UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Arquivos

Aula 15

Marcos Silvano Almeida
marcossilvano@professores.utfpr.edu.br
Departamento de Computação
UTFPR Campo Mourão

Arquivos

- As informações lidas e escritas pela CPU/GPU ficam em memória primária
  - Exemplo: RAM, ROM, VRAM, Cache
  - Precisam ser rápidos para CPU/GPU não ficar ociosa
  - Volátil

- Já as informações persistidas (***storage***), ficam em memória secundária
  - Mains lentos
  - Não volátil
  - Discos: HDD, Floppy Disk
  - Óptico: CD, DVD, Blu-ray
  - Circuito integrado: SSD, Pen Drive

- Os dados no storage estão armazenados pelo sistema de arquivos
  - Tudo é **arquivo**: imagem, som, vídeo, programa, documento, ...

UTFPR

Tipos de arquivos

- Na linguagem C, considera-se arquivo **texto** ou **binário**.
  - Diferença ocorre no momento de ler/escrever dados no arquivo.
    - Não é uma boa prática misturar ASCII com dados binários.

- Arquivo Texto (ASCII)
  - Armazena dados em formato de **texto puro**.
  - Editável em editores de texto.
  - De fácil compreensão, não seguros e ocupam mais espaço.
    - 2147483647 (int, 4 bytes) VS “2147483647” (string, 10 x 1 byte)
  - Ex: documentos de office (docx, xlsx, pptx)

- Arquivo Binário
  - Armazena dados como **números binários** (0|1)
  - Não são de fácil compreensão, (um pouco) mais seguros e menos espaço.
  - Ex: executáveis, dados de programas, bibliotecas, etc…

Abrindo e fechando arquivo

- Para acessar arquivo:
  - fopen(): abre arquivo no modo informado
  - fclose(): fecha arquivo
- Quando abrimos o arquivo, um cursor é posicionado
  - Cursor de leitura/escrita ⇒ como em um editor de textos

```
FILE* file;
file = fopen("arquivo.txt","r"); // read
if (file == NULL) {
    printf("ERRO: Arquivo nao existe. \n");
    return 1;
}
// faz alguma coisa...

fclose(file);
```

É aconselhável sempre fechar o arquivo após utilizá-lo, mesmo que, em tese, o Sistemas Operacional o faça quando o programa terminar.

Modos de abertura de arquivo com **fopen()**

Modo	Significado	Cursor	Flags
r	Somente leitura.	Início	`O_RDONLY`
w	Somente escrita. Apaga conteúdo do arquivo, se já existir. Cria arquivo, se não existir.	Início	`O_WRONLY|O_CREAT|O_TRUNC`
a	Acrescentar. Acrescenta conteúdo ao final. Cria se não existir.	Final	`O_WRONLY|O_CREAT|O_APPEND`
r+	Leitura e escrita.	Início	`O_RDWR`
w+	Leitura e escrita. Apaga conteúdo do arquivo, se já existir. Cria arquivo, se não existir.	Início	`O_RDWR|O_CREAT|O_TRUNC`
a+	Leitura e escrita. Cria se não existir. Cursor no início (se leitura) ou final (se escrita).	Início/Final	`O_RDWR|O_CREAT|O_APPEND`

Arquivo texto

Leitura e escrita de caracteres

Lendo arquivo: caractere a caractere

```
void read_file_char() {
    FILE* file;
    file = fopen("arquivo.txt","r"); // read
    if (file == NULL) {
        printf("ERRO: Arquivo nao existe.\n");
        return;
    }
    // leitura char-a-char
    char ch = fgetc(file);
    while ( ch != EOF) {
        printf("%c", ch);
        ch = fgetc(file);
    }
    fclose(file);
}
```

fgetc() faz a leitura de um char do arquivo e o retorna (ou EOF).

Lendo arquivo: caractere a caractere (versão compacta)

```
void read_file_char() {
    FILE* file;
    file = fopen("arquivo.txt","r"); // read
    if (file == NULL) {
        printf("ERRO: Arquivo nao existe.\n");
        return;
    }
    // leitura char-a-char
    char ch;
    while ( (ch = fgetc(file)) != EOF ) {
        printf("%c", ch);
    }

    fclose(file);
}
```

fgetc() faz a leitura de um char do arquivo e o retorna (ou EOF).

Escrevendo arquivo: caractere a caractere

```c
void write_file_char() {
    FILE* file = fopen("arquivo.txt","w"); // write + truncate/create

    char text[] = "Texto de teste 1.\nTexto de teste 2.\n";
    for (int i = 0; text[i] != '\0'; i++) {
        fputc(text[i], file);
    }

    fclose(file);
}
```

Escreve char a char no arquivo. Função fputc() retorna EOF se houver algum problema.

Exemplo: cópia de arquivos via parâmetros da main

```c
int main(int argc, char const *argv[]) {
    if (argc <= 2) {
        printf("Uso: copy origem destino\n");
        return 1;
    }
    FILE* file = fopen(argv[1], "r");
    if (file == NULL) {
        printf("ERRO: arquivo '%s' "
               "nao existe.\n", argv[1]);
        return 1;
    }

    FILE* copy = fopen(argv[2], "w");
    if (copy == NULL) {
        printf("ERRO: arquivo '%s' "
               "nao pode ser criado.\n", argv[2]);
        fclose(file); // fecha arquivo origem
        return 1;
    }

        // copia arquivo, char a char
        char c = fgetc(file);
        while (c != EOF) {
            fputc(c, copy);
            c = fgetc(file);
        }

        printf("INFO: arquivo clonado.\n");

        fclose(file);
        fclose(copy);

        return 0;
    }
```

Arquivo texto
Leitura e escrita de strings

Lendo arquivo: string

```
void read_file_string() {
    FILE* file =fopen("arquivo.txt","r"); // read
    if (file == NULL) {
        printf("ERRO: Arquivo nao existe.\n");
        return;
    }
    int n = 200; // 199 chars + '\0'
    char buffer[n];
    while (fgets(buffer, n, file) != NULL) {
        printf("%s", buffer);
    }

    fclose(file);
}
```

Retorna a string lida ou NULL. fgets() lê n-1 chars da entrada e acrescenta ‘\0’ ao final.

Escrevendo arquivo: string

```
void write_file_string() {
    FILE* file = fopen("arquivo.txt","w"); // write truncate+create

    char text[] = "Texto de teste 1.\nTexto de teste 2.\n";

    fputs(text, file);

    fclose(file);
}
```

Escreve a string no arquivo sem o '\0' ao final.

Exemplo++: conteúdo do arquivo para string expansiva

```c
char *load_text(const char *file_path) {        // Faz a leitura de conteúdo de texto de arquivo
    FILE *file = fopen(file_path, "r");                 // para um vetor alocado em heap
    if (!file) return NULL;

    int str_size = 50;
    char *str = malloc(str_size * sizeof(char));    // aloca array de char para texto do arquivo
    str[0] = 0;                                        // string vazia, com NULL na primeira posição

    char buffer[20];
    int buffer_counter = 0;
    while (fgets(buffer, 20, file) != NULL) {       // lê trechos de 19 chars do arquivo
        buffer_counter += strlen(buffer);
        if (buffer_counter >= str_size) {           // se não cabe no vetor, realoca
            str_size += buffer_counter;
            char *new_str = realloc(str, str_size * sizeof(char));
            if (!new_str) break;
            str = new_str;
        }
        strcat(str, buffer);                        // concatena na string resultante
    }
    fclose(file);
    return str;
}
```

Arquivo texto

Leitura e escrita formatada

Escrevendo e lendo arquivo com fprintf() e fscanf()

```c
void write_read_file_formatted() {
    FILE* file = fopen("arquivo.txt","w+"); // write-read trucate/create

    // Escreve com formato: deve iniciar com char
    for (int i = 0; i < 10; i++) {
        fprintf(file, " %d - %s - %f", i+1, "TextoTeste", (i+1)/2.0f);
    }
    rewind(file); // mesmo que fseek(file, 0, SEEK_SET);

    // Leitura com o mesmo formato usado para escrita
    int a; float b; char str[20];
    while (fscanf(file, " %d - %s - %f", &a, str, &b) != EOF) {
        printf("%2d, %s, %.2f\n", a, str, b);
    }
    fclose(file);
}
```

String “interna” não pode conter espaços

Posiciona cursor no início do arquivo.

Arquivo binário

Leitura e escrita com fread() e fwrite()

Arquivo de registros (structs)

- Vamos considerar a seguinte estrutura

```c
#include <stdio.h>

#define NAME_N 51

typedef struct {
   int id;
   char name[NAME_N];
   char email[NAME_N];
} Person;

void print_records(int n, Person* list) {
   for (int i = 0; i < n; i++) {
       printf("%d, %s, %s\n",
              list[i].id,
              list[i].name,
              list[i].email);
   }
}
```

Simplificação de definição de tipo estruturado

```c
int main() {
    Person person_list[] = {
       {1, "Carlos Antonio", "carlosanto@gmail.com"},
       {2, "Maria Clara", "maria.crala@gmail.com"},
       {3, "Jose Eduardo", "jose.eduardo@gmail.com"},
       {4, "Marcos Cillo", "marcos.cillo@gmail.com"},
       {5, "Marcia Marcozo", "mmmarcozo@gmail.com"}
    };
    // grava o vetor de registros no arquivo
    save_records("records.txt", 5, person_list);

    // lê o vetor de registros do arquivo
    int n = 0;
    Person* list = load_records("records.txt", &n);
    print_records(n, list);
    free(list);
    return 0;
}
```

Funções utilizadas na main: save_records()

```
/* Escreve os registros de vetor em arquivo binario.
 * Parâmetros:
 *     file_name      Caminho do arquivo
 *     n              Contendo o número de registros a escrever
 *      list           Endereço do vetor de registros
 *  Retorno:
 *      Quantidade de registros escritos no arquivo.
 */
int save_records(const char* file_name, int n, Person* list) {
    FILE* file = fopen(file_name, "w");
    if (!file) return 0;

    fwrite(list, sizeof(Person), n, file);

    fclose(file);
    return n;
}
```

Escreve o vetor de estruturas Person inteiro do arquivo

Funções utilizadas na main: load_records()

```
/* Lê o arquivo de registros os retorna em um vetor alocado em heap.
 * Parâmetros:
 *     file_name      Caminho do arquivo
 *     n              Parâmetro de retorno, contendo o número de registros lidos
 * Retorno:
 *     Endereço do vetor com os registros lidos, ou NULL em caso de erro.
 */
Person* load_records(const char* file_name, int* n) {
    FILE* file = fopen(file_name, "r");
    if (!file) return NULL;

    fseek(file, 0, SEEK_END);
    int file_size = ftell(file);
    *n = file_size / sizeof(Person);
    rewind(file); // mesmo que fseek(file, 0, SEEK_SET);

    Person* list = malloc(file_size);
    fread(list, sizeof(Person), n, file);
    fclose(file);
    return list;
}
```

Calcula o tamanho do arquivo, ao posicionar o cursor em seu final.

Posiciona cursor no início do arquivo.

Lê o vetor de estruturas Person inteiro do arquivo

Função interessante: read_record()

```c
/* Lê um registro do arquivo.
 * Parâmetros:
 *     file_name      Caminho do arquivo
 *     pos            Índice do registro
 *     record         Endereço da variável Person que será preenchida com o registro do arquivo.
 * Retorno:
 *     1 para sucesso ou 0 para erro.
 */
int read_record(const char* file_name, int pos, Person* record) {
    FILE* file = fopen(file_name, "r");
    if (!file) return 0;

    if (fseek(file, pos * sizeof(Person), SEEK_SET) != 0) {
        return 0;
    }
    fread(record, sizeof(Person), 1, file);

    fclose(file);
    return 1;
}
```

fseek() retorna 0 para sucesso ou outro valor quando posição é inválida.

Usa fread() para ler 1 único registro do arquivo.

Questões interessantes

- Como ficaria a função **`get_size()`**?
  - Permite obter o tamanho do arquivo, em bytes
- Como ficaria a função **`get_len()`**?
  - Permite obter a quantidade de registros no arquivo
- Como ficaria a função **`save_record()`**?
  - Permite salvar um registro em uma posição
- Como ficaria a função **`add_record()`**?
  - Permite adicionar um registro ao final do arquivo
- Como seria possível **apagar** um registro do arquivo?
  - Sem reescrever o arquivo por completo!
  - Como marcar e gerenciar os "buracos" resultantes das remoções? Defrag?

Quadro geral

Funções para manipulação de arquivos

Operação	Função	Descrição
open	`fopen(caminho, modo) → FILE*`	Abre arquivo (ver **modo**)
close	`fclose(FILE*)`	Fecha arquivo
read char	`fgetc(FILE*) → int`	Lê char
write char	`fputc(char, FILE*) → int`	Escreve char
read string	`fgets(string, n, FILE*) → string`	Lê string até '\n' de até **n** chars, incluindo NULL
write string	`fputs(string, FILE*) → string`	Escreve string
read formatted	`fscanf(FILE*, string_formatada, ...)`	Lê string formatada
write formatted	`fprintf(FILE*, string_formatada, ...)`	Escreve string formatada
read bin	`fread(end_buffer, tamanho, qtde, FILE*)`	Lê tamanho x qtde de bytes
write bin	`fwrite(end_buffer, tamanho, qtde, FILE*)`	Escreve tamanho x qtde de bytes
get cursor	`ftell(FILE*) → posição (bytes)`	Obtém posição do cursor, em bytes
set cursor	`fseek(FILE*, posição_bytes, referência)` `referência: SEEK_SET, SEEK_END, SEEK_CUR`	Posiciona cursor, em bytes. A partir da posição de referência: início, fim ou atual.