Programação de Computadores

Aula #05

Arquivos Binários

O que são arquivos binários? Quais são as diferenças de um arquivo texto? Quais são as funções de manipulação de arquivos binários?

Ciência da Computação - BCC e IBM - 2024/01
Prof. Vinícius Fülber Garcia

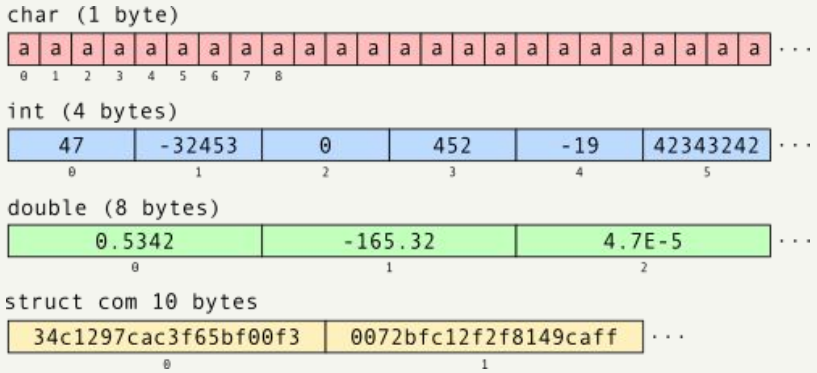
Arquivos Binários

Em suma, podemos compreender um arquivo como binário quando o mesmo contempla **dados não textuais** (isto é, que não seguem nenhuma tabela de codificação de símbolos, como ASCII ou UTF).

Quando nós falamos em arquivos contendo imagens, músicas, texto formatado e binários (entre outros), estamos nos referindo a arquivos binários!

Arquivos Binários

Tecnicamente, podemos entender um arquivo binário como uma **sequência de blocos de dados de mesmo tamanho**.



Por óbvio, o tamanho dos blocos de dados depende do tipo de informação persistida em um arquivo binário (define normalmente o “passo” de leitura).

Arquivos Binários

Vale a pena ressaltar: o SO compreende qualquer tipo de informação como uma sequência de bytes! Isto é, tudo é lido e escrito como uma sequência de bytes no arquivo.

Portanto, cabe à aplicação abstrair uma determinada quantidade sequencial de bytes como um tipo específico.

Manipulação de Arquivos Binários

Sabendo das características específicas dos arquivos binários, podemos inferir que vão existir funções específicas para realizar a leitura e escrita dos mesmos:

```
size_t fread (void* data, size_t size, size_t count, FILE* stream)
size_t fwrite (const void* data, size_t size, size_t count, FILE* stream)
```

Essas funções operam através da ideia de **blocos de dados**, sendo que um determinado bloco é definido pelo número de bytes que o compõem.

Manipulação de Arquivos Binários

```
size_t fread (void* data, size_t size, size_t count, FILE* stream)
```

A função *fread* é destinada à leitura de blocos de dados de um arquivo binário. Tal função define quatro parâmetros:

- data: *buffer* que armazenará os blocos lidos de um arquivo
- size: tamanho do bloco de dados, em bytes
- count: quantidade de blocos de dados a serem lidos
- stream: arquivo de origem dos blocos a serem lidos

O retorno consiste na quantidade de blocos lidos com sucesso para o *buffer*.

Manipulação de Arquivos Binários

```
size_t fwrite (const void* data, size_t size, size_t count, FILE* stream)
```

A função *fwrite* destina-se à escrita de blocos de dados de um arquivo binário. Essa função também define quatro parâmetros:

- data: *buffer* que armazena os blocos a serem escritos no arquivo
- size: tamanho do bloco de dados, em bytes
- count: quantidade de blocos de dados a serem escritos
- stream: arquivo de destino dos blocos a serem escritos

O retorno consiste na quantidade de blocos escritos com sucesso no arquivo.

Manipulação de Arquivos Binários

```c
#include <stdio.h>

int main() {
 int buffer[10];
 FILE *file = fopen("input.txt", "rb");
 fread(buffer, sizeof(int), 10, file);
 fclose(file);
 return 0;
}
```

Exemplo!

O programa ao lado faz uma leitura de dez blocos de dados do tamanho de um inteiro (4 bytes).

Manipulação de Arquivos Binários

```c
#include <stdio.h>

int main() {
 int buffer[10] = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
 FILE *file = fopen("output.txt", "wb");
 fwrite(buffer, sizeof(int), 10, file);
 fclose(file);
 return 0;
}
```

Exemplo!

O programa ao lado faz uma escrita de dez blocos de dados do tamanho de um inteiro (4 bytes).

Manipulação de Arquivos Binários

É importante ressaltar que, para todo arquivo aberto, o SO mantém um **ponteiro guardando a posição** na qual será realizada a próxima operação de arquivo.

Por default, as operações de arquivo sempre avançam o seu respectivo ponteiro, realizando um acesso sequencial até o final do arquivo.

Porém, eventualmente e dependendo do tipo de arquivo acessado, é necessário realizar **acesso direto a algumas posições** de um arquivo.

Manipulação de Arquivos Binários

Para realizar acesso a posições aleatórias de um arquivo, existe um conjunto de funções específicas disponíveis:

```
int fseek (FILE* stream, long int offset, int whence)
void rewind (FILE* stream)
long int ftell (FILE* stream)
```

Vale ressaltar que, no caso das funções de posicionamento, todas as posições se dão em termos dos bytes de um arquivo, não de blocos.

Manipulação de Arquivos Binários

A função de reposicionamento do ponteiro de um arquivo é a chamada *fseek*:

```
int fseek (FILE* stream, long int offset, int whence)
```

Tal função prevê três parâmetros:

- stream: arquivo em que se deseja mover o ponteiro
- offset: tamanho do deslocamento, em número de bytes
- whence: parâmetro de para aplicação do deslocamento (SEEK_SET, SEEK_END, SEEK_CUR)

Manipulação de Arquivos Binários

A função *rewind* é uma função específica para a movimentação do ponteiro para o início do arquivo (prevê apenas o arquivo como parâmetro):

```
void rewind (FILE* stream)
```

Finalmente, a função *ftell* indica a posição corrente do ponteiro do arquivo recebido como parâmetro (a partir do início):

```
long int ftell (FILE* stream)
```

Manipulação de Arquivos Binários

Existem algumas outras funções de manipulação de arquivos que podem ser interessantes:

```
int truncate (const char *path, off_t length);
int ftruncate (int fd, off_t length);
```

As **funções *truncate* removem dados de um arquivo** (a primeira sem necessidade de abrir o mesmo, a segunda com o arquivo já aberto), mantendo apenas os primeiros *length* bytes.

Manipulação de Arquivos Binários

Outras funções são dedicadas à obtenção de informações de arquivos:

```
int stat  (const char *pathname, struct stat *statbuf);
int fstat (int fd, struct stat *statbuf);
```

As **funções *stat* escrevem uma estrutura de informações sobre um arquivo** (a primeira sem necessidade de abrir o mesmo, a segunda com o arquivo já aberto), retornando zero (0) em caso de execução com sucesso.

Manipulação de Arquivos Binários

```
struct stat {
 dev_t     st_dev;         /* ID of device containing file */
 ino_t     st_ino;         /* Inode number */
 mode_t    st_mode;        /* File type and mode */
 nlink_t   st_nlink;       /* Number of hard links */
 uid_t     st_uid;         /* User ID of owner */
 gid_t     st_gid;         /* Group ID of owner */
 dev_t     st_rdev;        /* Device ID (if special file) */
 off_t     st_size;        /* Total size, in bytes */
 blksize_t st_blksize;     /* Block size for filesystem I/O */
 blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */
 struct timespec st_atim;  /* Time of last access */
 struct timespec st_mtim;  /* Time of last modification */
 struct timespec st_ctim;  /* Time of last status change */
 #define st_atime st_atim.tv_sec   /* For backward compatibility */
 #define st_mtime st_mtim.tv_sec
 #define st_ctime st_ctim.tv_sec
};
```

A estrutura *stat*

A estrutura *stat* é definida como mostra ao lado. O número de informações obtidas de um arquivo é bastante grande.

Exercício #5

Escreva um programa em C que leia um arquivo binário contendo informações de cidadãos em um mundo *cyberpunk*, incluindo seus identificadores únicos (*string*, até 10 caracteres), níveis de habilidades em hacking (*int*, de 0 a 100) e níveis de criminalidade (*int*, de 0 a 10). O programa deve exibir as seguintes informações:

1. A quantidade total de cidadãos no arquivo
2. Os identificadores e os níveis de habilidades em hacking de cada cidadão
3. A média dos níveis de habilidades em hacking de todos os cidadãos
4. O número de cidadãos com níveis de criminalidade acima de 5

Obrigado!

Vinícius Fülber Garcia
inf.ufpr.br/vinicius
vinicius@inf.ufpr.br