Arquitetura de Computadores 2017-18

Ficha 9

Tópicos: *Entradas/saídas, espera ativa e ficheiros em C.*

Introdução

Como visto nas aulas teóricas, em computadores com uma arquitetura compatível com a família dos processadores *Intel 80x86,* o acesso às interfaces dos periféricos existentes é realizado através do *mapa de Input/Output* (*I/O* ou seja, *entrada/saída*). Este mapa pode corresponder a um espaço de endereçamento separado da memória central, no qual cada periférico ocupa um conjunto de endereços (portos ou portas). Estes conjuntos de endereços (portos) estão mapeados em registos internos do *hardware* do dispositivo periférico e permitem transferir dados, dar comandos e consultar o estado desse dispositivo. Estes processadores dispõem de instruções específicas para *Input/Output* que se designam, em *assembly*, pelas mnemónicas ***in*** e ***out***, respetivamente.

Nos processadores que suportam modos de execução utilizador e supervisor, estas operações estão reservadas apenas ao modo supervisor usado pelo SO, pelo que não podem ser usadas pelas aplicações normais. Para que neste trabalho possa exercitar essas operações, vamos recorrer à sua simulação usando software desenvolvido na linguagem C (fornecido). Este software simula as instruções IN e OUT e um dispositivo periférico fortemente baseado no controlador da porta série RS-232, vista nas aulas teóricas. Neste trabalho esta interface vai servir de pretexto para manipular diretamente um dispositivo de entrada/saída.

O principal objetivo é implementar dois níveis de software:

- a função que poderia fazer parte de um "*device driver*" no SO, que permite enviar bytes pela porta série em modo de espera ativa;
- a função para uso pelos programadores para a escrita de caracteres, simulando um canal *bufferizado*, semelhante ao que se passa na biblioteca do C quando se usa `putchar` para escrever no *standard output* (tipicamente o ecrã).

O código fornecido contém programas de teste e vem pronto a ser compilado com o comando `make` como descrito mais à frente.

Simulação da porta série

O código fornecido procura simular, de uma forma simplificada, o funcionamento das instruções IN e OUT do CPU e o controlador de uma interface de comunicação série, como visto na aula teórica. Para que se possa facilmente observar o que um programa envia para a interface série simulada e para enviarmos facilmente caracteres para o programa, o envio e a recepção desta interface serão na realidade simulados escrevendo ou lendo ficheiros normais ou de/para terminais de linha de comandos.

Controlador da interface série: a UART

Cada interface, ou porta, série nos PCs é **controlada** por uma **UART** (Universal Asynchronous Receiver/Transmiter) — circuito integrado *NS 8250* ou compatível (por exemplo: 16450, 16550, 16C552). Esta é operada através de um conjunto de registos (ver figura 3), acessíveis por ***portos de I/O*** colocados a partir do endereço inicial definido pelo respectivo descodificador de endereços. Geralmente, o endereço inicial da **primeira porta série** é 0x**3F8**. O nome atribuído a esta porta série pelo sistema de operação MS-DOS e MS-Windows é de *COM1*:.

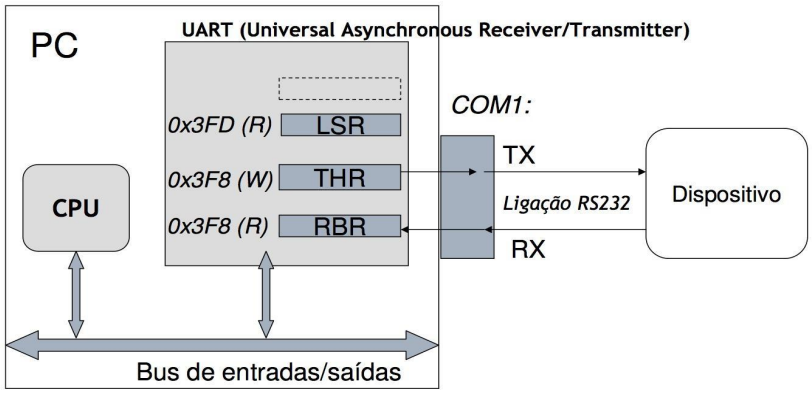

Figura 3: Interação entre o CPU, a UART e a porta série

Na tabela seguinte são apresentados os registos e portos relevantes para este trabalho:

Registo	**Abrev.**	**Acesso**	***Port***
Dados – Transmissão (TX)	THR	Escrita	0x3f8
Dados – Recepção (RX)	RBR	Leitura	0x3f8
Estado do controlador	LSR	Leitura	0x3fd

De notar que os registos **RBR** e **THR** partilham o mesmo endereço de I/O — o primeiro é acedido através da instrução ***in*** e o segundo da instrução ***out***. Quando o software envia um byte para o registo de dados de transmissão (**THR**), o controlador da porta automaticamente trata do seu envio. O registo de estado, **LSR**, dá informação sobre a recepção e transmissão dos dados. Cada um dos seus bits fica a 1 caso a condição enunciada seja verdadeira e a 0 se falsa:

bit 0	Chegou um caráter que está disponível no registo RBR
bits 1-4	---
bit 5	O registo de transmissão (THR) está livre e pode-se ordenar o envio de um novo byte
bit 6-7	---

Só se deve enviar um byte para o registo de envio (THR) quando o bit 5 do registo de estado (LSR) está a 1. Este bit 5 do registo LSR passa a 0 assim que se escreve em THR. Pode-se ler um byte do registo de dados de recepção (**RBR**) quando o bit 0 do registo de estado (LSR) está a 1. Este bit 0 do registo LSR passará a zero assim que essa leitura seja efetuada.

1. Envio de bytes (*Device driver*)

Pretende-se implementar uma função que permite enviar bytes pela interface série, usando o modo de espera ativa. Esta função poderia fazer parte do "device driver" do SO para usar este periférico. Implemente a seguinte função no ficheiro driver.c:

```
void sendByte( unsigned char byte );
```

Esta função deve utilizar o método de espera ativa para aguardar que seja possível enviar um novo byte, sem destruir outro envio que possa estar em curso. Na sua implementação use as funções **int** e **out** fornecidas via hw.h que simulam as respetivas instruções do CPU para ler/escrever nos *ports* da UART.

Implemente também a função **`waitBussy`**, que aguarda enquanto a porta série está ocupada a enviar bytes. Tal será usada para testar e garantir que os bytes já foram enviados, por exemplo, antes do programa terminar.

Use o programa teste1.c fornecido para testar a sua função. Basta compilar dando o comando "**`make teste1`**". Este programa começa por inicializar o simulador (hwInit de hw.h) e recorre diretamente à sua função para enviar uma sequência de alguns caracteres de uma *string*. A saída pela porta série deste simulador vai aparecer no output standard do seu programa (o ecrã) visto o ficheiro indicado no hwInit, "`/dev/tty`", representar o próprio terminal onde o programa corre. Confirme que a *string* aparece correta.

2. Output de caracteres usando um *buffered stream*

Pretende-se agora simular (uma pequena) parte do comportamento da biblioteca de I/O da linguagem de programação C. Em particular deverá implementar uma função semelhante à função **putchar**, oferecida pela biblioteca da linguagem C. Veja a documentação sobre essa função, no seu livro de C ou consultando o manual em Linux usando o comando "`man putchar`". Estas operações, quando escrevem no ecrã, usam bufferização orientada à linha, ou seja, só quando uma linha está completa (ou o buffer enche) é que os seus caracteres são passados ao SO e escritos no periférico.

Neste trabalho deve assim implementar as funções `myPutChar` para enviar caracteres pela porta série, e `myFlush` para garantir que tudo é enviado e aguardar pelo fim desse envio. Estas devem estar de acordo com as seguintes assinaturas:

```
void myPutChar( unsigned char c );
void myFlush( void );
```

O código fornecido para esta fase do trabalho é o seguinte:

- **mylib.c** Corresponde ao código de "biblioteca" onde deve implementar as duas funções anteriores. A primeira permite enviar um caráter, mas recorrendo a um buffer que armazena temporariamente os caracteres a enviar. O buffer é despejado e os caracteres realmente enviados à porta série usando a sua função do *driver* quando:
  - foi colocado no buffer uma mudança de linha;
  - o buffer ficou cheio;
  - é chamada explicitamente a função `myFlush`.

  A função `myFlush`, para além de despejar o buffer, aguarda também que o último byte tenha sido enviado, usando a função `waitBusy`.
- **main.c** Programa principal de uma aplicação que envia um ficheiro pela porta série. Este programa está incompleto. Implemente o código em falta que deve ler todo o ficheiro byte-a-byte e enviar cada um com recurso à sua função `myPutChar` e terminando com `myFlush`.

Para compilar basta, no terminal, estar na diretoria do código e dar o comando **`make`**.

Testar myPutChar

Para testar e verificar o correto funcionamento da sua função (e do seu buffer), e apenas temporariamente para esse fim, de cada vez que enviar os caracteres para a porta série (em `myPutChar`), envie também o caracter '**$**'. Por exemplo, para um buffer com a dimensão 20 e querendo enviar um ficheiro com os seguintes caracteres:

01234567890123456789012345678901234567890123456789012345678901234

o output realmente observado com este teste, se correr corretamente, será:

01234567890123456789$01234567890123456789$012345678901234
$

Como a linha é longa e o buffer tem um tamanho de 20, a linha será impressa em tranches. As duas primeiras porque o buffer ficou cheio (daí o "$" no meio da linha, o que nos permite confirmar que está a funcionar como esperado) e a última porque encontrou uma mudança de linha.

Mais informação

Veja a documentação sobre as funções `putchar()` e `fflush()` no seu livro de C ou nos manuais do Linux (exemplo: `man putchar`).