Arquitetura de Computadores

MIEI – 2017/18

DI-FCT/UNL

Aula 14

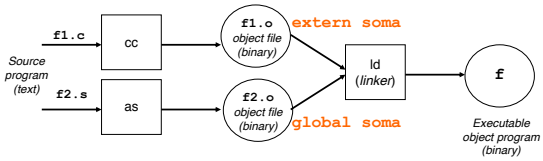
Ligação de C e *Assembly* – Símbolos públicos

```
#include <stdio.h>
// implicito
extern
int soma(int a,int b);

int main(){
      int x, y;

      x = 6;
      y = soma(x,3);
      printf("%d\n", y);
      return 0;
}
```

```
.global soma
.text
soma: push %ebp
      mov %esp, %ebp

      mov 8(%ebp),%eax
      add 12(%ebp),%eax

      pop  %ebp
      ret
```


Ligação de ficheiros "objeto"

- O ficheiro objecto (.o) tem código máquina, num formato independentemente da sua origem, C, Pascal, *assembly*, etc.
- Pode incluir referências a símbolos externos (usados mas não definidos)
- Indica os símbolos exportados (podem ser usados por outros)
- Na ligação (*linking*) todos os símbolos externos de um .o têm de ficar resolvidos pelos exportados por outros ficheiros .o



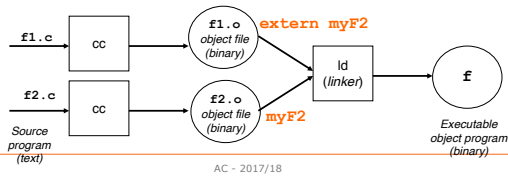
Ligação de ficheiros "objeto" (2)

- No caso de ser só em C:

f1.c:
```
extern int myF2( float a );

int main( ) {
      int x = myF2( 3.6 );
...
}
```

f2.c:
```
int myF2( float a ) {
      ...
}
```

cc -o f f1.c f2.c



Módulos em C

- Módulo f2 com *header*:

f1.c:
```
#include "f2.h"

int main(  ) {
      int x = myF2( 3.6 );
      ...
}
```

f2.h:
```
extern  int myF2( float a );
```

f2.c:
```
int myF2( float a ) {
      ... // implementação
}
```

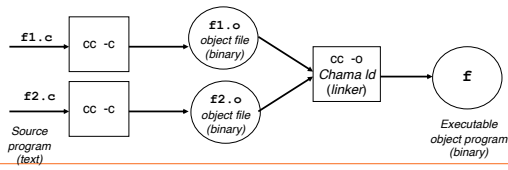
cc -o f f1.c f2.c

Compilação separada

- Podemos compilar em separado:

cc -c f2.c
cc -o f f1.c f2.o

cc -c f1.c
cc -c f2.c
cc -o f f1.o f2.o

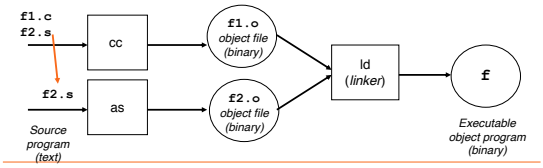


Compilação separada C + *assembly*

as -o f2.o f2.s
cc -o f f1.c f2.o

cc -c f1.c
as -o f2.o f2.s
cc -o f f1.o f2.o

ou ainda: **cc -o f f1.c f2.s**

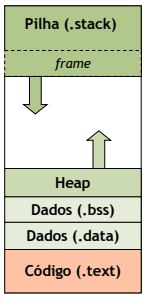


Zonas de dados nos programas

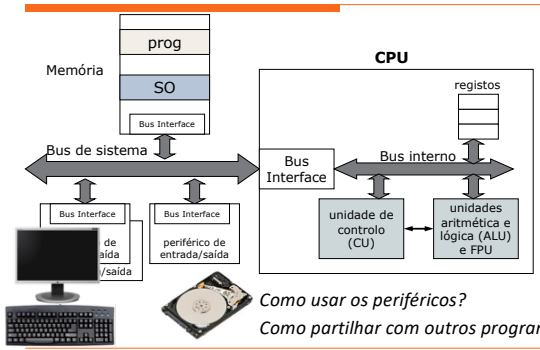
- Globais (estáticos)
  - Inicializados
  - Não inicializados
- Globais (dinâmicos)
  - Criados durante a execução
- Locais ao contexto de uma função/método
  - Parâmetros
  - Variáveis locais

Mapa de memória durante a execução

```
#include <stdio.h>

int x;
int y = 5;

int soma( int x, int y) {
   int z = x+y;
   return z;
}

int main( ){
   int a;
   int *b = malloc(sizeof(int));
   *b = 4;
   a = soma( a, *b );
   return 0;
}
```



Execução de um programa. E o IO?



Como usar os periféricos?
Como partilhar com outros programas?

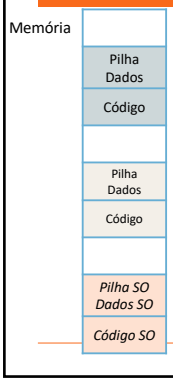
O que é o SO?

- O gestor de recursos
  - Gere e permite a partilha do *hardware* pelos programas
  - Gere os programas e suas interações
- Uma máquina virtual
  - Oferece abstrações de mais fácil utilização
  - Oferece operações de mais "alto nível" usadas pelo resto do sistema computacional
  - Define uma ABI
    - Exemplo: Linux, intel 32bits
  - Define uma API
    - Exemplo: POSIX, X/OPEN

Na pratica o que é?

- Um programa permanentemente carregado na memória central (RAM) – também chamado "kernel"
- A sua ação é complementada por
  - Programas de sistema: interpretador de comandos, carregadores, ligadores, serviços de impressão e rede...
- O SO é sempre carregado na RAM quando:
  - O computador é ligado (power up)
  - Quando se reinicia o hardware (reset)

Sistema multi-programado ou *multitasking*



- Em memória estão vários programas em execução
  - Podem existir múltiplas instâncias do mesmo programa
- Cada **processo** tem a ilusão de um sistema dedicado
  - Memória, periféricos e CPU

Um programa num Sist. Operação

- O SO gere tudo, incluindo pôr os programas em execução
- Num sistema multiprogramado, os programas não podem fazer tudo o que querem:
  - Não podem aceder à memória dos outros programas
  - Não podem alterar diretamente os periféricos (têm de os partilhar com os outros programas)
- Têm de recorrer a pedidos ao SO para esses tipos de operações, usando a API do SO
- Algumas instruções do CPU têm assim de ser vedadas aos programas "normais"
  - Exemplo: as que permitem aceder aos periféricos

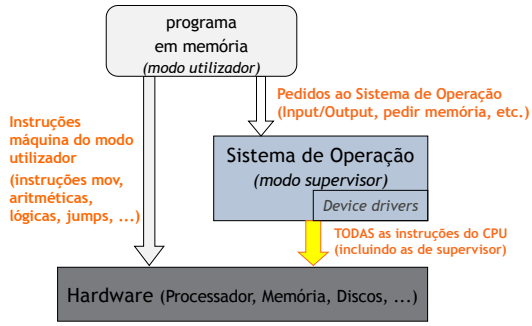
Dois modos de operação

- O CPU tem de suportar pelo menos dois modos de operação:
  1. *Modo utilizador* – CPU está a executar por conta de um programa normal. Nem todas as instruções estão disponíveis.
  2. *Modo supervisor* (*kernel mode* or *system mode*) – execução de código do SO. Todas as instruções disponíveis.

Ciclo de execução:

```
Fetch
Decode
if ( inst previlegiada && modo CPU != supervisor)
                interrupção por exepção
else Execute
```

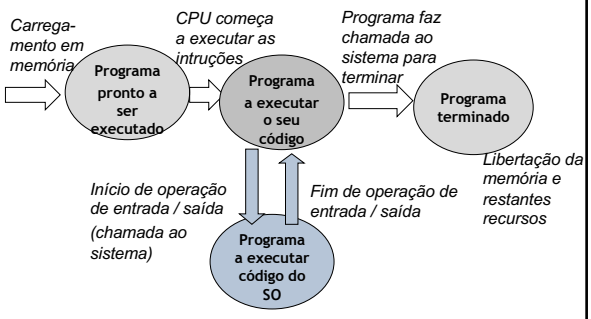

Um programa em execução



Exemplo de troca de modo do CPU

- **`Int`** (interrupt) é uma das instruções que muda o modo de funcionamento (utilizador → supervisor)
  - Semelhante ao **`call`**, mas subrotina identificada por um índice numa tabela e mudando o CPU para modo supervisor
- Um programa em modo utilizador nunca consegue passar a modo supervisor
  - O SO é o primeiro a executar e inicia a execução dos programas sempre em modo utilizador
  - Cada "int nº" chama código definido pelo SO
  - Cada subrotina termina sempre repondo o CPU em modo utilizador (**`iret`**)

Programa em execução: Processo

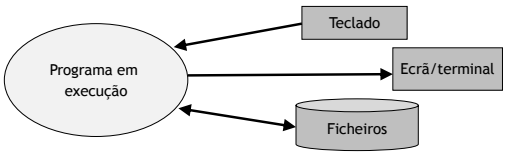


Helloworld.s

```
EXIT = 1
WRITE = 4
LINUX_SYSCALL = 0x80
.data           # secção de dados (variáeis)
msg:   .ascii    "Hello, world!\n"  # um vetor de caracteres
len = . - msg    # len representa o tamanho do vetor
.text           # secção de código
.global _start   # exportar o simbolo _start (inicio do programa)
_start:  movl    $len,%edx      # num de bytes a escrever
         movl    $msg,%ecx      # end dos bytes a escrever
         movl    $1,%ebx        # canal onde escrever (1=stdout)
         movl    $WRITE,%eax    # pedir write ao sistema
         int     $LINUX_SYSCALL         # chama o sistema

         movl    $0,%ebx          # codigo de terminacao
         movl    $EXIT,%eax       # pedir o exit ao sistema
         int     $LINUX_SYSCALL         # chama o sistema
```

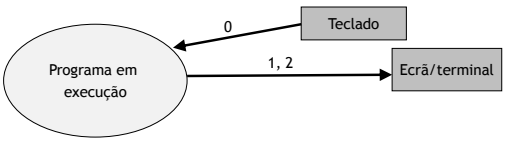

Canais de entrada/saída

- Um programa em execução não acede diretamente aos periféricos
- Usa ficheiros através de canais (*streams*) de entrada/saída
- Esses canais permitem ler e escrever sequências de bytes de forma fiável e mantendo a ordem



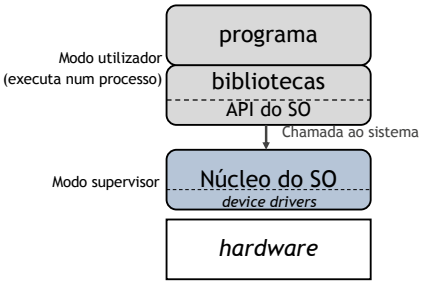
Canais de entrada/saída

- Os canais no SO são identificados por números
  - Começam em 0 e vão crescendo à medida das necessidades
  - Os canais 0 (stdin), 1 (stdout) e 2 (stderr) são abertos pelo SO quando o programa começa a executar
  - O processo pode pedir ao SO mais canais, p.e. para aceder a ficheiros



Chamadas entre níveis de software

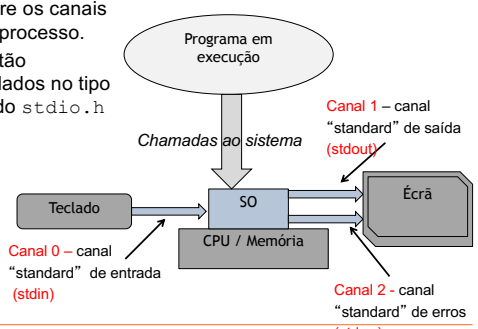
Os programadores usam a API da linguagem



Processo e IO - Unix e C

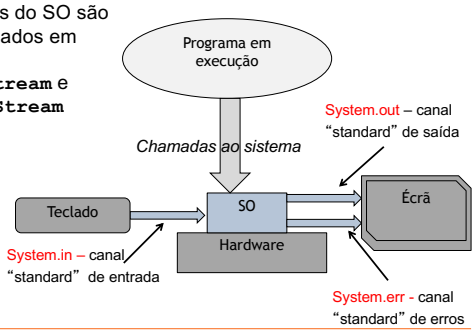
O SO gere os canais de cada processo.

Em C estão encapsulados no tipo **`FILE*`** do `stdio.h`



Processo e IO - Java

Os canais do SO são encapsulados em objectos **`InputStream`** e **`OutputStream`**



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()