PROCEDIMENTOS EM MEMÓRIA EXTERNA

Saulo Queiroz, UTFPR-PG.

Questão

- Quanto tempo leva para lermos um dado se dispomos do **ponteiro** para esse dado?
  -
  -

Questão

- Quanto tempo leva para lermos um dado se dispomos do **ponteiro** para esse dado?
  - Rigorosamente falando, isso depende da tecnologia do meio de armazenamento!
  - Comumente, algoritmos em EDs assumem dados armazenados em memória RAM!

Memória RAM (“interna” ou “principal”)

- Acesso aleatório
  - Uma unidade **qualqer** (e.g. byte) da memória pode ser diretamente acessada se dispusermos de seu endereço (i.e. ponteiro)
    - Para acessar o byte `0xff` não precisamos acessar seus antecessores!
-
  -
  -

Memória RAM (“interna” ou “principal”)

- Acesso aleatório
  - Uma unidade **qualqer** (e.g. byte) da memória pode ser diretamente acessada se dispusermos de seu endereço (i.e. ponteiro)
    - Para acessar o byte `0xff` não precisamos acessar seus antecessores!
- Desempenho do Acesso aleatório na RAM
  - Tempo para acessar qualquer byte da memória a partir de ponteiro é aproximadamente o mesmo!
  - Ler o dado armazenado em `0x0` é tão rápido quanto ler aquele em `0xffffffff`

Memória RAM

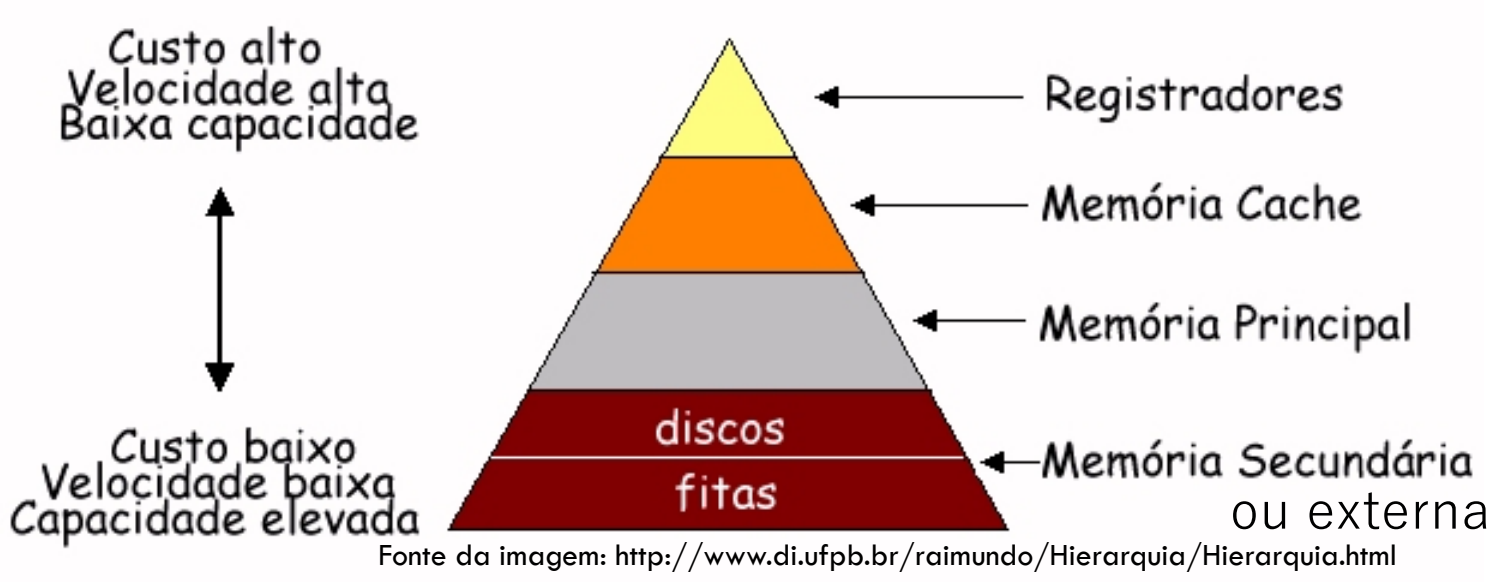
- Os comandos de alocação estática (declaração de variáveis) e dinâmica (`malloc`) referem-se à memória RAM
- Em geral, um dado "passa pela RAM" antes de chegar à CPU

Memória RAM

- Os comandos de alocação estática (declaração de variáveis, alocação (malloc)) alocam na memória RAM
- Em geral, um dado "passa pela RAM" antes de chegar à CPU

Onde mais posso armazenar meus dados?

Hierarquia de Memória

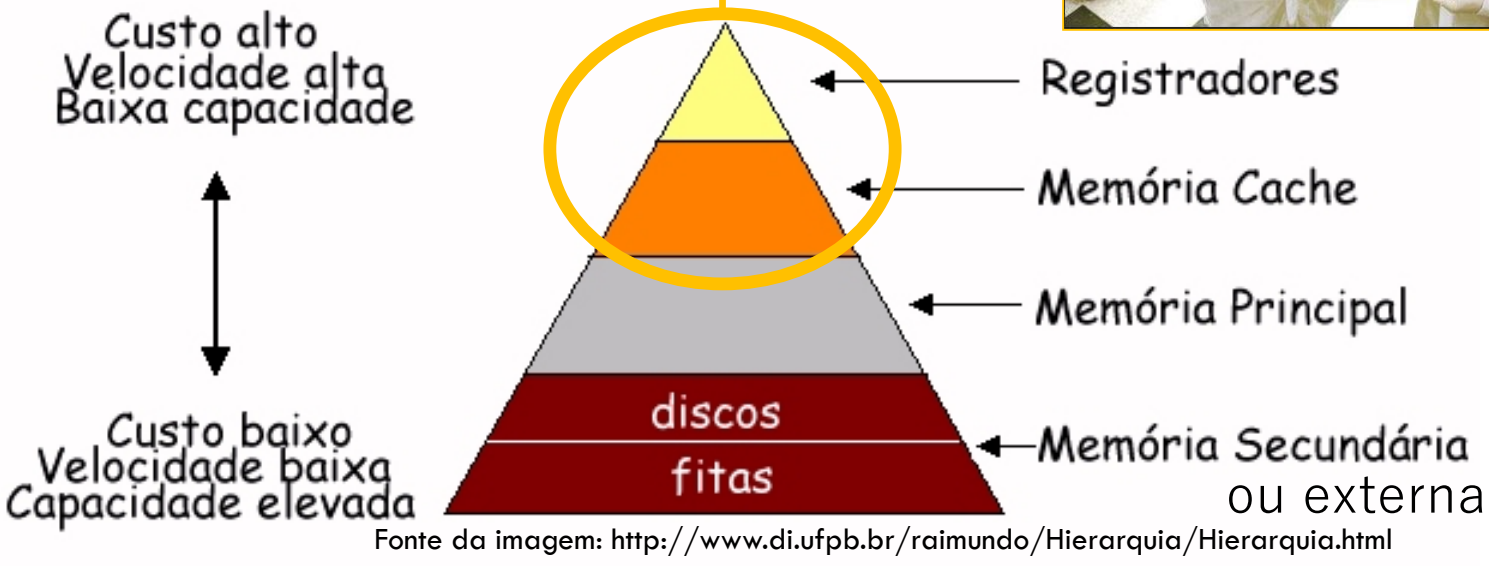


Fonte da imagem: http://www.di.ufpb.br/raimundo/Hierarquia/Hierarquia.html

Hierarquia de Memória

Mesma matéria prima do processador.

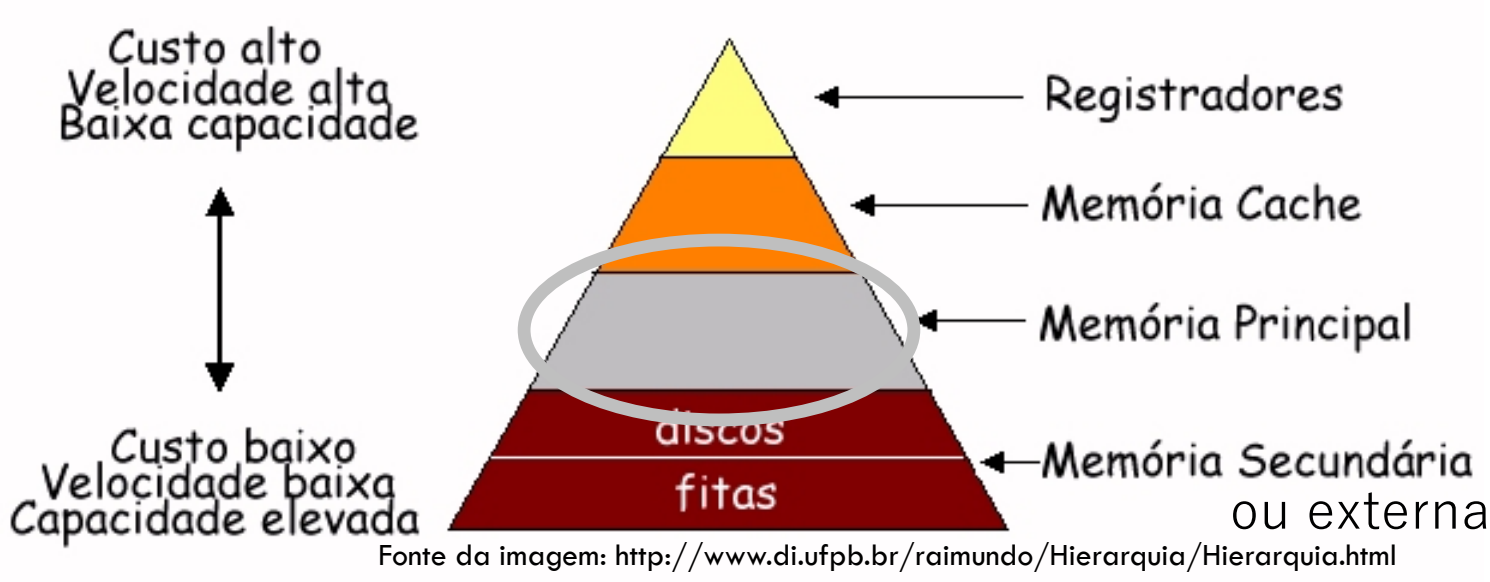
Extremamente rápido e caro!



Fonte da imagem: http://www.di.ufpb.br/raimundo/Hierarquia/Hierarquia.html

Hierarquia de Memória

"Meio termo" entre custo e desempenho!



Fonte da imagem: http://www.di.ufpb.br/raimundo/Hierarquia/Hierarquia.html

Hierarquia de Memória

Muito lento (tecnologia mecânica), muito barato! Tempo de acesso pode ser 6 ordens de magnitude maior que na RAM!

Fonte da imagem: http://www.di.ufpb.br/raimundo/Hierarquia/Hierarquia.html

Acesso Aleatório ou sequencial

O acesso à memória externa é aleatório?

Acesso Aleatório ou sequencial

- Cada dispositivo tem sua própria tecnologia: uns com outros sem acesso direto!
  - Ex.: Fitas magnéticas tem acesso sequencial (não aleatório)
-
  -

Acesso Aleatório ou sequencial

- Cada dispositivo tem sua própria tecnologia: uns com outros sem acesso direto!
  - Ex.: Fitas magnéticas tem acesso sequencial (não aleatório)
- Consideraremos "memória externa" sinônimo de discos magnéticos (HD) em que o acesso, apesar de ser aleatório, é **muito mais lento que na memória RAM.**
  - Note que: Os cuidados de programação em memória externa depende do dispositivo considerado

Memória Externa: Item de desempenho

- Cada dispositivo tem sua própria tecnologia: uns com acesso direto e outros com acesso sequencial
  - Ex.: Fitas magnéticas tem acesso sequencial (não aleatório)
- Comparando a memória externa com a memória RAM, os discos magnéticos (HD) em que o acesso, apesar de ser aleatório, é muito mais lento que na memória RAM.
- Note que: Os cuidados de programação em memória externa depende do dispositivo considerado

O principal item de desempenho de algoritmos que manipulam memória externa é o total de acessos a disco!

Memória Secundária: Peculiaridades

- Para compensar o “lento” acesso, HDs têm uma unidade de armazenamento/transferência maior que a da RAM
  - Lembre: a unidade da RAM é 1 Byte.
-
-

Memória Secundária: Peculiaridades

- Para compensar o "lento" acesso, HDs têm uma unidade de armazenamento/transferência maior que a da RAM
  - Lembre: a unidade da RAM é 1 Byte.
- A unidade mínima para leitura/escrita *física* (i.e., em *hardware*) oferecida por um HD é o **setor**
-

Memória Secundária: Peculiaridades

- Para compensar o "lento" acesso, HDs têm uma unidade de armazenamento/transferência maior que a da RAM
  - Lembre: a unidade da RAM é 1 Byte.
- A unidade mínima para leitura/escrita *física* (i.e., em *hardware*) oferecida por um HD é o **setor**
- No nivel lógico (i.e., *software*) a terminologia da unidade mínima é vaga, podendo variar de um SO para outro

Unidade de Lógica (Shaffer, 13)

- No Windows (FAT, NTFS)
  - "*A cluster is the smallest unit of allocation for a file, so all files are a multiple of the cluster size. The cluster size is determined by the operating system*"
- No Linux (ext4, reiserfs, etc)
  - "*In contrast, UNIX does not use clusters. The smallest unit of file allocation and the smallest unit that can be read/written is a **sector**, which in UNIX terminology is called a **block**.*"

Tamanho do Bloco no Unix


```
user@linux> sudo fdisk -l
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disklabel type: dos
Disk identifier: 0x00021caa
```

Nesse caso o S.O. organiza uma leitura física do HD em 8 blocos lógicos. Por que?

Tamanho do Bloco no Unix

512 B era a unidade de HDs antigos. Vários softwares adotaram isso como unidade lógica no Linux.

Nos HDs sob o padrão subsequente ("*advanced format*"), o setor é maior que 512 B (geralmente 4096 B)

Arquivos

- A gestão dos dados sobre o HD resulta do
  - Sistema de arquivos do SO
  - Funcionamento do HD
- Podemos usar o HD em nossos programas por meio de **arquivos**

Noções de Arquivos em C: stdio

Prof. Saulo Queiroz

Arquivos

- Definição clássica
  - *Recurso para armazenamento de informações* (Wikipedia)
  - São estruturas dinâmicas: crescem/diminuem sob demanda
-
  -
-
  -

Arquivos

- Definição clássica
  - *Recurso para armazenamento de informações* (Wikipedia)
  - São estruturas dinâmicas: crescem/diminuem sob demanda
- Podem ser acessados por diferentes programas
  - Na **memória primária** (RAM) o **espaço de endereços** é dedicado por processo
-
  -

Arquivos

- Definição clássica
  - *Recurso para armazenamento de informações* (Wikipedia)
  - São estruturas dinâmicas: crescem/diminuem sob demanda
- Podem ser acessados por diferentes programas
  - Na **memória primária** (RAM) o **espaço de endereços** é dedicado por processo
- Manipulamos arquivos com base em procedimentos que realizam chamadas ao sistema
  - Biblioteca `stdio.h` (C, C++), `fstream` (C++).

Arquivos no C

- O arquivo `stdio.h` define `FILE`, uma estrutura de dados para fluxo de arquivos contendo (dentre outros elementos)
  - Um indicador de posição de leitura/escrita (número de byte)
  - Um sinalizador de erro de acesso ao arquivo
  - Um sinalizador de fim de arquivo (não há mais informações a ler)
-
  -
  -
  -

Arquivos no C

- O arquivo `stdio.h` define `FILE`, uma estrutura de dados para fluxo de arquivos contendo (dentre outros elementos)
  - Um indicador de posição de leitura/escrita (número de byte)
  - Um sinalizador de erro de acesso ao arquivo
  - Um sinalizador de fim de arquivo (não há mais informações a ler)
- Etapas básicas de acesso a um arquivo
  - 1. Abertura do arquivo
  - 2. Leitura/escrita
  - 3. Fechamento do arquivo

`stdio.h` Definições para arquivos

- Declaração básica
  - O programador manipula arquivos por meio de uma variável ponteiro para arquivo do tipo `FILE *`
  - Ex.: `FILE *pArq1;//pArq1 representa um arquivo`
- Passo após a declaração: abertura

Abertura de Arquivos

- `FILE *fopen(char *NomeDoArquivo, char *ModoDeAcesso);`
  - `NomeDoArquivo` *string* com o nome tal como ele aparece ao usuário no sistema operacional
  - `ModoDeAcesso` *string* indicando modo de acesso

Abertura de Arquivos

- `FILE *fopen(char *NomeDoArquivo, char *ModoDeAcesso);`
  - `NomeDoArquivo` *string* com o nome tal como ele aparece ao usuário no sistema operacional
  - `ModoDeAcesso` *string* indicando modo de acesso
- Tipos de fluxos que podem ser manipulados
  - **Texto:** a sequência de *bytes* é interpretada como sequência de zero ou mais caracteres alfa-numéricos
  - **"Binário":** sequência de bytes tal como armazenado em memória
    - Abuso de linguagem, já que tudo em um PC é binário

Códigos para abrir arquivos de **texto**

Code	Tipo	Finalidade	Requisito	OBS.
r	Texto	Leitura	Arquivo **deve** existir no diretório indicado	fopen devolve NULL caso arquivo não exista
w	Texto	Escrita	Arquivo **não precisa** existir no diretório	Cria arquivo ou sobrescreve caso existe um com mesmo nome
r+	Texto	Leitura e escrita	Arquivo **deve** existir no diretório indicado	fopen devolve NULL caso arquivo não exista
w+	Texto	Leitura e escrita	Arquivo **não precisa** existir no diretório	Cria arquivo ou sobrescreve caso existe um com mesmo nome
a	Texto	Escrita	*Mesmo do anterior*	*Grava no fim do arquivo se existir*
a+	Texto	Leitura e escrita	*Mesmo do anterior*	*Mesmo do anterior*

Abertura de arquivo “binário”

- Para fazer o C trabalhar com arquivos “binários”, basta acrescentar b **ao final** dos códigos anteriores

Códigos para abrir arquivos de **texto**

Code	Tipo	Finalidade	Requisito	OBS.
rb	Bin.	Leitura	Arquivo **deve** existir no diretório indicado	fopen devolve NULL caso arquivo não exista
wb	Bin.	Escrita	Arquivo **não precisa** existir no diretório	Cria arquivo ou sobrescreve caso existe um com mesmo nome
r+b	Bin.	Leitura e escrita	Arquivo **deve** existir no diretório indicado	fopen devolve NULL caso arquivo não exista
w+b	Bin.	Leitura e escrita	Arquivo **não precisa** existir no diretório	Cria arquivo ou sobrescreve caso existe um com mesmo nome
ab	Bin.	Escrita	*Mesmo do anterior*	*Grava no fim do arquivo se existir*
a+b	Bin.	Leitura e escrita	*Mesmo do anterior*	*Mesmo do anterior*

Exemplo fopen: Abrindo arquivo local


```
#include <stdio.h>
void main()
{
  FILE *pArquivo = fopen(“dados.txt”, “w+”);
  if (pArquivo == NULL)
    printf(“Não conseguiu criar/abrir”);
}
```

Exemplo fclose: Fechando arquivo local


```
#include <stdio.h>
void main()
{
  FILE *pArquivo = fopen(“dados.txt”, “w+”);
  if (pArquivo == NULL)
    printf(“Não conseguiu criar/abrir”);
  fclose(pArquivo); //impede escritas equivocadas
}
```

Lendo um único caracter do arquivo

- `char getc(FILE *)`
  - Lê e devolve um caracter do arquivo indicado
  - A leitura é feita a partir da posição atual do cabeçote
  - A leitura implica em avançar o cabeçote de leitura adiante no arquivo
  - Devolução pode ser recebida por um `char` ou inteiro
-
  -
  -

Lendo um único caracter do arquivo

- `char getc(FILE *)`
  - Lê e devolve um caracter do arquivo indicado
  - A leitura é feita a partir da posição atual do cabeçote
  - A leitura implica em avançar o cabeçote de leitura adiante no arquivo
  - Devolução pode ser recebida por um `char` ou inteiro
- `feof (FILE *)`
  - Devolve valor lógico *true* (não zero) se o caracter lido mais recentemente foi o EOF ou *false* (zero) caso contrário
    - EOF: Significa `End Of File`

getc: Exemplo


```
#include <stdio.h>
void main()
{
  FILE *pArquivo = fopen(“dados.txt”, “w+”);
  if (pArquivo == NULL)
   printf(“Não conseguiu criar/abrir”);
  char c = getc(pArquivo);
  if (c == EOF) printf(“Arquivo chegou ao fim!”)
  fclose(pArquivo); //impede escritas equivocadas
}
```

Escrevendo um caracter no arquivo

- `int putc(char,FILE *)`
  - Escreve um caracter no arquivo
  - A escrita é feita a partir da posição atual do cabeçote
  - Avança o cabeçote do disco em 1 byte
  - Devolução (pode ser recebida por um `char` ou inteiro)
    - caracter escrito, em caso de **sucesso**
    - EOF em caso de **insucesso**

putc: Exemplo


```
#include <stdio.h>
void main()
{
  FILE *pArquivo = fopen(“dados.txt”, “w+”);
  if (pArquivo == NULL)
   printf(“Não conseguiu criar/abrir”);
  char c = ‘a’;
  putc(c, pArquivo);
  fclose(pArquivo); //impede escritas equivocadas
}
```

Exercício

- Faça um programa que leia e imprima na tela caracter a caracter de um arquivo chamado "`big.txt`".
  - Use o arquivo de texto: `http://norvig.com/big.txt`

Leitura/impressão caracter-a-caracter


```
#include <stdio.h>
int main()
{
  char letra;
  FILE *p = fopen("big.txt", "r");
  if (!p) return -1;
  while( (letra=getc(p)) != EOF)
    printf("%c", letra);
  fclose(p);
  return 0;
}
```

Exercício

- Critique o algoritmo anteriormente solicitado considerando as seguintes questões:
  - Quantos acessos a disco são feitos?
  - É possível diminuir o número de acesso a discos? Como?
    - Dica: Quanto espaço um arquivo com 1 caracter consome?

Crítica ao exercício anterior

- Comando shell: `du -hs arq1letra.txt`
  - Saída: `4,0K  xx.c`
  - Embora só haja 1 caracter, o tamanho mínimo é o do setor!
- Conclusão?
  -
  -

Crítica ao exercício anterior

- Comando shell: `du -hs arq1letra.txt`
  - Saída: `4,0K  xx.c`
  - Embora só haja 1 caracter, o tamanho mínimo é o do setor!
- Conclusão?
  - O algoritmo é ineficiente pois sub-utiliza a unidade de leitura do sistema i.e. 4096 Bytes!
  - **Como melhorar?**

Crítica ao exercício anterior

- Comando shell: du -hs arq1letra.txt
- Saída do Código:
- Embora só haja 1 caracter, o tamanho mínimo é o do setor!
- Conclusão:
- O algoritmo é ineficiente pois subutiliza a unidade de leitura do sistema i.e. 4096 Bytes!
- Como melhorar?

Carregue na memória o máximo de dados de uma leitura do disco, i.e., um bloco inteiro!

Leitura em blocos com fread

- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream) //stdio.h`
  - `ptr` endereço da RAM onde dados do hd serão gravados
  - `size` tamanho unitário do dado em bytes e.g. `sizeof(char)`
  - `nmemb` qtd. de unidades de tamanho `size` a serem lidas do hd e.g. `char`
  - `stream` ponteiro para o arquivo

Leitura em blocos com `fread`

- Saída: Quantidade de itens lidos com sucesso
  - Se diferente da qtd. solicitada `nmemb`, então:
    - Ocorreu um erro ou
    - O fim do arquivo chegou

Versão melhorada (trecho)


```
char letra[4097]; //setores de 4096 B + 1 B para o ‘\0’
size_t nread; //para o total de bytes devolvidos pelo fread
while(!feof(p))
  {
    nread = fread(letra, sizeof(char), 4096, p);
    letra[nread] ='\0'; //fread não fecha string. Fazemos nós
    printf("%s",letra);
  }
  fclose(p);
```

Comparação dos dois programas

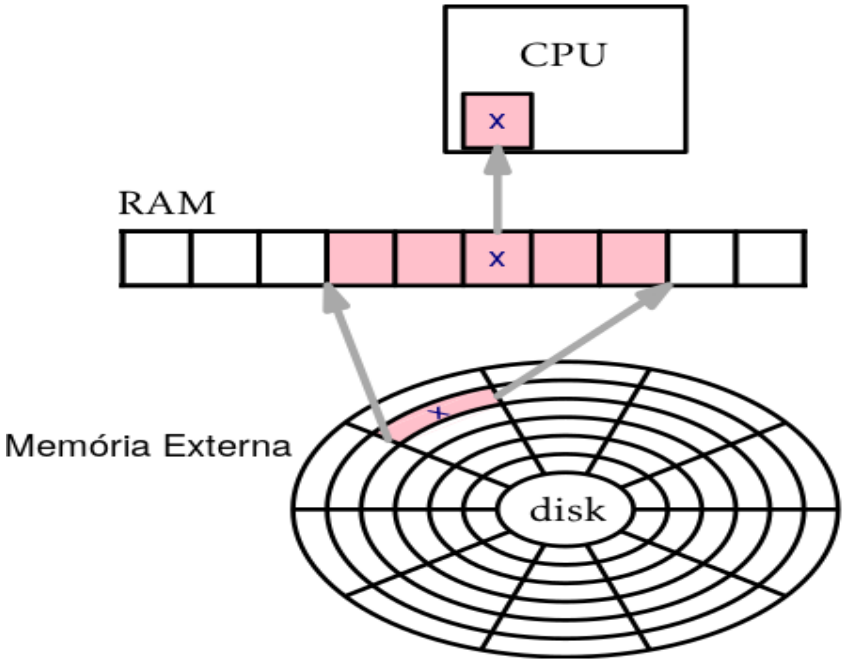
- time ./io1.c
  - real 0m1.204s
  - user 0m0.264s
  - sys 0m0.204s
- time ./io2.c
  - real 0m0.689s
  - user 0m0.012s
  - sys 0m0.112s

Que lições aprendemos sobre programar eficientemente em memória externa (disco)?

Prof. Saulo Queiroz

Acesso a Item na mem. externa

- Transferir um **item** x da mem. externa (e.g. variável `struct`) para a RAM implica ler todo o **bloco** que o contêm
  - Mesma ideia para escrever



Fonte da imagem: http://opendatastructures.org/ods-cpp/14_External_Memory_Searchin.html

Lições para Programar em Disco

- Cada acesso a disco lê ou escreve um bloco (4 KB)
  - Dados maiores que 4 KB, demandam mais de um acesso
-
  -
  -

Lições para Programar em Disco

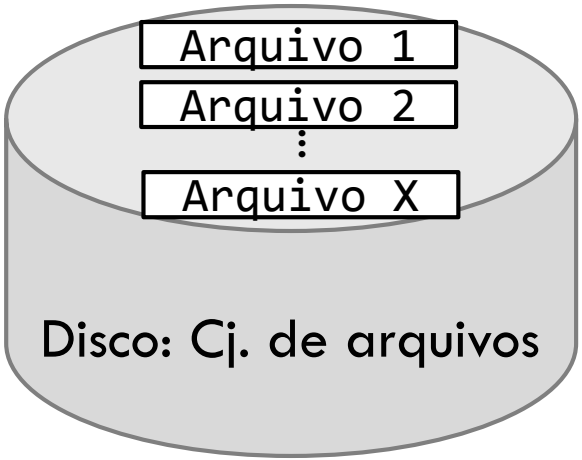
- Cada acesso a disco lê ou escreve um bloco (4 KB)
  - Dados maiores que 4 KB, demandam mais de um acesso
- Use a RAM como memória intermediária para diminuir os caros acessos a disco (*cachering*, *buffering*) Ex.:
  - Suponha que sua `struct` tem 1 KB. Então lembre que:
  - 1 escrita de 4 KB é muito melhor que 4 escritas de 1 KB

Acesso a Item na mem. externa

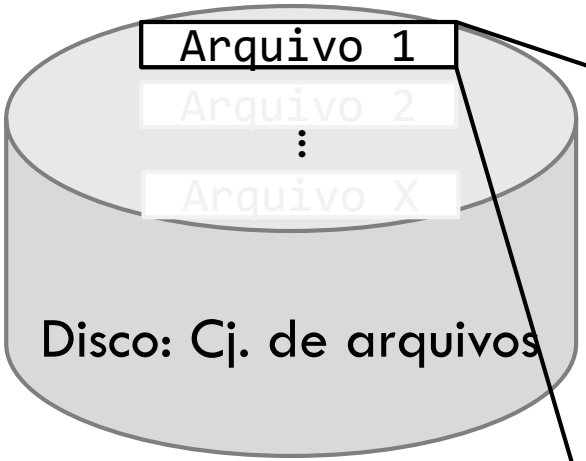
- Geralmente, fazemos um bloco conter um vetor de itens (i.e, structs)

Fonte da imagem: http://opendatastructures.org/ods-cpp/14_External_Memory_Searchin.html

Modelo de Programação em Disco



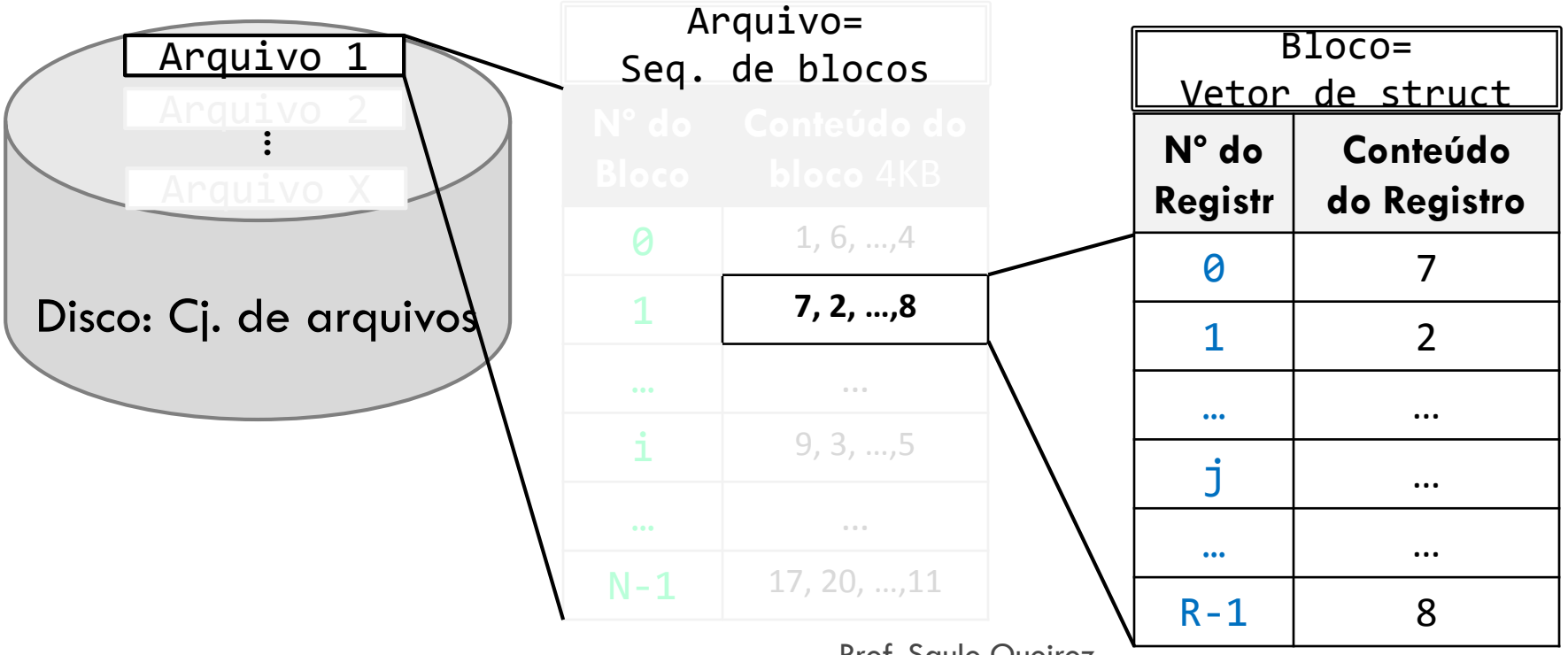
Modelo de Programação em Disco



Arquivo=
Seq. de blocos

N° do Bloco	**Conteúdo do bloco** 4KB
0	1, 6, ...,4
1	7, 2, ...,8
...	...
i	9, 3, ...,5
...	...
N-1	17, 20, ...,11

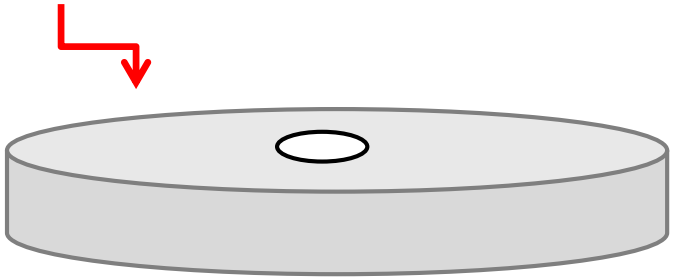
Modelo de Programação em Disco



Modelo de Programação em Disco

- Síntese:
  - Blocos indexados de `0` até `N-1` (`N` blocos)
    - `i` é o índice de um bloco
  - Registros indexados de `0` até `R-1` (`R` registros)
    - `j` é o índice de um registro no vetor de blocos
  - M quantidade de blocos que cabem na RAM
    - M = O(N/N)=O(1) Ex. M = 1 bloco (ou R registros)

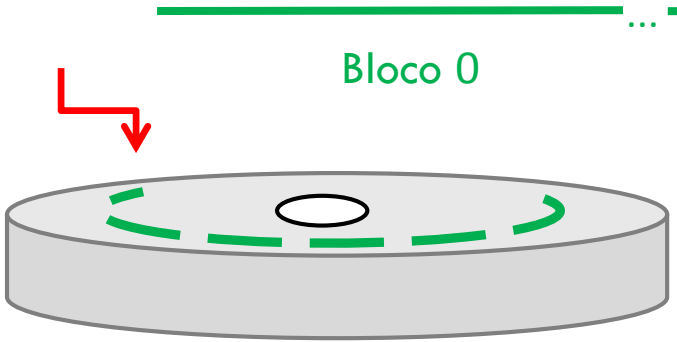
Arquivo em Disco: Abertura



Memória Interna (RAM)

Área do programa

Arquivo em Disco: Abertura



Memória Interna (RAM)

Área do programa

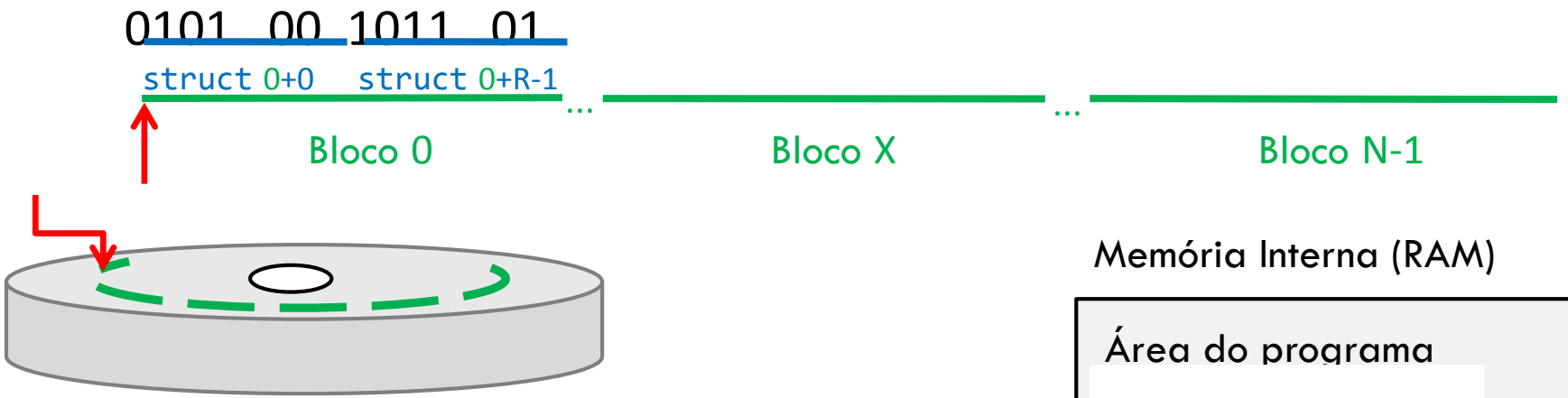
Arquivo em Disco: Abertura


```
FILE *p = fopen("arq.bin", "rb");
```

Memória Interna (RAM)

Área do programa

Arquivo em Disco: Abertura

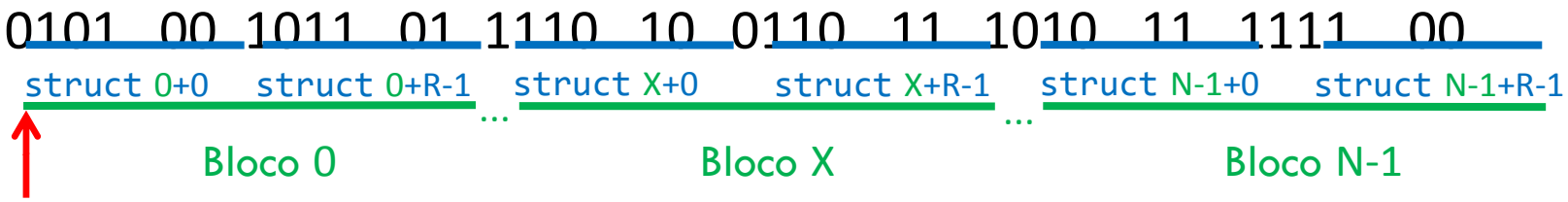


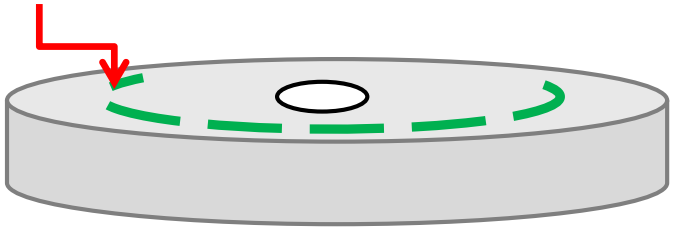

```
FILE *p = fopen("arq.bin", "rb");
```

Memória Interna (RAM)

Área do programa

Arquivo em Disco: Abertura



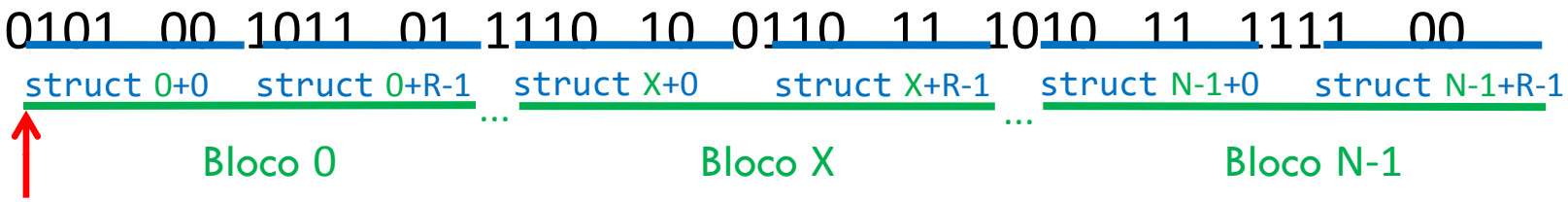


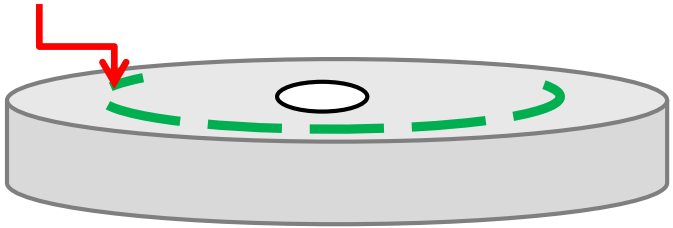
```
FILE *p = fopen("arq.bin", "rb");
```

Memória Interna (RAM)

Área do programa

Arquivo em Disco: Abertura



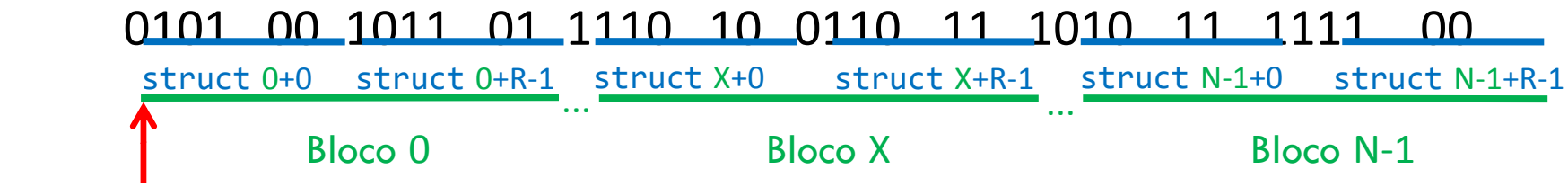


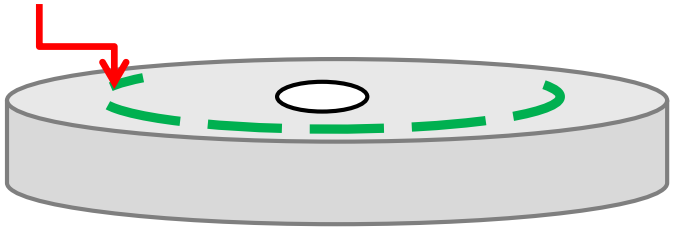
```
FILE *p = fopen("arq.bin", "rb");
if (p == NULL) return -1;
```

Memória Interna (RAM)

Área do programa

Arquivo em Disco: Abertura

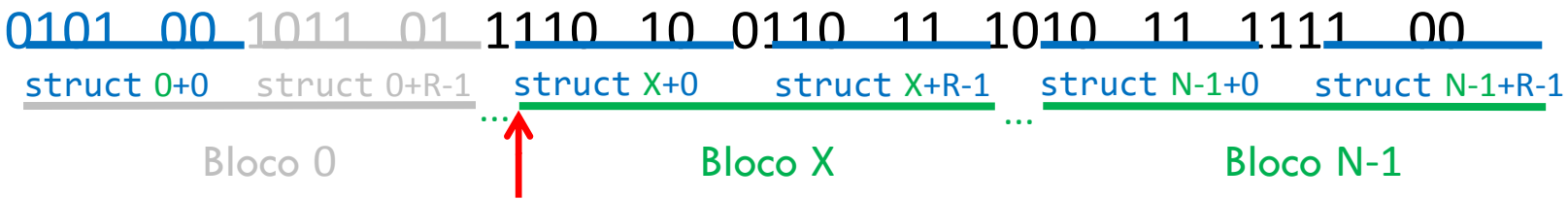


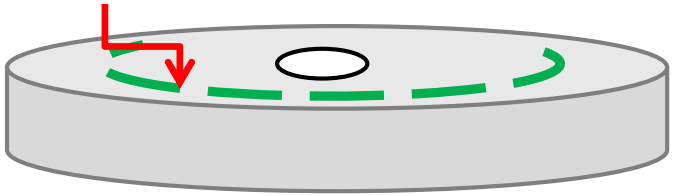


```
FILE *p = fopen("arq.bin", "rb");
if (p == NULL) return -1;
MyStruct varStruct;
```

Memória Interna (RAM)

Arquivo em Disco: Leitura





```
FILE *p = fopen("arq.bin", "rb");
if (p == NULL) return -1;
MyStruct varStruct;
//ler 1 unidade de sizeof(MyStruct) bytes
fread(&varStruct,sizeof(MyStruct),1,p);
```

Memória Interna (RAM)

Notas importantes

- 1 bloco foi lido (4 KB)
- Só `sizeof(MyStruct)*1` bytes foram carregados na RAM
  - Os demais dados do bloco foram descartados


```
FILE *p = fopen("arq.bin", "rb");
if (p == NULL) return -1;
MyStruct varStruct;
//ler 1 unidade de sizeof(MyStruct) bytes
fread(&varStruct,sizeof(MyStruct),1,p);
```

Notas importantes

- 1 bloco foi lido (4 KB)
- Só `sizeof(MyStruct)*1` bytes foram carregados na RAM
  - Os demais dados do bloco foram descartados
  - Poderíamos ter carregado todos os registros de bloco em um vetor de struct com R posições

```
MyStruct v[R];
fread(&v,sizeof(MyStruct),R,p);//&v==v
```

No final devemos fechar o arquivo `fclose(p);`

Busca sequencial externa

Prof. Saulo Queiroz

Busca sequencial indexada

Prof. Saulo Queiroz

Ordenação externa

Prof. Saulo Queiroz

Intercalação Balanceada de f Caminhos

- f é a quantidade de registros que cabem na RAM
  - f = M x R
- Entrada
  - Arquivo a ser ordenado
- Requisitos necessários:
  - 2f “fitas” (arquivos), sendo
    - espaço em disco não é problema!

Intercalação Balanceada f-caminhos

Animação em Aula

Ordenação Externa: Outros algoritmos

- Quick-sort externo
  - Usa somente 1 arquivo (não carece de “multiplos” caminhos)
  - Requisitos: Acesso aleatório e espaço O(log(RN)) na memória interna onde RN é o total de registros no disco
- Intercalação polifásica de múltiplos caminhos

Referências

- Shaffer, Clifford A. **Data Structures and Algorithm Analysis**. Ed. 3.2 (C++ Version), 2013.
  - Disponível gratuitamente em https://people.cs.vt.edu/shaffer/Book/C++3elatest.pdf
- https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.bpxbd00/feof.htm