

# Recursos do Sistema Linux

Sistemas Embarcados

# Conteúdo

1. Acesso a arquivos
2. Processos e Sinais
3. Threads POSIX e biblioteca *pthread*s
4. Comunicação e Sincronismo entre Processos
5. Programação usando *sockets*
6. Device Drivers

# Acesso a Arquivos

1. **Arquivo regular:** tipo comum que contém dados somente. Os arquivos regulares podem ser dos mais variados tipos, guardando os mais diferentes tipos de informações. Existem arquivos de áudio, vídeo, imagem, texto, etc. Os arquivos se dividem em 2 (duas) categorias principais:
  - **Binários:** arquivos binários são compostos por bits 1 e 0 e só podem ser interpretados pelo sistema operacional. Programas e bibliotecas são exemplos de arquivos binários;
  - **Texto:** arquivos do tipo texto são compostos por informações em forma de texto, que podem ser entendidas pelo usuário comum. Arquivos desse tipo não necessariamente contém texto propriamente dito. Quando se diz que um arquivo é do tipo texto isto significa que, se aberto em um editor de texto, serão exibidos informações legíveis (ainda que possam não fazer muito sentido).

# Acesso a Arquivos

2. **Diretórios:** os diretórios são utilizados para separar um grupo de arquivos de outros. Um diretório pode conter arquivos e outros diretórios, que serão chamados subdiretórios;
3. **Dispositivos:** todo componente de hardware instalável é chamado dispositivo. Placas de vídeo, som, rede, drives de CD-ROM, tudo o que se liga na interface USB do PC, memória RAM, são dispositivos. Os dispositivos podem ser:
  - **De bloco:** dispositivos de bloco utilizam buffer para leitura/gravação. Geralmente são unidades de disco, como HDs, SSDs, PenDrive, CDs, etc;
  - **De caracter:** esses dispositivos não utilizam buffer para leitura/gravação. A maioria dos dispositivos PCI e outros dispositivos como impressoras, mouse, teclado, porta serial, etc. são do tipo caracter;

# Acesso a Arquivos

<i>Dispositivo de Caracter</i>	<i>Dispositivo de Bloco</i>
Acesso de dados sequencial, um byte por vez em modo de fluxo (stream).	Dispositivos de hardware que permitem acesso aleatório em blocos de tamanhos pré-definidos e fixos.
Não há bufferização	O acesso requer buffers.
Não há busca aleatória de posição (seeking)	É possível mover a posição do ponteiro de acesso (seeking).
Exige menos esforço do kernel para gerenciamento.	Exige mais esforço do kernel para gerenciamento.
Nas operações de escrita e leitura só há uma posição possível, a atual. Não há como avançar ou retroceder o ponteiro.	As operações de leitura / escrita podem ser feitas em qualquer posição do dispositivo.
As chamadas de leitura / escrita só retornam quando a operação é completada.	Leituras / Escritas são realizadas através de um mecanismo de buffer com cache.
O kernel não precisa de um subsistema específico para tratar este tipo de dispositivo.	O kernel necessita de um subsistema específico para tratar este tipo de dispositivo.
Menos complexo	Mais complexo.

# Acesso a Arquivos

4. **Links:** os links são arquivos utilizados para fazer referência a um outro arquivo localizado em outro local. Em outras palavras, são atalhos. Os links podem ser de 2 (dois) tipos:
- **Simbólicos:** fazem uma referência ao arquivo através de seu endereço lógico no disco ou memória. São os links mais comuns;
  - **Absolutos:** fazem referência ao arquivo através do seu endereço físico no disco rígido ou memória.
5. **FIFO:** canal de comunicação, utilizado para direcionar os dados produzidos por um processos para um outro processo.



# Bibliotecas Básicas em C

- **stdio.h** : Biblioteca C para efetuar operações de I/O (Entrada/Saída) - <http://www.cplusplus.com/reference/cstdio/>
- **stdlib.h** : Biblioteca de Propósito Geral Padrão - <http://www.cplusplus.com/reference/cstdlib/>
- **POSIX** - Portable Operating System Interface:  
Família de normas definidas pelo IEEE para a manutenção de compatibilidade entre sistemas operacionais.

# stdio.h

- Declaração de Arquivos:

```
FILE *ponteiro_para_arquivo;
```

- Função para abrir arquivos

```
FILE *fopen (char *nome_do_arquivo, char *modo)
```

- Função para Fechar arquivos

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



# stdio.h

<i>Modo</i>	<i>Significado</i>
<b>"r"</b>	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.
<b>"w"</b>	Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
<b>"a"</b>	Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
<b>"rb"</b>	Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário.
<b>"wb"</b>	Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário.
<b>"ab"</b>	Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário.

# stdio.h

<i>Modo</i>	<i>Significado</i>
<b>"r+"</b>	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
<b>"w+"</b>	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
<b>"a+"</b>	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
<b>"r+b"</b>	Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
<b>"w+b"</b>	Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
<b>"a+b"</b>	Acrescenta dados ou cria um arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário

# stdio.h

```
#include <stdio.h>

int main(int argc, const char * argv[]) {

    FILE *fp; // Declaração da estrutura
    fp=fopen ("exemplo.bin","wb"); /* o arquivo se chama
                                   exemplo.bin e está
                                   localizado no
                                   diretório corrente */

    if (!fp)
        printf ("Erro na abertura do arquivo.");

    return 0;
}
```

Exemplo\_1

**fp** = ponteiro para o arquivo

A posição atual na leitura do arquivo é armazenada a cada leitura.

**fseek()**

# stdlib.h

Abortar a execução de um programa:

```
void exit (int codigo_de_retorno);
```

```
#include <stdio.h>
#include <stdlib.h> /* Para a função exit() */

int main(int argc, const char * argv[]) {

    FILE *fp;
    fp=fopen ("exemplo.bin","wb");
    if (!fp)
    {
        printf ("Erro na abertura do arquivo. Fim de programa.");
        exit (1);
    }
    return 0;
}
```

Exemplo\_2

# stdio.h

## Escrita em arquivos:

```
int putc (int ch, FILE *fp);
```

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, const char * argv[]) {

    FILE *fp;
    char string[100];
    int i;
    fp = fopen("arquivo.txt","w");
    if(!fp)
    {
        /* Arquivo ASCII, para escrita */
        printf( "Erro na abertura do arquivo");
        exit(0); }
    printf("Entre com a string a ser gravada no arquivo:");
    gets(string);
    for(i=0; string[i]; i++)
        putc(string[i], fp); /* Grava a string, caractere a caractere */
    fclose(fp);
    return 0;
}
```

**Exemplo\_3**

# stdio.h

## Leitura de arquivos:

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

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, const char * argv[]) {

    FILE *fp;
    char c;
    fp = fopen("arquivo.txt","r"); /* Arquivo ASCII, para leitura */
    if(!fp)
    {
        printf( "Erro na abertura do arquivo");
        exit(-1);
    }
    while((c = getc(fp) ) != EOF) /* Enquanto não chegar ao final do arquivo */
        printf("%c", c);          /* imprime o caracter lido */
    fclose(fp);
    return 0;
}
```

Exemplo\_4



# stdio.h

## Detectar fim de arquivos:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, const char * argv[]) {
    FILE *p;
    char c, str[30], frase[80] = "Este é um arquivo chamado: ";
    int i;
    /* Le um nome para o arquivo a ser aberto: */
    printf("\n\n Entre com um nome para o arquivo:\n");
    gets(str);
    if (!(p = fopen(str, "w"))) /* Caso ocorra algum erro na abertura do arquivo...*/
    {                          /* o programa aborta automaticamente */
        printf("Erro! Impossível abrir o arquivo!\n");
        exit(1); }
    /* Se não houve erro, imprime no arquivo e o fecha ...*/
    strcat(frase, str);
    for (i=0; frase[i]; i++)
        putc(frase[i], p);
    fclose(p);
    /* Abre novamente para leitura */
    p = fopen(str, "r");
    c = getc(p);          /* Le o primeiro caracter */
    while (!feof(p)) {    /* Enquanto não se chegar no final do arquivo */
        printf("%c", c);  /* Imprime o caracter na tela */
        c = getc(p);      /* Le um novo caracter no arquivo */
    }
    fclose(p);           /* Fecha o arquivo */
}
```

Exemplo\_5

# Arquivos Padrão

- ***stdin***: dispositivo de entrada padrão (geralmente o teclado)
- ***stdout***: dispositivo de saída padrão (geralmente o vídeo)
- ***stderr***: dispositivo de saída de erro padrão (geralmente o vídeo)
- ***stdaux***: dispositivo de saída auxiliar (em muitos sistemas, associado à porta serial)
- ***stdprn*** : dispositivo de impressão padrão (em muitos sistemas, associado à porta paralela)

# Arquivos Padrão

Exemplo de uso:

```
ch = getc(stdin);
```

```
putc(ch, stdout);
```

# Leitura e Escrita em stream

Lê uma string de um *stream*

```
char *fgets (char *str, int tamanho, FILE *fp);
```

Escreve um string em um *stream*

```
char *fputs (char *str, FILE *fp);
```

# Tratamento de Erros

Verifica o indicador de erro

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

Imprime a mensagem de erro

```
void perror ( const char * str );
```

# Mais funções

## Exemplo\_6

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *pf;
    char string[100];
    if((pf = fopen("arquivo.txt","w")) ==NULL)
    {
        printf("\nNao consigo abrir o arquivo ! ");
        exit(1); }
    do {
        printf("\nDigite uma nova string. Para terminar, digite <enter>: ");
        gets(string);
        fputs(string, pf);
        putc('\n', pf);
        if(ferror(pf))
        {
            perror("Erro na gravacao");
            fclose(pf);
            exit(1);
        }
    } while (strlen(string) > 0);
    fclose(pf);
}
```



# Leitura e Escrita em stream (Arquivos)

Lê bloco de dados do *stream*

```
unsigned fread (void *buffer, int numero_de_bytes, int count, FILE *fp);
```

Escreve bloco de dados em um *stream*

```
unsigned fwrite (void *buffer, int numero_de_bytes, int count, FILE *fp);
```

# Mais funções

## Exemplo\_7

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *pf;
    float pi = 3.1415;
    float pilido;
    if((pf = fopen("arquivo.bin", "wb")) == NULL) /* Abre arquivo binário para escrita */
    {
        printf("Erro na abertura do arquivo");
        exit(1);
    }
    if(fwrite(&pi, sizeof(float), 1, pf) != 1)
        printf("Erro na escrita do arquivo");
    fclose(pf);

    if((pf = fopen("arquivo.bin", "rb")) == NULL) /* Abre o arquivo novamente para leitura */
    {
        printf("Erro na abertura do arquivo");
        exit(1);
    }
    if(fread(&pilido, sizeof(float), 1, pf) != 1) /* Le em pilido o valor da variável armazenada anteriormente */
        printf("Erro na leitura do arquivo");
    printf("\n0 valor de PI, lido do arquivo e': %f", pilido);
    fclose(pf);
    return(0);
}
```

# Mais funções

Reposiciona o ponteiro do *stream*

```
int fseek ( FILE * stream, long int offset, int origin );
```

<i>Nome</i>	<i>Valor</i>	<i>Significado</i>
<i>SEEK_SET</i>	0	Início do arquivo
<i>SEEK_CUR</i>	1	Posição corrente do arquivo
<i>SEEK_END</i>	2	Fim do arquivo

Volta ponteiro do *stream* para o início

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

Apaga um arquivo

```
int remove (char *nome_do_arquivo);
```

# Mais funções

## Exemplo\_8

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
    FILE *p;
    char str[30], frase[] = "Este e um arquivo chamado: ", resposta[80];
    int i;
    /* Le um nome para o arquivo a ser aberto: */
    printf("\n\n Entre com um nome para o arquivo:\n");
    fgets(str, 29, stdin); /* Usa fgets como se fosse gets */
    for(i=0; str[i]; i++)
        if(str[i]=='\n')
            str[i]=0; /* Elimina o \n da string lida */
    if (!(p = fopen(str, "w"))) /* Caso ocorra algum erro na abertura do arquivo..*/
    {
        printf("Erro! Impossivel abrir o arquivo!\n");
        exit(1);
    }
    fputs(frase, p);
    fputs(str, p);
    fclose(p);
    /* abre novamente e le */
    p = fopen(str, "r");
    fgets(resposta, 79, p);
    printf("\n\n%s\n", resposta);
    fclose(p);
    remove(str);
    return(0);
}
```

# Saída Formatada de dados

Escreve caracteres formatados em *stdout*

```
int printf (char *str,...);
```

Escreve caracteres formatados em uma *string*

```
int sprintf ( char * str, const char * format, ... );
```

Escreve caracteres formatados em *stream*

```
int fprintf ( FILE * stream, const char * format, ... );
```

# Saída Formatada de dados

<i><b>Códi</b></i>	<i><b>Formato</b></i>
<b>%c</b>	Um caracter (char)
<b>%d</b>	Um número inteiro decimal (int)
<b>%i</b>	O mesmo que %d
<b>%e</b>	Número em notação científica com o "e"minúsculo
<b>%E</b>	Número em notação científica com o "e"maiúsculo
<b>%f</b>	Ponto flutuante decimal
<b>%g</b>	Escolhe automaticamente o melhor entre %f e %e
<b>%G</b>	Escolhe automaticamente o melhor entre %f e %E

<i><b>Códi</b></i>	<i><b>Formato</b></i>
<b>%o</b>	Número octal
<b>%s</b>	String
<b>%u</b>	Decimal "unsigned" (sem sinal)
<b>%x</b>	Hexadecimal com letras minúsculas
<b>%X</b>	Hexadecimal com letras maiúsculas
<b>%%</b>	Imprime um %
<b>%p</b>	Ponteiro



# Leitura Formatada de dados

Lê dados formatados de *stdin*

```
int scanf (char *str,...);
```

Lê dados formatados de uma *string*

```
int sscanf ( const char * s, const char * format, ... );
```

Lê dados formatados de *stream*

```
int fscanf ( FILE * stream, const char * format, ... );
```

# Leitura Formatada de Dados

```
#include <stdio.h>
int main() {
    int i;
    char string1[20];

    printf( " Entre um valor inteiro: ");
    scanf("%d", &i);

    sprintf(string1, "Valor de i = %d", i);
    puts(string1);

    return 0;
}
```

# Leitura Formatada de Dados

```
#include <stdio.h>
int main() {
    int i, j, k;
    char string1[] = "10 20 30";

    sscanf(string1, "%d %d %d", &i, &j, &k);
    printf("Valores lidos: %d, %d, %d", i, j, k);

    return 0;
}
```

# Mais funções

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *p;
    char str[80], c;
    /* Le um nome para o arquivo a ser aberto: */
    printf("\n\n Entre com um nome para o arquivo:\n");
    gets(str);
    if (!(p = fopen(str, "w"))) /* Caso ocorra algum erro na abertura do arquivo...*/
    {
        printf("Erro! Impossível abrir o arquivo!\n");
        exit(1);
    }
    /* Se não houve erro, imprime no arquivo, fecha ...*/
    fprintf(p, "Este é um arquivo chamado:\n%s\n", str);
    fclose(p);
    /* abre novamente para a leitura*/
    p = fopen(str, "r");
    while (!feof(p))
    {
        fscanf(p, "%c", &c);
        printf("%c", c);
    }
    fclose(p);
    return(0);
}
```

# POSIX

Portable **O**perating **S**ystem Interface

# Biblioteca C vs UNIX/POSIX

## C stdio.h

- ▶ **fopen** - abre arquivo
- ▶ **fscanf** - entrada formatada
- ▶ **fprintf** - saída formatada
- ▶ **fclose** - fecha arquivo
- ▶ outros (fseek, rewind, ...)

+ Buffered, poderoso, sempre disponível

- buffering pode ser confuso;  
adiciona *overheads*

## UNIX / POSIX

- ▶ **open**
- ▶ **read**
- ▶ **write**
- ▶ **close**
- ▶ outros (mmap, lseek, poll, ...)

+ simples, rápido, baixo-nível,  
preferível para interfaces com o hardware

- somente para UNIX



# UNIX/POSIX I/O de Arquivos

- Sem formatação - somente read/write *raw bytes*
- Preferido para trabalhar com *hardware*
- Não faz parte da Biblioteca padrão do C

# POSIX open

abrir e opcionalmente criar um arquivo para leitura e/ou escrita

- Protótipo:

```
int open(const char* path, int oflag, ...)
```



“descriitor do arquivo”  
(integer)



caminho do arquivo



Flag de Abertura

```
{O_WRONLY;  
O_RDONLY;  
O_RDWR;...}
```

**Exemplo:**

```
int fd;  
fd = open("/tmp/teste.txt", O_WRONLY);
```

# POSIX open (Flags)

Devem ser combinados com "OR", definidos em <fcntl.h>

<b>O_RDONLY</b>	open for reading only
<b>O_WRONLY</b>	open for writing only
<b>O_RDWR</b>	open for reading and writing
<b>O_NONBLOCK</b>	do not block on open or for data to become available
<b>O_APPEND</b>	append on each write
<b>O_CREAT</b>	create file if it does not exist
<b>O_TRUNC</b>	truncate size to 0
<b>O_EXCL</b>	error if O_CREAT and the file exists
<b>O_SHLOCK</b>	atomically obtain a shared lock
<b>O_EXLOCK</b>	atomically obtain an exclusive lock
<b>O_NOFOLLOW</b>	do not follow symlinks
<b>O_SYMLINK</b>	allow open of symlinks
<b>O_EVTONLY</b>	descriptor requested for event notifications only
<b>O_CLOEXEC</b>	mark as close-on-exec

**Exemplo: (read/write, criar se não existir)**

```
int fd;
```

```
fd = open("/tmp/teste.txt", O_RDWR | O_CREAT);
```

# POSIX - Descritor de Arquivos

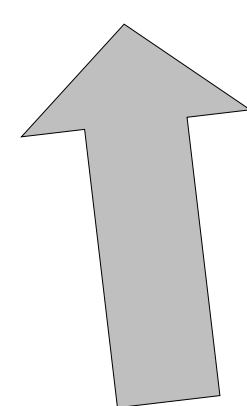
- ❖ Número inteiro que representa um "arquivo" **aberto**
- ❖ Cada processo (aplicação) possui uma tabela de descritores de arquivos (veja `/proc/<PID>/fd`). Alguns são padrão e abertos junto com a aplicação:
  - ❖ 0 - `stdin`
  - ❖ 1 - `stdout`
  - ❖ 2 - `stderr`
- ❖ A função **`open()`** retorna este descritor
  - ❖ [http://en.wikipedia.org/wiki/File\\_descriptor](http://en.wikipedia.org/wiki/File_descriptor)
- ❖ Os Tipos de “arquivo” podem ser:
  - ❖ arquivo em disco
  - ❖ diretório em disco
  - ❖ dispositivo de caracter (ex: porta serial)
  - ❖ dispositivo de bloco (ex: HDD; CDROM)
  - ❖ Tubo (*pipe*) (para comunicação interna)
  - ❖ *socket* (para rede)
  - ❖ outros...

Quase tudo no UNIX / Linux  
é representado por um arquivo

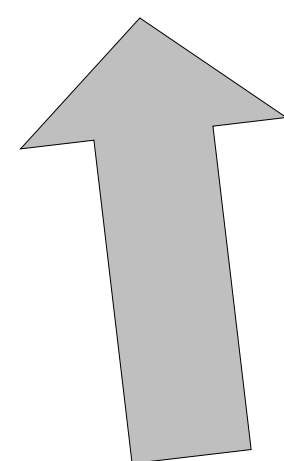
# POSIX read

- Protótipo:

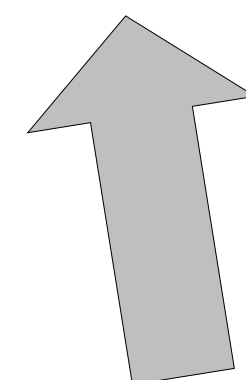
```
ssize_t read(int fildes, void *buf, size_t nbyte);
```



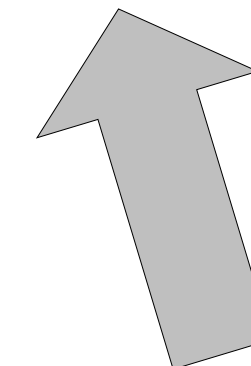
Número de bytes lidos  
ou EOF (0)  
ou Error (-1)



“descriptor do  
arquivo”  
(integer)



**Endereço** do  
buffer



Número de bytes  
a serem lidos

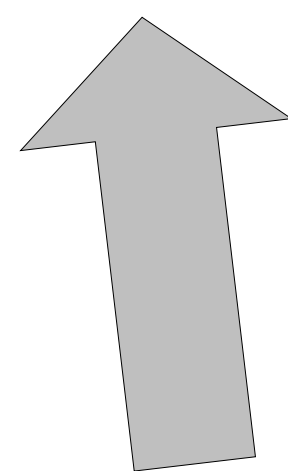
## Exemplo:

```
short siX16;  
int res = read(fid, &siX16, sizeof(short) );
```

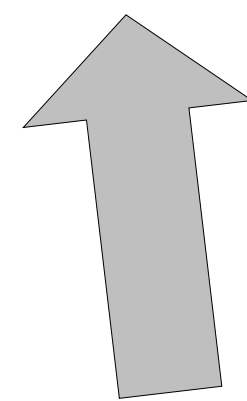
# POSIX write

- Protótipo:

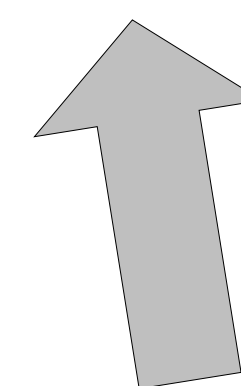
```
ssize_t write(int fildes, const void *buf, size_t nbyte);
```



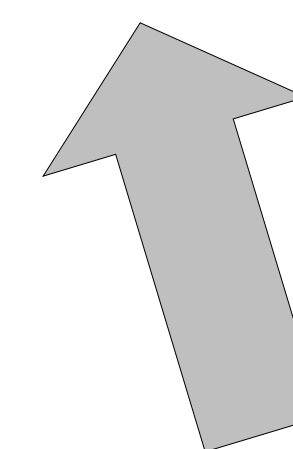
Número de  
bytes escritos  
ou Error (-1)



“descriptor de arquivo”  
(integer)



**Endereço** do  
buffer



Número de  
bytes a serem  
escritos

## Exemplo:

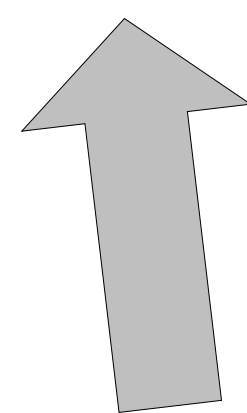
```
short siX16=0x7FFF;  
int res = write(fid, &siX16, sizeof(short) );
```

# POSIX lseek

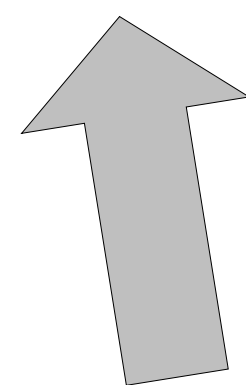
reposiciona o ponteiro do descritor do arquivo em um arquivo aberto

- Protótipo:

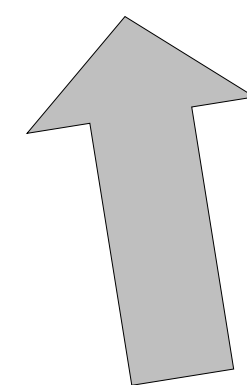
```
off_t lseek(int fd, off_t offset, int whence);
```



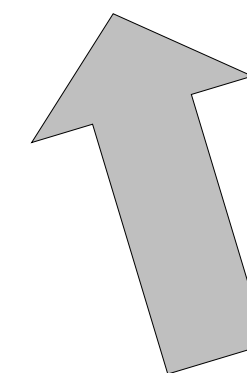
**“offset”**  
(-1 se der erro)



Descritor de  
arquivo



Valor de *offset*



{ SEEK\_SET;  
 SEEK\_CUR;  
 SEEK\_END }



# POSIX lseek

reposiciona o ponteiro do descritor do arquivo em um arquivo aberto

- Protótipo:

```
off_t lseek(int fd, off_t offset, int whence);
```

## Exemplo:

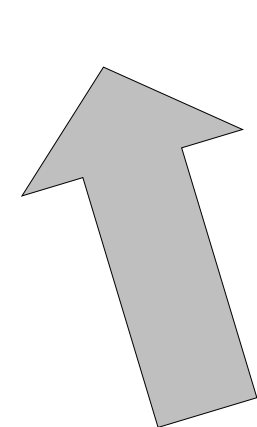
O Arquivo teste.txt contém o texto  
"Hello World"

```
int fd;  
char c;  
fd = open("/tmp/teste.txt", O_RDONLY);  
lseek(fd, 6, SEEK_SET);  
read(fd, &c, 1);
```

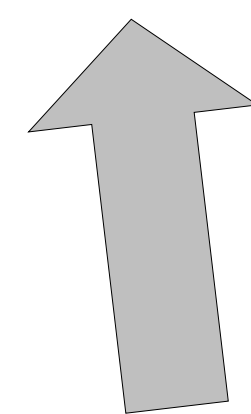
Qual character será gravado em **c**? **W**

# POSIX close

- Protótipo: `int close(int fildes)`



OK (0)  
Error (-1)



“descriptor de arquivo”  
(integer)

## **Exemplo:**

```
int iStatus = close(fd);
```