NOVA FACULDADE DE CIÊNCIAS E TECNOLOGIA DEPARTAMENTO DE INFORMÁTICA

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
Mª. Cecília Gomes

1

Aula 13

- Partilha e mapeamento de ficheiros
- Bibliotecas dinâmicas partilhadas

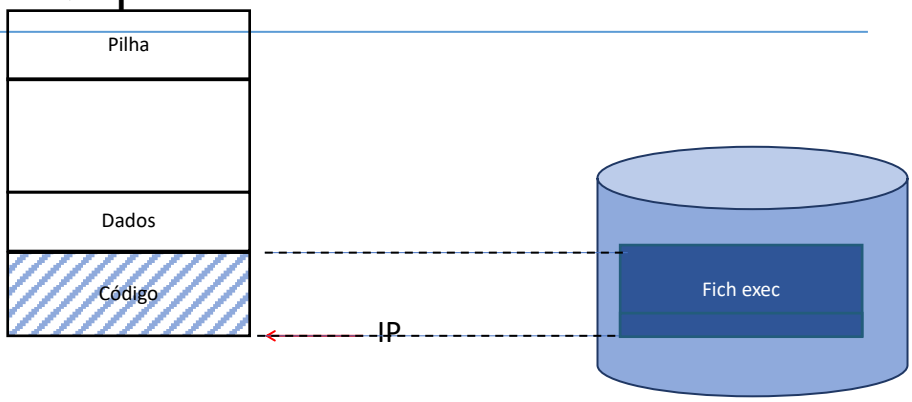
- OSTEP: cap. 14.5, 17, 39.Aside (pág. 13), F.4
- Silberschatz, Operating Systems Concepts, 10th Ed. cap. 2.5, 9.1.5, 3.7.1, 13.5.1
- *UNIX Network Programming, Volume 2, Richard Stevens,*

cap 12:
http://www.kohala.com/start/unpv22e/unpv22e.chap12.pdf

NOVA FACULDADE DE CIÊNCIAS E TECNOLOGIA DEPARTAMENTO DE INFORMÁTICA

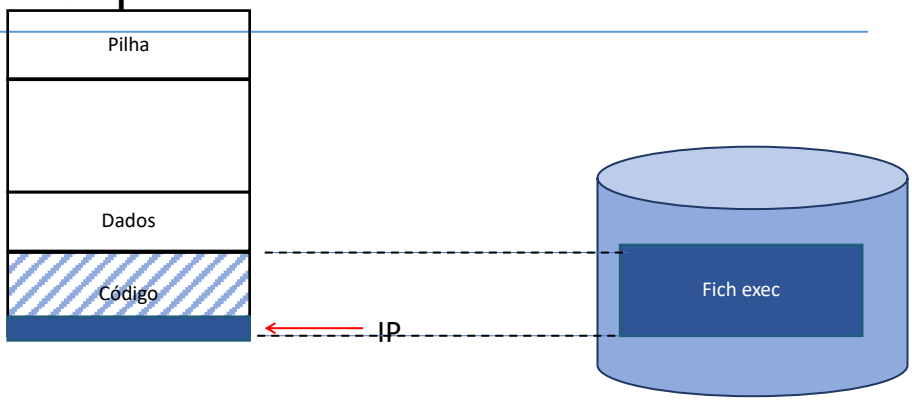
2

Mapeamento de ficheiro em memória



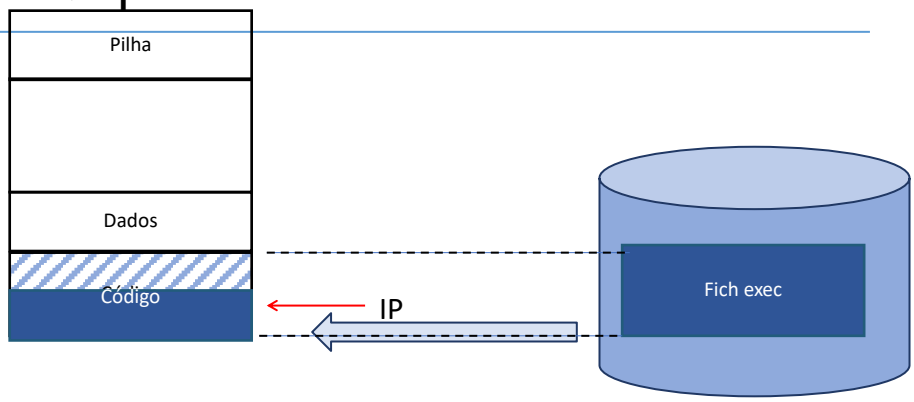
De início de um novo programa, a tabela só tem paginas marcadas como não presentes, mas baseadas no fich. executável.

Mapeamento de ficheiro em memória



Parte do espaço do processo corresponde aos bytes no ficheiro. Vão sendo trazidos para memória a pedido ("on-demand")

Mapeamento de ficheiro em memória



Parte do espaço do processo corresponde aos bytes no ficheiro. Vão sendo trazidos para memória a pedido ("on-demand")

Memória virtual com paginação a pedido

- Permite **memória virtual** de dimensão arbitrária:
  - Usa o disco para "estender" a memória real
  - Gere a memória real como uma **cache** para todas as imagens dos processos
- Memória virtual (VM) – separação da memória lógica (virtual) da memória física
  - MV definida pelo espaço de endereçamento lógico (ou virtual)
  - Só parte da imagem do processo precisa de estar realmente em memória (o seu **working set**)
  - Um processo pode ter mapa de memória **>** memória física
  - É possível ter memória física **<** soma das imagens de todos os processos
  - Todas as páginas dos processos podem vir a ser guardadas em memória secundária, (tipicamente chamado disco/ficheiro de paginação ou de swap) ou recuperadas do ficheiro de origem (eg. executável)
  - As páginas em swap só voltam para memória central quando são novamente referenciadas

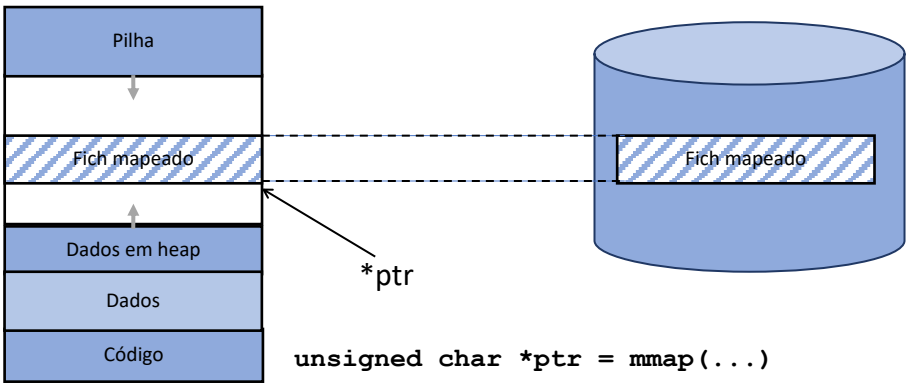
"Memory mapping" de ficheiros

- O SO mantém no mapa de memória de cada processo a descrição de todo o espaço de endereçamento atribuído e o suporte secundário do seu conteúdo (ficheiros ou swap)
- O mapeamento de ficheiros para memória pode ser uma nova funcionalidade, para uso do programador
- Uma chamada ao sistema permite pedir um novo mapeamento
- Exemplo UNIX:

```
void *mmap(void *start, int len,
      int prot, int flags, int fd, int offset)
```

- Retorna um apontador para o início da região.
  - O apontador é sempre para o início de uma página (tipicamente múltiplo de 4096) no espaço de endereçamento virtual do processo
  - Fica associada ao ficheiro com descritor *fd* a partir da posição *offset*.

7




```
unsigned char *ptr = mmap(...)
```

primeiro byte apontado = byte no offset do ficheiro

ptr[100] = byte offset+100 do ficheiro

etc

8

Ficheiros "Memory-Mapped"

- Permite que as operações read/write sejam substituídas por acessos à memória.
  - A cada bloco do ficheiro corresponde uma página em RAM.
- O ficheiro é trazido para memória por paginação a pedido.
  - O 1º acesso a cada bloco dá *page-fault* e o SO trá-lo para uma *frame*; os acesso seguintes são acessos normais.
- O acesso aos ficheiros pode ser aleatório e por operações sobre memória (ex.: memcpy).
  - IO transparente

Exemplo: ler ficheiro com mmap()

```
struct stat stat;
int fd, size;
char *ptr;

fd=open("./fich.txt", O_RDONLY);
fstat(fd, &stat);
size = stat.st_size;
/* map all the file to a new VM area */
ptr = mmap(NULL, size, PROT_READ,
           MAP_PRIVATE, fd, 0);
/* write all the mapped area to stdout */
write(1, ptr, size);
```

Ficheiros "Memory-Mapped" (2)

- Vários processos podem incluir o mesmo ficheiro no seu espaço de endereçamento
  - SO gere a partilha das mesmas páginas físicas (frames).
- As proteções associadas no mmap permitem partilha só em leitura ou leitura escrita, etc
  - Consequência de partilhar em leitura/escrita?
- Permite criar zonas de memória partilhada entre processos
- Usado intensivamente para suporte de "*shared libraries*" carregadas dinamicamente
  - "Automaticamente" mapeadas e partilhadas em modo de leitura pelo *linker*
- Sem ficheiro (ou com /dev/zero), permite pedir mais memória ao SO

Partilha de memória (*shared memory*)

- Os processos podem pedir ao SO regiões de memória partilhada
- Podem ser anónimas ou com nome
- Duas interfaces:
  - Sys V: shmget, shmat
  - Posix: shm_open, mmap (permite também mapear ficheiros em memória)

Shared memory Posix

- Criar ou aceder a uma zona de memória partilhada **com nome**:

```
int shm_open(name,flags, ...)
```

  - Semelhante à criação/abertura de ficheiros
  - Cria/pede acesso a memória partilhada com *name*
  - Devolve descritor (fd) para usar no mmap
- Mapear no espaço do processo:

```
void *mmap(void *start, len, prot,
           flags, fd, offset)
```

  - *flags* deve incluir MAP_SHARED para partilhar entre processos
  - pode incluir MAP_ANONYMOUS (não usa *shm_open*)
  - fd pode ser fechado após o mmap

(relembre)Comunicação pai/filho c/ *pipe*

- O pai cria o *pipe*; o filho partilha-o; o SO garante a sincronização:

```
if (pipe(p)==-1) abort();
switch(fork())
{ case -1: abort();
  case  0: // filho
           write(p[1],"ola",4);
           // etc....
           exit(0);
  default: // pai
           read(p[0], buff, SZ);
           // ...
}
```

Partilha de memória entre pai e filho

```
sem_t *ex = sem_open("mysem", O_CREAT, 0700, 0);
sem_unlink("mysem");

char *src = mmap(NULL, LNSIZE, PROT_READ|PROT_WRITE,
                 MAP_ANONYMOUS|MAP_SHARED,-1,0);
if (src == MAP_FAILED) {
       perror("mmap");
       exit(1);
}
switch ( fork() ) {
  case -1: perror("fork");
           break;
  case 0:  strcpy( src, "Hello!!");
           sem_post(ex);
           exit(0);
  default:
           sem_wait(ex);
           printf("%s\n", src );
}
```


Vantagens do "Memory mapping" de ficheiros

- Unifica o acesso aos dados de um programa
  - Passa a usar acessos (mesmo aleatórios) a memória. O programador é libertado da tarefa de movimentar dados entre o disco e RAM.
- Exige menos chamadas ao sistema.
- As transferências podem ser mais rápidas
  - O kernel faz operações de IO directamente entre a memória do processo e o disco sem passar pelos "buffers" do sistema.
- Dá oportunidade do SO partilhar os frames entre diferentes processos
  - Permite partilha entre processos nos modos SHARED e READ
  - Partilha com Copy-on-Write no modo PRIVATE com WRITE

Desvantagens do "Memory mapping" de ficheiros

- Pode aumentar fragmentação interna
  - Mapeamentos em múltiplos das páginas
- Ficheiros grandes devem (ou só podem) ser parcialmente mapeados
  - Usa o espaço virtual de endereçamento
- O mmap e munmap têm custos (espaço e tempo) e o IO da paginação a pedido pode não ser o melhor para o padrão de acessos ao ficheiro
  - pedir logo um bloco grande do ficheiro numa única chamada pode ser melhor que deixar a paginação ir lendo uma página de cada vez;
  - escrever num ficheiro uma sequencia de dados pode ser mais fácil e eficiente do que fazer o ficheiro mapeado em memória crescer.

Ficheiros "Memory-Mapped" (2)

- Vários processos podem incluir o mesmo ficheiro no seu espaço de endereçamento, partilhando as mesmas páginas físicas.
- As proteções associadas no mmap permitem partilha só em leitura ou leitura escrita, etc
- Usado intensivamente para suporte de "shared libraries" carregadas dinamicamente

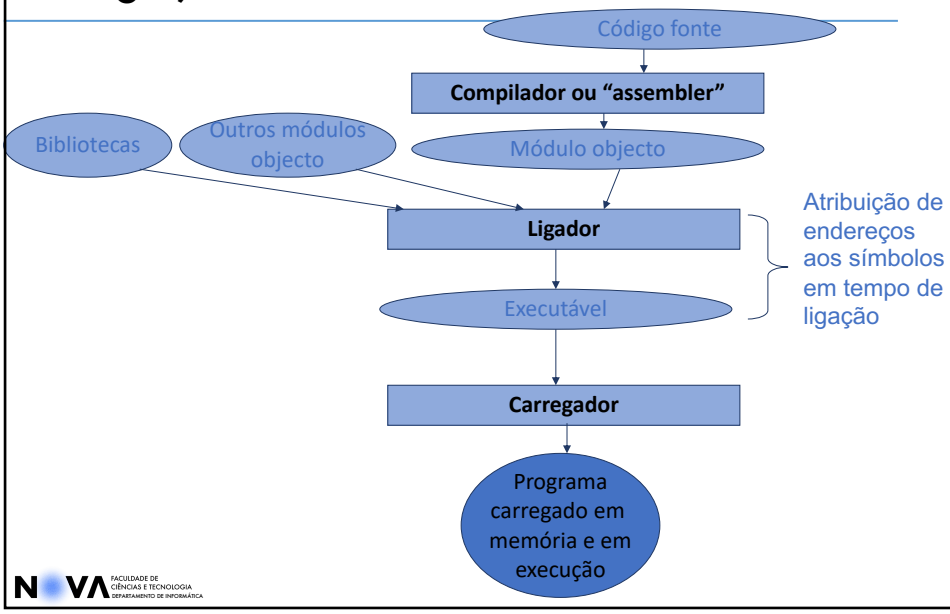
Uso do mmap files pelas linguagens

- Vários processos podem incluir a mesma biblioteca no seu espaço de endereçamento (ex. libC)
- mmap usado intensivamente para suporte de "*shared libraries*" e carregadas dinamicamente
- As ferramentas de desenvolvimento e run-time das linguagens tiram partido deste mecanismo
  - Permitem a partilha de código
  - Permitem o carregamento dinâmico, a pedido, durante a execução, de código ou dados.
- Exemplo:
  - As bibliotecas no sistema costumam existir em duas versões: estáticas e dinâmicas partilháveis
    - Unix: *shared libraries*
    - MS-Windows: *dynamic link libraries (DLL)*

Ligação estática

- A ligação estática de bibliotecas é feita pelo *linker* para obter o executável completo
  - todo o código e dados está no executável
- Desvantagens:
  - Muito código comum nos ficheiros executáveis guardados em disco e, depois, na memória durante a execução
    - e.g., cada programa em C necessita da standard C library
  - Pequenas modificações nas bibliotecas obrigam a voltar a ligar todas as aplicações

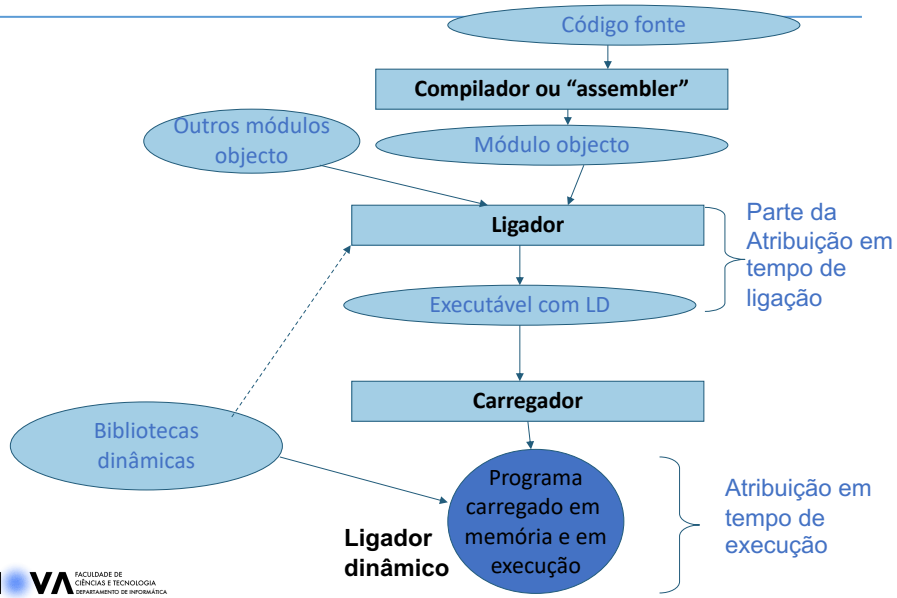
Ligação estática



Ligação dinâmica

- O *linker* confirma a possibilidade de resolver todos os símbolos e suas bibliotecas.
- O código das bibliotecas não é inserido no ficheiro executável. Mas inclui código de ligação dinâmica.
- A ligação apenas é terminada durante a execução
  - Durate a execução o linker dinâmico vai resolver os símbolos à primeira referência e carregar as bibliotecas

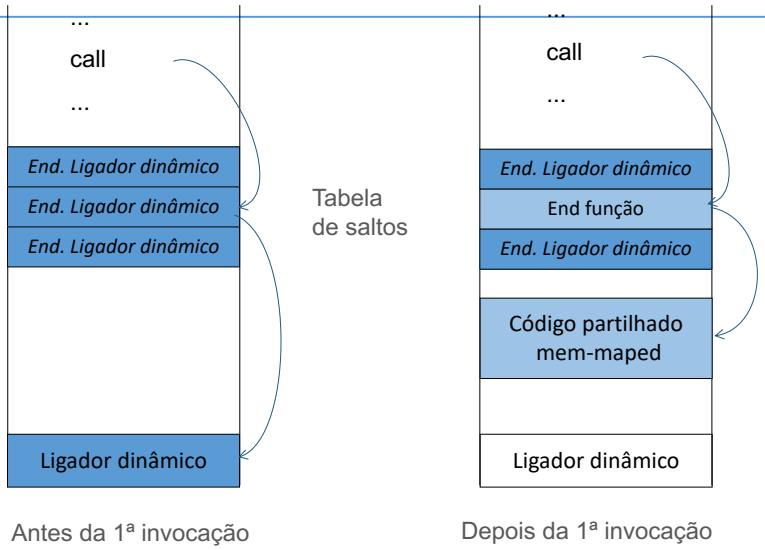
Ligação dinâmica



Ligação dinâmica(*dynamic linking*)

- Ligação é adiada até ao momento da execução.
- As chamadas são indiretas via uma tabela de saltos que, de início, refere o *linker* dinâmico
- Este identifica a biblioteca (o ficheiro) e mapeia-o para memória, substituindo na tabela a referência pelo respectivo endereço
- O seu carregamento será por paginação a pedido
- Alterações às bibliotecas, que não introduzam incompatibilidades, podem ser efetuadas e todos os programas usam as novas versões sem serem "religados"

Ligação dinâmica

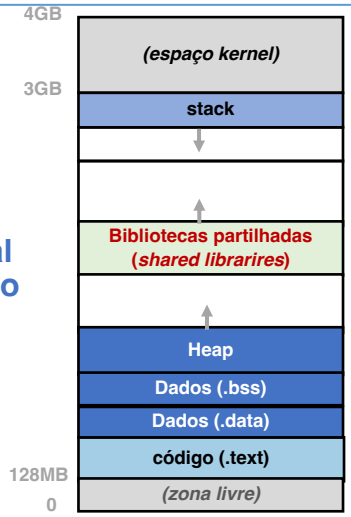


Bibliotecas partilhadas (*Shared libraries*)

- As bibliotecas mapeadas dinamicamente para memória são partilhadas (SO partilha as *frames*)
  - O código é só de leitura; os dados da biblioteca são copiados em *Copy-on-Write*
  - No disco só existe uma cópia das bibliotecas (os executáveis só têm indicação das bibliotecas usadas)
  - À primeira referência o ficheiro (biblioteca) começa a ser carregado para memória e todos os processos partilham as páginas já carregadas

Imagem de um processo no Linux/32bits

Imagem da memória virtual de um processo Linux/x86



Dependendo da versão ou se é 32 bits vs 64 bits, pode ser diferente.

NOVA FACULDADE DE CIÊNCIAS E TECNOLOGIA DEPARTAMENTO DE INFORMÁTICA

27