Arquitetura de Computadores

MIEI – 2017/18
DI-FCT/UNL
Aula 15

Biblioteca C (exemplos)

- Canais standard (abertos desde início):
  - stdin, stdout, stderr
  - (em Java: System.in, System.out, System.err)
- Abrir/fechar:
  - FILE *fopen(char*name, char*mode);
  - fclose(FILE*stream);
- Escrever/Ler:
  - printf() / fprintf() / fwrite()
  - scanf() / fscanf() / fread()

Helloworld.c

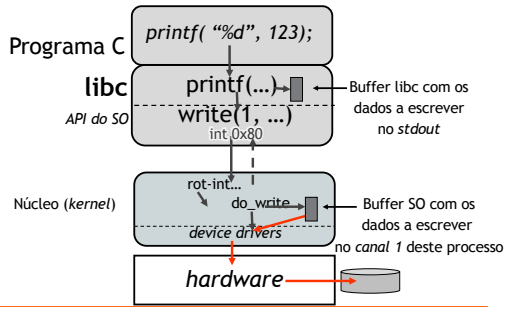
```
#include <stdio.h>
int main( ) {
   printf("Hello world!\n");
   return 0;
}
```

Dentro da libc em Linux Intel 32bits:

```
printf(...)                 // C
    fprintf( stdout, ... )
        write( 1, ... ) // POSIX/Unix
            int 0x80    // Linux/32
```


Vários níveis no IO - exemplo

Níveis e Buffers (exemplo de escrita):



Ficheiro

- Um ficheiro é um espaço de logicamente contíguo, contendo um conjunto de dados, acessível através de um **identificador único** (nome ou caminho).
  - Um ficheiro pode conter qualquer **sequencia de bytes**, sejam dados (texto, imagem, som, etc.) ou programas.
  - A informação é normalmente guardada de forma **persistente**.
- O SO encarrega-se de gerir os ficheiros e os periféricos em que estes residem. A organização interna do periférico é escondida dos programas, programadores e utilizadores.

Tipos de ficheiros em C

- **Ficheiros de texto**
  - Contêm apenas bytes que podem ser interpretados como caracteres imprimíveis: letras maiúsculas e minúsculas, algarismos e sinais de pontuação.
  - É usada uma codificação de caracteres, como por exemplo a ASCII ou Unicode/UTF8.
- **Ficheiros binários**
  - A interpretação do seu conteúdo está a cargo do programa que o utiliza; contém quaisquer bytes.
    - Podem ser dados (inteiros, reais), programas (códigos máquina de um dado CPU), sons (segundo uma codificação como o MP3), imagens (codificadas num formato como o JPEG), filmes (por exemplo no formato MPEG), etc.

Exemplo – escrever texto

```
#include <stdio.h>

int main( )
{    float x = 3.14;
     FILE*f = fopen("fich.txt", "w");
     fprintf(f, "Hello world!\n");
     fprintf(f, "%f", x);
     fclose(f);
     return 0;
}
```


Exemplo – lêr texto

```
#include <stdio.h>
#define SZ  100
int main( )
{    char line[SZ];  float x;
     FILE*f = fopen("fich.txt", "r");
     fgets( line, SZ, f );
     fscanf(f, "%f", &x);
     fclose(f);
     printf("%s%f\n", line, x);
     return 0;
}
```


Exemplo – escrever bytes genéricos

```
#include <stdio.h>

int main(int ac, char*av[])
{    float arr[6]
        ={1.2, 2.3, 3.4, 4.5, 5.6, 6.7};
     FILE*f = fopen("fich", "w");
     fwrite(arr, sizeof(float), 6, f );
     fclose(f);
     return 0;
}
```

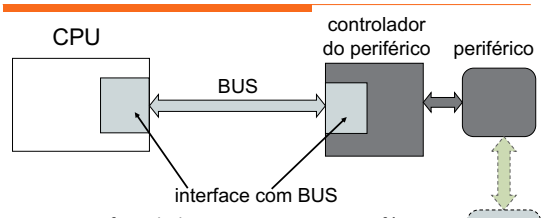

Exemplo – lêr bytes genéricos

```
#include <stdio.h>

int main(int ac, char*av[])
{    float arr[6] ;
     FILE*f = fopen("fich", "r");
     fread(arr, sizeof(float), 6, f );
     fclose(f);
     for (int i=0; i<6; i++)
         printf("%f ", arr[i]);
     return 0;
}
```


- Como o SO garante a persistência da informação?
  - Usa um dispositivo que tenha *memória persistente*: discos, memória de estado sólido, ...
- Como gere o espaço e encontra os dados?
  - Mantém estruturas de dados nessas *memórias* com informação sobre o espaço livre e a localização dos dados (ex. os discos são formatados com um sistema de ficheiros)

Interligação CPU–Dispositivo periférico



- Para transferir dados entre o CPU e o periférico
  - Quem inicia a transferência?
  - Que protocolos (nível eléctrico, ritmo, ...)?
  - Que quantidade de dados transferir?

Outro periférico, rede, etc.

Transferência em série vs. paralelo

- Interface paralelo
  - Num dado instante de tempo, transferem-se vários bits em **simultâneo**;
  - "Largura" da interface é o nº de bits transferidos em paralelo (8bits, 16, ...);
  - Por exemplo, usam-se vários fios para transferir os dados, em que cada fio transporta em simultâneo um bit
- Interface série
  - 1 bit enviado de cada vez (para enviar um byte tem de ser enviada uma **série** de 8 bits)
  - Só um fio para dados (mais a referência ou massa,...)
  - Em princípio mais lento que a interface paralela mas usa menos fios, contactos e electrónica mais simples
    - **A tecnologia tem permitido continuar a criar ligações série cada vez mais rápidas**

Sentido das transferências

- De um modo geral as transferências são bidireccionais. Por exemplo, num disco pode-se ler e escrever.
- A bidireccionalidade pode ser conseguida de duas formas:
  - Interacção ***full-duplex***: são permitidas transferências nos dois sentidos em simultâneo; conseguido através da duplicação das ligações
  - Interacção ***half-duplex***: em cada momento pode-se transferir apenas num sentido; as entidades nas extremidades têm de negociar a inversão do sentido

Sincronização entre os intervenientes

- Os periféricos e o CPU trabalham de forma independente
  - P.ex. a placa de rede pode receber dados da rede enquanto o CPU executa código de um programa
- Não trabalham ao mesmo ritmo, nem partilham um relógio único
- É preciso acomodar diferentes ritmos de transmissão
- Duas formas possíveis
  - O sinal é codificado de forma a que se conseguem identificar as "fronteiras" entre bits
  - Há linhas de controlo extra que transportam um sinal de relógio ou marcam a existência de novos dados

Latência e débito binário

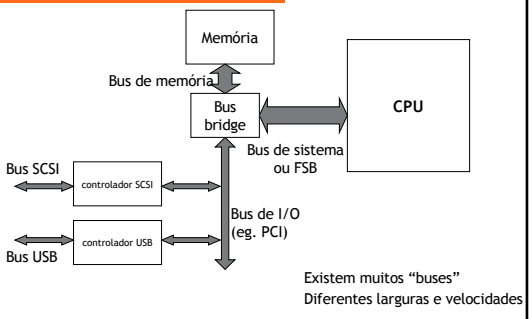
- **Latência** (*latency*): é o tempo que um sinal demora a propagar; ou o tempo que um dispositivo demora a detetar esse sinal; p.e. transferir 1 bit – medida em unidades de tempo (p.ex. ns)
- **Débito** (*throughput*): número de bits que se conseguem transmitir por unidade de tempo; velocidade de um fluxo de bits – medido p.ex. em Mbit/s (Mbps) ou Mbytes/s (MBps)

Compromissos custo/velocidade

Para aumentar o débito binário, dever-se-iam usar interfaces paralelas com um grande número de linhas e full-duplex

- Há muitas desvantagens desta solução: o número de contactos nos circuitos integrados, interferências entre os sinais, o custo das ligações e dos controladores, etc.
- Uma solução: os dados são divididos em blocos que são transmitidos de forma independente. Permite transferir uma quantidade arbitrária de dados sobre uma ligação com um número limitado de linhas.
  - 1 bloco de 512 bytes pode ser transferido em sub-unidades consecutivas, ou byte a byte, ou mesmo bit a bit (em série)

Arquitecturas de computadores/Buses



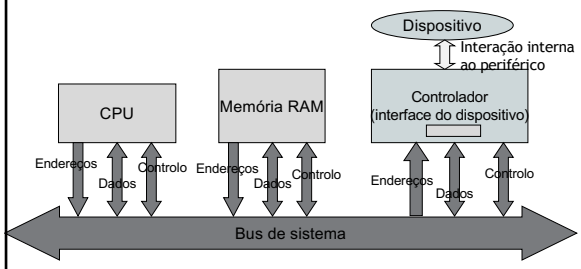
Suporte físico do "bus"

- Fios numa placa (*board*) de circuito impresso ou num cabo
- Fichas (*sockets*) na placa ou para o exterior do computador, ou nos extremos do cabo
  - permitem inserir e remover dispositivos facilmente
- Normas definem:
  - Características físicas anteriores
  - As especificações eléctricas exatas
  - As temporizações dos sinais
  - A forma como os dados são codificados

Organização do "bus"

- Um Bus está normalmente separado funcionalmente em 3 partes:
  - Controlo – controlar a partilha do bus e para indicar o pedido
  - Especificação do endereço (origem/destino dos dados)
  - Dados – envio ou recepção dos dados

Partilha de um BUS



- Como distinguir endereços I/O de endereços de Memória?

Uso do "bus"

- O "bus" permite duas operações (via controlo)
  - Leitura
  - Escrita
- No controlo o CPU assinala se é acesso a memória ou IO
- Estas traduzem-se na transferência dos dados de/para a memória ou as interfaces dos periféricos, como se fossem leituras e escritas em **registos internos** ao controlador do periférico

Acesso em leitura (Load ou Input)

- Usa as linhas de controlo para obter acesso ao "bus"
- Coloca um endereço nas linhas de endereço
- Usa as linhas de controlo para pedir uma operação de leitura, e indicar se memória ou IO
- Testa as linhas de controlo para verificar se a transferência já se concluiu
- Lê o valor das linhas de dados (resposta ao pedido)
- Posiciona as linhas de controlo para deixar outra entidade aceder ao "bus"

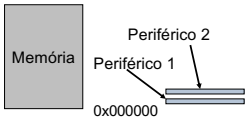
Acesso em escrita (Store ou Output)

- Usa as linhas de controlo para obter acesso ao "bus"
- Coloca um endereço nas linhas de endereço
- Coloca um valor nas linhas de dados
- Usa as linhas de controlo para pedir uma operação de escrita, e indicar se memória ou IO
- Testa as linhas de controlo para verificar se a transferência já se concluiu
- Posiciona as linhas de controlo para deixar outra entidade aceder ao "bus"

Espaço de endereços separado

- Dois espaços de endereçamento separados:
  - Um para a memória
  - Um para os controladores dos dispositivos, os **portos** ou **portas** (*ports*)
- Cada controlador é configurado para responder a uma faixa de endereços diferente
- A faixa de endereços da memória é sempre disjunta da dos periféricos
- O CPU necessita de **instruções diferentes** para separar os acessos a memória dos acessos aos periféricos

Entidade	Faixa de endereços
Memória	0x000000-0xFFFFFF
Periférico 1	0x000000-0x00000B
Periférico 2	0x00000C-0x000017



Instruções dedicadas ao IO

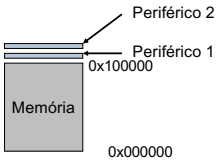
- Espaço separado
  - Há instruções máquina dedicadas ao acesso do espaço de endereços de entrada/saída (chamados *ports*)
  - Exemplo da família Intel x86/Pentium
    - **`IN`** *port, regA* (leitura do *port* para o registo EAX/AX/AL)
    - **`OUT`** *regA, port* (escrita do EAX/AX/AL para o *port*)
      (Se *port* > 255 então: **`IN %DX`**, *regA* e **`OUT`** *regA*, **`%DX`**)
- Exemplo (envio de 1 byte—OUT):

```
mov $5, %al
mov $0x3f8, %dx
out %al, %dx
```


Alternativa: Espaço de endereços único (**memory-mapped I/O**)

- Uma faixa de endereços corresponde à memória
- Cada controlador é configurado para responder a uma faixa de endereços fora do espaço da memória e a que nenhum outro controlador responde
- As faixas de endereços da memória e dos controladores são disjuntos

Entidade	Faixa de endereços
Memória	0x000000-0x0FFFFF
Periférico 1	0x100000-0x10000B
Periférico 2	0x10000C-0x100017



Espaço separado vs. Espaço único

- Espaço único
  - Não há instruções máquina especiais para acesso aos controladores. São usadas *LOAD/STORE* (MOV) para endereços que pertencem à faixa de endereços de entrada/saída
- Exemplo (envio de 1 byte—como um OUT):

```
movb $5, 0x10000C
```


Ponto de vista do CPU

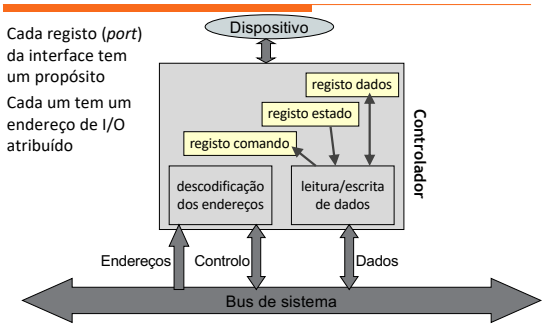
- O controlador do dispositivo fornece uma interface de programação, que tem apenas duas operações:
  - ler e escrever, também chamadas entrada e saída
- Cada periférico possui um controlador com uma faixa de endereços **distinta**
- Os valores escritos em cada *port* têm significados especiais ou desencadeiam ações no controlador
  - Definem **uma linguagem** de operação/controlo específica do periférico
- Para o programador, o bus é invisível

Erros

- Podem ocorrer dois tipos de erros:
  - Conflito de endereços: mais de um periférico tenta responder
  - Endereços não atribuídos: nenhum responde (*timeout*)

Interface genérica de periférico

- Cada registo (*port*) da interface tem um propósito
- Cada um tem um endereço de I/O atribuído



Controlador genérico

- Em geral, as interfaces têm três tipos de registos (portos ou portas):
  1. Porto de dados: para guardar os dados enviados pelo CPU ou recebidos do exterior pelo dispositivo periférico
  2. Porto de estado: indica informação sobre o estado corrente do periférico
     - existência de dados para o CPU ler
     - disponível para receber dados do CPU
     - erros e outras informações
  3. Porto de controlo: para receber comandos vindos do CPU (inicializar interface, configurar a interface para receber ou enviar, e outros comandos que desencadeiam a operação do periférico)