (ou não) e para indicar a terminação da rotina de tratamento de cada interrupção (para que outras possam ser passadas pelo PIC ao CPU). A figura seguinte ilustra estas ligações, considerando apenas a porta série, ignorando todo o restante hardware.



Endereços de I/O da UART (COM1):

- Registo de Dados – Transmissão (THR): 0x3F8
- Registo de Dados – Recepção (RBR): 0x3F8
- Registo de Controlo das Interrupções (IER): 0x3F9
- Registo de Controlo do Modem (MCR): 0x3FC
- Registo de Estado (LSR): 0x3FD

Quando é escrito um byte no registo de dados de transmissão (**THR**), este passa logo que possível para o *shift register* de envio (chamado TSR) após o que começa a ser enviado pela linha série. Quando o **THR** fica livre o bit 5 do registo de estado é colocado a 1. Se programar a UART para gerar interrupções no envio, será também enviado uma interrupção via IRQ4.
Quando chega um byte pela porta série, este fica no registo de dados de recepção (**RBR**) e o bit 0 do registo de estado é colocado a 1. Se programar a UART para gerar interrupções na chegada de cada byte, será também enviada uma interrupção via IRQ4.

Existem outros registos que são relevantes para este trabalho na operação do PIC:

- Registo de Máscara (IMR): 0x21
- Registo de Comandos (ISR): 0x20
  - Comando relevante para este trabalho: Fim de Interrupção (EOI) escrever 0x20 no ISR

Note que a relação entre IRQ que chega ao PIC e qual a posição do vetor de interrupções que contém o endereço da rotina a chamar, é efectuado no arranque do computador ficando fixo. No caso do MS-DOS esta definição para o IRQ4 da COM1 corresponde à entrada 12 no vetor de interrupções.

Sugerimos que implemente funções que permitam ligar e desligar as interrupções da porta série e ligar e desligar o atendimento de interrupções da linha IRQ4 do PIC, de acordo com os passos seguintes.

Ligar interrupções da porta série

1. No vetor de interrupções:
   1.1. É necessário indicar na entrada 12 o endereço da função que vai atender as interrupções (use a função setvect). Guarde previamente o valor nessa posição para o repor, quando desligar as suas interrupções.
2. Na UART é necessário:
   2.1. colocar o bit 3 do registo MCR – Model Control Register (porto 0x3FC) a 1, para permitir as interrupções da UART;
   2.2. colocar a 1 o bite 0 do registo IER – Interrupt Enable Register (porto 0x3F9), para gerar interrupções quando chega um byte; ou colocar a 1 o bit 1 do registo IER, para gerar interrupções quando o registo THR fica pronto a enviar mais um byte.
3. No PIC é necessário:
   3.1. colocar o bit 4 do registo máscara (porto 0x21) a 0, de modo a deixar passar as interrupções que vêm da porta série.

   Nota: a alteração do registo máscara do PIC deve ser efectuada com as interrupções do CPU desligadas (i.e. com a *interrupt flag* a 0). Use `disable()` e `enable()`.

Desligar interrupções da porta série

1. Na UART é necessário:
   1.1. Repor a zero os bits antes alterados no MCR e no IER.
2. No PIC é necessário:
   2.1. colocar o bit 4 do registo máscara (0x21) a 1.
3. No vetor de interrupções: deve repor o endereço que lá estava antes da sua inicialização.

Nota 1: como o *buffer* pode ser acedido em qualquer altura pela rotina de tratamento de interrupções, nas operações chamadas a partir do programa principal, deve garantir que as variáveis partilhadas com a rotina de atendimento são acedidas com as interrupções do CPU desligadas.

Nota 2: quando a rotina de tratamento de interrupções é chamada, as interrupções são desligadas. Devemos ligá-las o mais cedo possível, pois podem ocorrer interrupções mais importantes (de nível mais elevado) que seja importante tratar imediatamente.

Nota 3: Os ritmos de leitura/escrita de comunicação via porta série, escrita em ficheiro e leitura do teclado, são normalmente diferentes levando a que possam acontecer situações em o *buffer* fique cheio ou vazio. Tenha em conta estas situações no seu dimensionamento e as consequências de ficarem cheiros ou vazios para o funcionamento do mecanismo de interrupções (p.e. se a rotina de atendimento devia receber um byte mas o *buffer* está cheio, não o pode guardar e este terá de ser perdido).