Leitura e Escrita em Arquivos

INF1007 – Programação em C

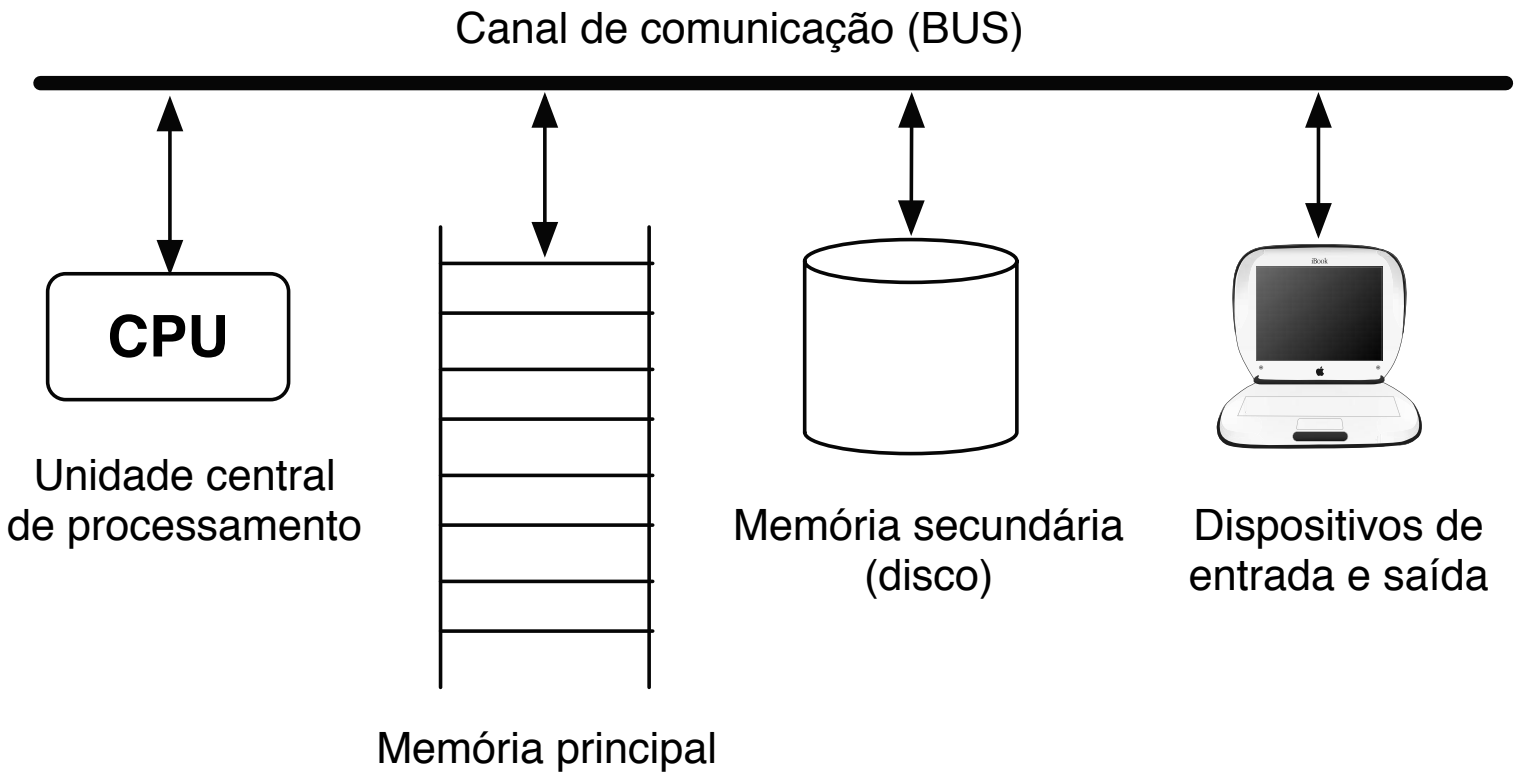
Waldemar Celes

Departamento de Informática, PUC-Rio

Modelo de computador

Memória secundária

- Memória persistente
- Operações de leitura e escrita ineficientes

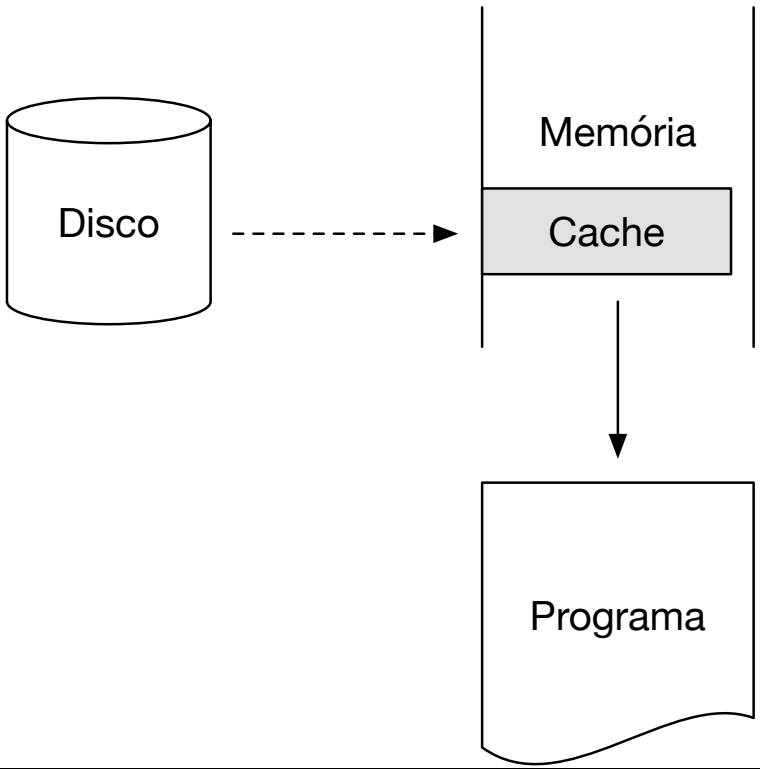


Operações em arquivos

Uso de cache em memória

- Esconder latência de acesso a disco
- É um buffer (espaço) na memória principal

Operações de leitura

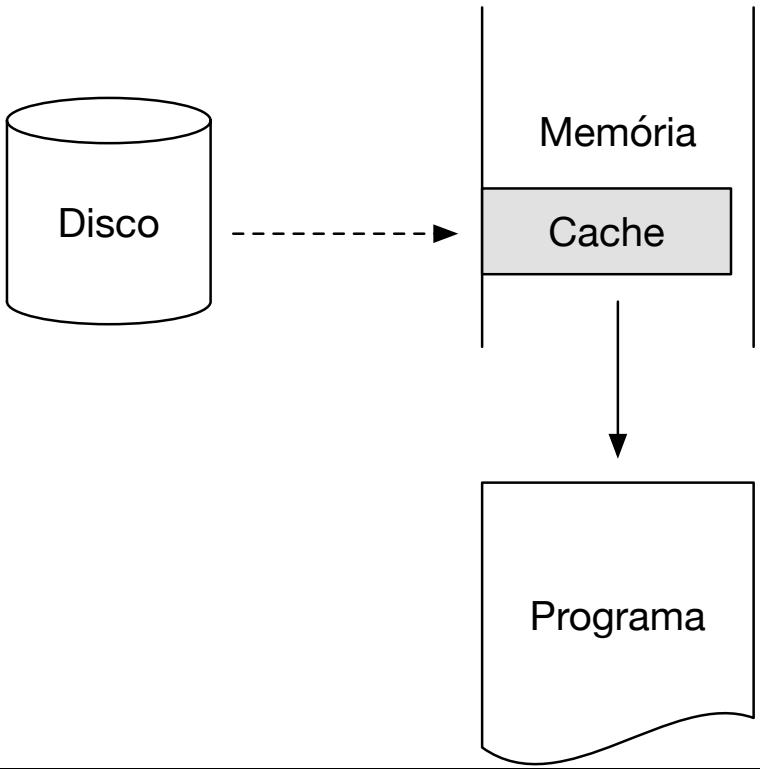


Operações em arquivos

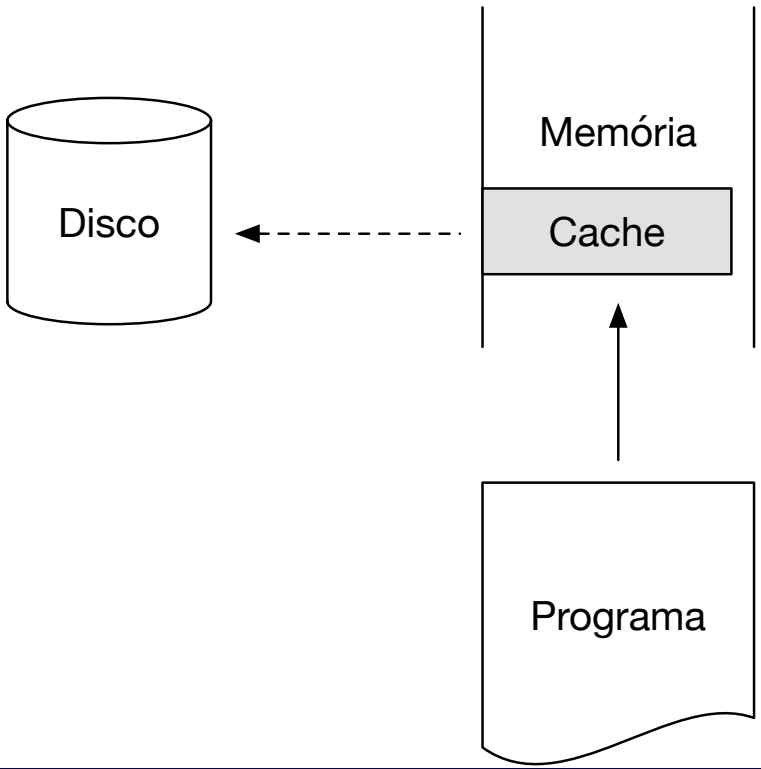
Uso de cache em memória

- Esconder latência de acesso a disco
- É um buffer (espaço) na memória principal

Operações de leitura



Operações de escrita



Arquivos

Identificação de arquivos: *diretório* (*pasta*) + *nome*

- Nome, em geral, com extensão
  - `.c`: extensão usada para arquivos com programas em C
  - `.txt`: extensão usada para arquivos com informações textuais
  - ...

Arquivos

Identificação de arquivos: *diretório* (*pasta*) + *nome*

- Nome, em geral, com extensão
  - `.c`: extensão usada para arquivos com programas em C
  - `.txt`: extensão usada para arquivos com informações textuais
  - ...

Tipos de arquivos

- Arquivo **texto**
  - Armazena uma sequência de caracteres
    - Pode ser lido e editado por humanos
- Arquivo **binário**
  - Armazena sequência de bytes
    - Permite operações de leitura e escrita mais eficientes

Arquivos

Operações providas pelos sistema operacional

- Disponibilizadas em C pela biblioteca `stdio.h`

Arquivos

Operações providas pelos sistema operacional

- Disponibilizadas em C pela biblioteca `stdio.h`

- **Abertura de arquivo**
  - Sistema operacional localiza/abre arquivo e prepara cache

Arquivos

Operações providas pelos sistema operacional

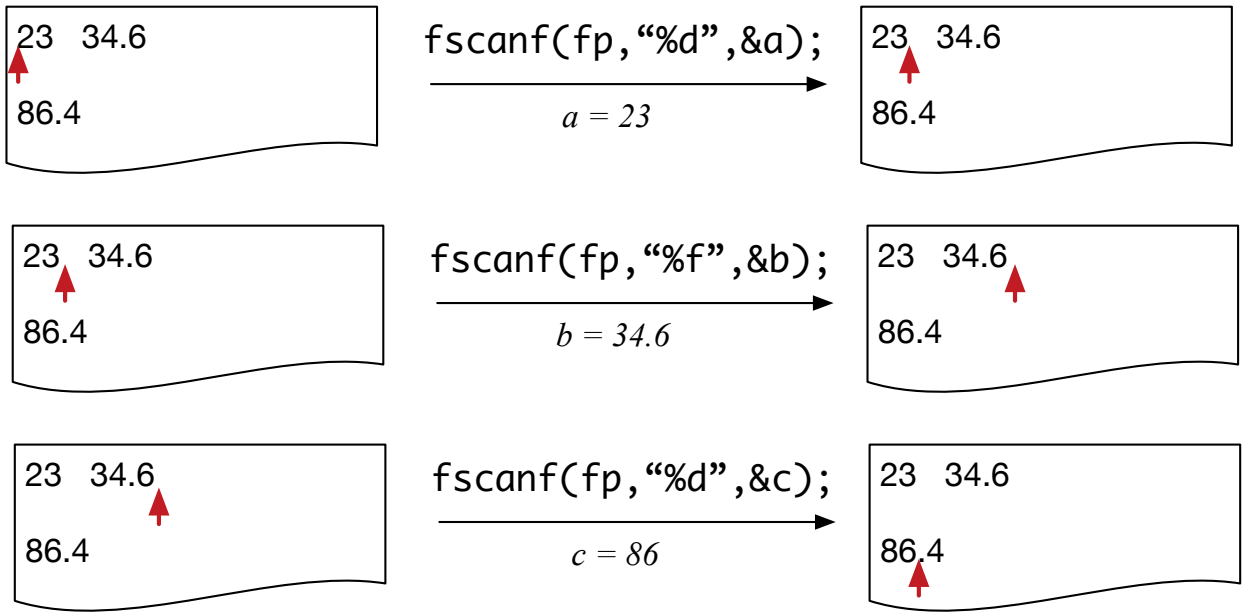
- Disponibilizadas em C pela biblioteca `stdio.h`

- **Abertura de arquivo**
  - Sistema operacional localiza/abre arquivo e prepara cache
- **Leitura de arquivo**
  - Sistema operacional recupera o trecho solitado do arquivo, que pode vir do cache
- **Escrita de arquivo**
  - Sistema operacional acrescenta/altera o trecho solitado no arquivo, usando o cache para acumular informação e minimizar acesso so disco
- **Fechamento de arquivo**
  - Sistema operacional atualiza arquivo com informação do cache (se for de escrita), fecha arquivo e libera cache.

Arquivo

Cursor do arquivo

- Posição corrente no arquivo
  - Operação de leitura captura próxima informação
  - Operação de escrita escreve na posição corrente
- Cursor é atualizada após cada operação
  - Existem funções para mover o cursor sem ler/escrever



Funções da biblioteca padrão

- Função para abrir um arquivo

```
FILE* fopen (char* nome_arquivo , char* modo);
```

- `FILE`: tipo definido por `stdio.h`
- `nome_arquivo`: relativo ou absoluto (*path* completo)
  - Exemplo Windows
    - `"arquivo.txt"` (busca no diretório corrente)
    - `"C:/Users/usuario/programas/arquivo.txt"`, ou
    - `"C:\\Users\\celes\\programas\\arquivo.txt"`
  - Exemplo de camiho completo no MacOS
    - `"/Users/usuario/programas/arquivo.txt"`

Funções da biblioteca padrão

- Função para abrir um arquivo

```
FILE* fopen (char* nome_arquivo , char* modo);
```

- `FILE`: tipo definido por `stdio.h`
- `nome_arquivo`: relativo ou absoluto (*path* completo)
  - Exemplo Windows
    - `"arquivo.txt"` (busca no diretório corrente)
    - `"C:/Users/usuario/programas/arquivo.txt"`, ou
    - `"C:\\Users\\celes\\programas\\arquivo.txt"`
  - Exemplo de camiho completo no MacOS
    - `"/Users/usuario/programas/arquivo.txt"`
- `modo`: string com o modo de abertura
  - Leitura em modo texto: `"r"` ou `"rt"`
  - Escrita em modo texto: `"w"` ou `"wt"`
  - Leitura em modo binário: `"rb"`
  - Escrita em modo binário: `"wb"`

Abertura de arquivos

Abertura de arquivo para **leitura**

- Arquivo deve existir
- Usuário deve ter permissão de leitura

```
...
FILE* fp = fopen("entrada.txt","rt");
if (fp == NULL) {
  printf("Erro na abertura do arquivo!\n");
  exit (1);
}
```

Abertura de arquivos

Abertura de arquivo para **leitura**

- Arquivo deve existir
- Usuário deve ter permissão de leitura

```
...
FILE* fp = fopen("entrada.txt","rt");
if (fp == NULL) {
  printf("Erro na abertura do arquivo!\n");
  exit (1);
}
```

Abertura de arquivo para **escrita**

- Se arquivo existe, é sobrescrito (cuidado!)
- Usuário deve ter permissão de escrita no diretório

```
...
FILE* fp = fopen("saida.txt","wt");
if (fp == NULL) {
  printf("Erro na abertura do arquivo!\n");
  exit (1);
}
```

Fechamento de arquivos

Função para fechar arquivo

- Todo arquivo aberto tem que ser fechado
  - Existe um limite do número de arquivos abertos do sistema operacional

```
int fclose (FILE* fp);
```

- Valor de retorno
  - `0`: se bem sucedido
  - `EOF`: se ocorreu erro (`EOF` é constante definida pela `stdio.h`)

Fechamento de arquivos

Função para fechar arquivo

- Todo arquivo aberto tem que ser fechado
  - Existe um limite do número de arquivos abertos do sistema operacional

```
int fclose (FILE* fp);
```

- Valor de retorno
  - `0`: se bem sucedido
  - `EOF`: se ocorreu erro (`EOF` é constante definida pela `stdio.h`)

```
...
FILE* fp = fopen("entrada.txt","rt");
...
fclose(fp);
...
```

Arquivo em modo texto

Funções para ler dados

- Leitura formatada

```
int fscanf (FILE* fp, char* formato, ...);
```

  - Retorna número de valores lidos com sucesso
  - Preenche os endereços das variáveis

Arquivo em modo texto

Funções para ler dados

- Leitura formatada

```
int fscanf (FILE* fp, char* formato, ...);
```

  - Retorna número de valores lidos com sucesso
  - Preenche os endereços das variáveis

- Leitura caractere a caractere

```
int fgetc (FILE* fp);
```

  - Retorna o código do caractere lido (ou `EOF`)

Arquivo em modo texto

Funções para ler dados

- Leitura formatada

```
int fscanf (FILE* fp, char* formato, ...);
```

  - Retorna número de valores lidos com sucesso
  - Preenche os endereços das variáveis

- Leitura caractere a caractere

```
int fgetc (FILE* fp);
```

  - Retorna o código do caractere lido (ou `EOF`)

- Leitura de linha

```
char* fgets (char* s, int n, FILE* fp);
```

  - Preenche e retorna a string passada
    - A leitura inclui o `\n`
    - Lê no máximo `n-1` caracteres do arquivo
    - Retorna `NULL` em caso de erro

Arquivo em modo texto

Formato da `fscanf` como "casamento de padrão"

- Espaço em branco: cursor pula eventuais espaços em branco
- Caractere não branco: cursor verifica ocorrência e pula caractere (funciona como uma "máscara")
- Especificador `%[...]`: cursor avança sobre eventuais caracteres do conjunto, capturando-os
- Especificador `%[^...]`: cursor avança sobre eventuais caracteres do complemento do conjunto, capturando-os
- Especificador com `%*...`: cursor avança o tipo especificado sem capturar

Arquivo em modo texto

Formato da `fscanf` como "casamento de padrão"

- Espaço em branco: cursor pula eventuais espaços em branco
- Caractere não branco: cursor verifica ocorrência e pula caractere (funciona como uma "máscara")
- Especificador `%[...]`: cursor avança sobre eventuais caracteres do conjunto, capturando-os
- Especificador `%[^...]`: cursor avança sobre eventuais caracteres do complemento do conjunto, capturando-os
- Especificador com `%*...`: cursor avança o tipo especificado sem capturar

---

Função de consulta

- Indica se o cursor alcançou o fim do arquivo

```
int feof (FILE* fp);
```

Arquivo em modo texto

Exemplos de leitura de dados formatados

- Arquivo de entrada

```
                              Rio de Janeiro, 04/05/2021

14:27:34, Mensagem no twitter
14:28:14, Outra mensagem no twitter
14:28:54, Terceira mensagem no twitter
```

Arquivo em modo texto

```
#include <stdio.h>

int main (void)
{
  FILE* fp = fopen("t.dat","r");
  if (!fp)
    return 1;
  char cidade[64];
  int dia, mes, ano;
  fscanf(fp, " %[^,], %d/%d/%d",cidade, &dia, &mes, &ano);
  printf("Cidade: %s\n", cidade);
  printf("Data: %02d/%02d/%d\n\n", dia, mes, ano);

  int hora, min, seg;
  char msg[128];
  while (fscanf(fp, "%d:%d:%d, %[^\n]",
                &hora, &min, &seg, msg) == 4) {
    printf("Hora: %02d:%02d:%02d\n", hora, min, seg);
    printf("Mensagem: %s\n---\n", msg);
  }
  fclose(fp);
  return 0;
}
```

```
                    Rio de Janeiro, 04/05/2021

14:27:34, Mensagem no twitter
14:28:14, Outra mensagem no twitter
14:28:54, Terceira mensagem no twitter
```

Arquivo em modo texto

Saída

```
Cidade: Rio de Janeiro
Data: 04/05/2021

Hora: 14:27:34
Mensagem: Mensagem no twitter
---
Hora: 14:28:14
Mensagem: Outra mensagem no twitter
---
Hora: 14:28:54
Mensagem: Terceira mensagem no twitter
---
```

Arquivo em modo texto

Funções para escrever dados

- Escrita formatada

```
int fprintf (FILE* fp, char* formato, ...);
```

  - Retorna o número de bytes escritos (similar à `printf`)

- Escrita de um caractere

```
int fputc (int c, FILE* FP);
```

  - Retorna o próprio código do caractere escrito ou `EOF`

Arquivo em modo texto

```c
#include <stdio.h>
int main (void)
{
  FILE* fp = fopen("t.dat","r");
  FILE* out = fopen("t.out","w");
  if (!fp || !out)
    return 1;
  char cidade[64];
  int dia, mes, ano;
  fscanf(fp, " %[^,], %d/%d/%d", cidade, &dia, &mes, &ano);
  fprintf(out, "Cidade: %s\n", cidade);
  fprintf(out, "Data: %02d/%02d/%d\n\n", dia, mes, ano);

  int hora, min, seg;
  char msg[128];
  while (fscanf(fp, "%d:%d:%d, %[^\n]",
                &hora, &min, &seg, msg) == 4) {
    fprintf(out, "Hora: %02d:%02d:%02d\n", hora, min, seg);
    fprintf(out, "Mensagem: %s\n---\n", msg);
  }
  fclose(out);
  fclose(fp);
  return 0;
}
```

Exercício

BixTex é um software de gerenciamento de referências comumente usado pelo LaTex. Abaixo, ilustra-se a base BibTex de um artigo.

```
@article{turing_paper
    author = {Turing, A. M.},
    title = {Computing Machinery and Intelligence},
    journal = {Mind},
    volume = {LIX},
    number = {236},
    pages = {433-460},
    year = {1950},
}
```

Faça um programa que leia um artigo no formato ilustrado abaixo e exiba na tela o título do artigo entre aspas duplas, seguido na linha seguinte pelo autor , o número de páginas e o ano, como ilustrado abaixo.

"Computing Machinery and Intelligence"

Turing, A. M., 28 pages, 1950.