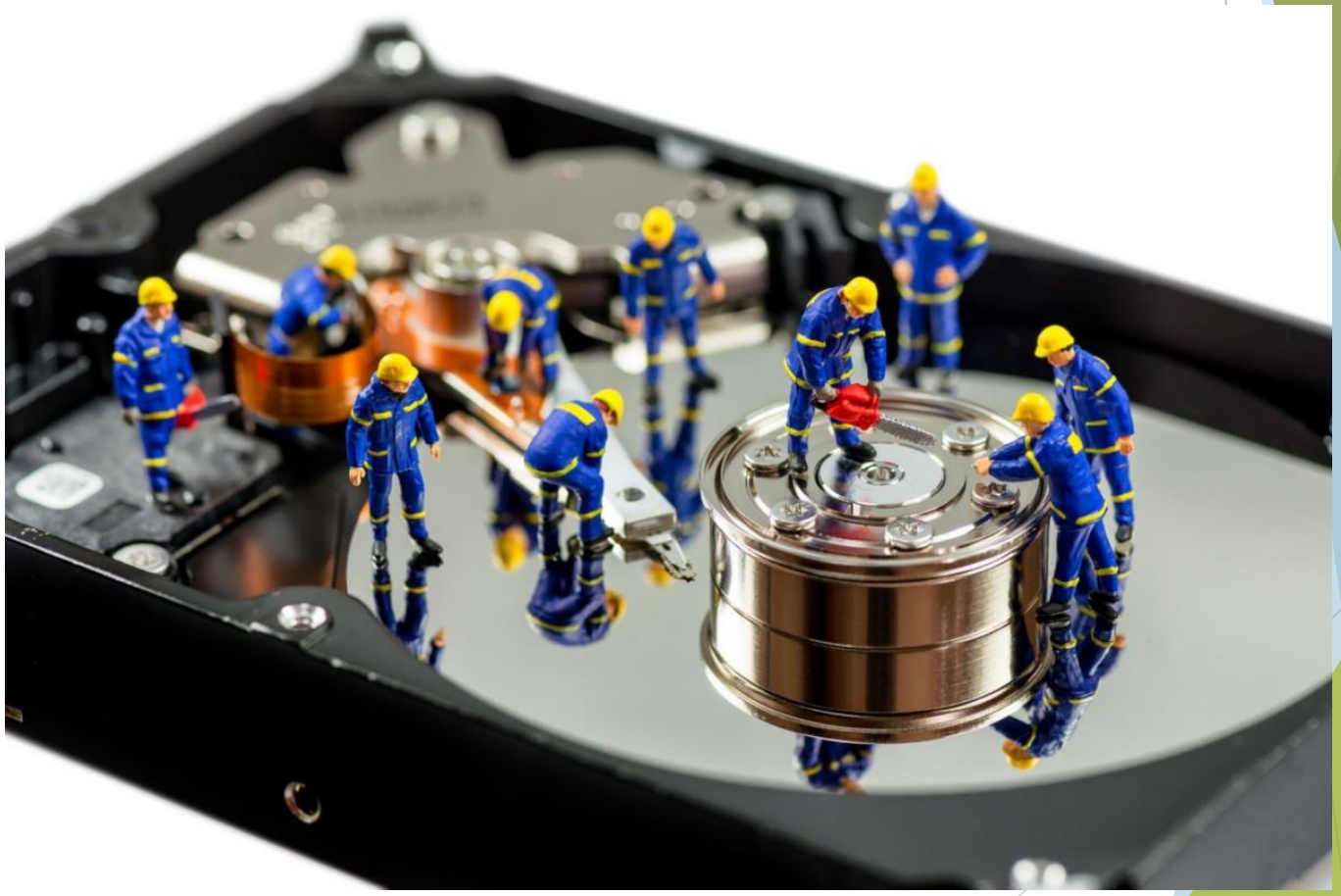
116327 -Organização de Arquivos



Organização de Arquivos

Disciplina: 116327

Prof. Oscar Fernando Gaidos Rosero

Universidade de Brasília – UnB
Instituto de Ciências Exatas – IE
Departamento de Ciência da Computação – CIC

Organização de Arquivos
Aula 2

M.Sc. Oscar Gaidos

Universidade de Brasília (UnB)

Sumário

- Conceitos Iniciais
- Histórico
- Arquivos
- Unix e DOS

UnB

Arquivo Lógico vs Arquivo Físico

- Arquivo Físico
- Uma coleção de bytes armazenado em um determinado dispositivo
- Arquivo Lógico
- Um “canal” (como uma linha telefónica) que conecta o programa ao arquivo físico. As operações do programa sobre o arquivo físico são feito através do arquivo lógico.
- Quando um determinado programa quer usar um determinado arquivo “dados”, o sistema operacional (SO) deve encontrar o arquivo físico e fazer a ligação do arquivo lógico ligado a ele.
  - Todas as operações no arquivo são via SO.
  - O arquivo físico tem um nome (myfile.txt, por exemplo).
  - O arquivo lógico tem um nome lógico que é usado dentro do programa.
  - Este nome lógico é uma **variável** dentro do programa.

UnB

Arquivos lógicos

Arquivos lógicos

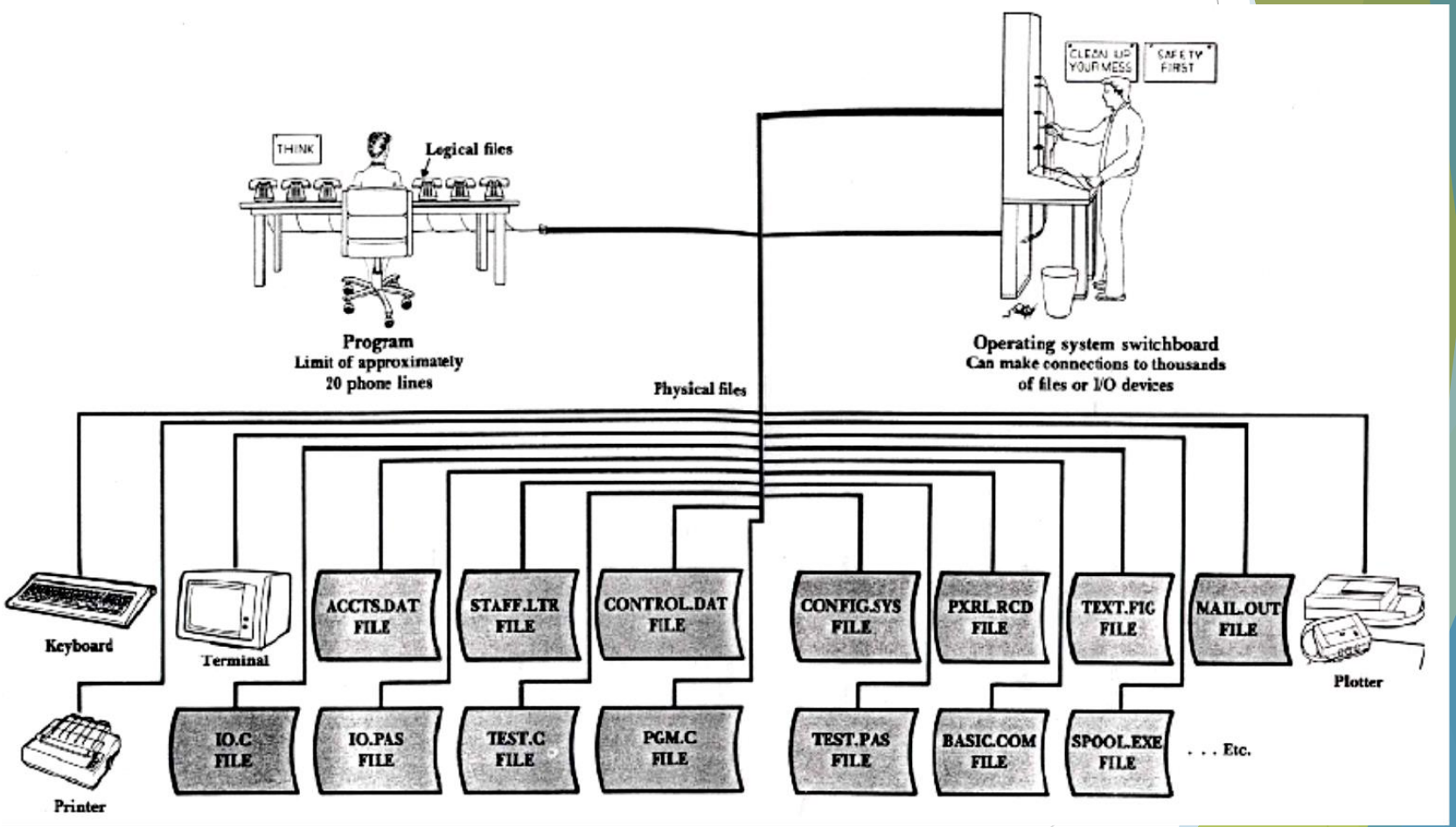
Utilizando a linguagem C, a estrutura lógica de arquivos (variável do tipo ponteiro) é declarada como:

FILE* fp;

Essa estrutura apontara para o nome lógico, o qual deve ser associado a um nome físico no momento de abertura do arquivo.

UnB

Arquivos lógicos



System Calls

- System Call (Syscall)
- Um system call (chamada ao sistema) é a forma que um programa tem para pedir um serviço ao kernel do sistema operacional que normalmente não teria permissão de executar.

- Em outras palavras o system call são funções que o programador pode utilizar para realizar serviços diretos do sistema operacional.

UnB

Manipulando Arquivos: Unix e C

- Manipulando Arquivos: Unix vs C

```
Syscall      : fd = open(filename, flags [, pmode])
Linguagem  C: fp = fopen(filename, type)

Syscall      : close (fd)
Linguagem  C: fclose(fp)

Syscall      : lseek(fd, offset, origin)
Linguagem  C: fseek(fp, offset, origin)

Syscall      : read (fd, buf, buf_size)
Linguagem  C: fread(buf, num, len, fp)

Syscall      : write (fd, buf, size)
Linguagem  C: fwrite(buf, num, len, fp)
```

A biblioteca de C onde estão as funções de manipulação básica de arquivos é a: **stdio.h**. Para utilizar as chamadas de baixo nível de arquivos (Syscalls), normalmente é necessário a biblioteca **fcntl.h**.

UnB

Abrindo Arquivos

- Abrindo Arquivos
- Abrir o arquivo torna-o pronto para ser usado pelo programa. Existem duas opções para abrir um arquivo:
  - Abrir um arquivo existente
  - Criar um novo arquivo

  Quando abrimos um arquivo, geralmente estamos posicionados no início do arquivo e estamos prontos para ler ou escrever nele.

UnB

Abrindo Arquivos: Syscall

- Abrindo arquivos usando syscall

```
Headers
int open(char * filename, int flags);
int open(char * filename, int flags, int mode);
    #include <fcntl.h>
    flags = bitwise | or of any of the following:
            O_RDONLY        Only read operations permitted
            O_WRONLY        Only write operations permitted
            O_RDWR          Read and Write operations both permitted
            O_NONBLOCK      Non-blocking, applies to open operation only
            O_APPEND        All writes go to end of file
            O_CREAT         Create file if it doesn't already exist
            O_TRUNC         Delete existing contents of file
            O_EXCL          Open fails if file already exists
            O_SHLOCK        Get a "shared lock" on the file
            O_EXLOCK        Get an "exclusive lock" on the file
            O_DIRECT        Try to avoid all caching of operations
            O_FSYNC         All writes immediately effective, no buffering
            O_NOFOLLOW      If file is symbolic link, open it, don't follow it
      mode required if file is created, ignored otherwise.
      mode specifies the protection bits, e.g. 0644 = rw-r--r-- (Octal)
      returns <0 for error, or integer file descriptor.
```

Obs: Se a flag estiver setada como O_APPEND, o ponteiro do arquivo apontrará para o final dele, e não para o começo.

UnB

Abrindo Arquivos: Permissões

- Permissões
- Como visto anteriormente, o modo passado durante a criação do arquivo estabelece que tipo de permissões cada usuário do sistema tem sobre o arquivo.
- O modo é um inteiro composto por três grupos de três bits
- Olhando o inteiro da esquerda para a direita, temos:
- O primeiro grupo tem as informações relativas às permissões do criador do arquivo
- O primeiro grupo tem as informações relativas às permissões dos usuários do grupo que o usuário pertence.
- O primeiro grupo tem as informações relativas aos usuários que não pertencem ao grupo do criador.

UnB

Abrindo arquivos: permissões

- Permissões
- O primeiro bit de cada um desses grupos, representa que é possível fazer a leitura do arquivo. O segundo bit representa que é possível fazer a escrita no arquivo. O terceiro bit representa que é possível executar o arquivo.

- Então, por exemplo, se tomarmos o número octal 0644 = 110100100 como o modo, temos que o criador do arquivo tem permissão de leitura e escrita no arquivo, os usuários pertencentes ao grupo do criador tem permissão de leitura apenas, e os usuários que não pertencem ao grupo do criador tem permissão de leitura apenas.

UnB

Abrindo arquivos: C

- Abrindo arquivos usando C

```
FILE *fp;
fp = fopen ("myfile.txt","w");
```

- O 1° argumento indica o nome físico do arquivo.
- O 2° argumento determina o "modo" (ou seja, a maneira como o arquivo é aberto). Este modo pode ser:
  - "r" : abrir um arquivo existente para input (leitura);
  - "w" : criar um novo arquivo, ou truncar um arquivo existente, para output;
  - "a" : abrir um novo arquivo, ou append um existente, para output;
  - "r+" : abrir um arquivo existente para input e output;
  - "w+" : criar um novo arquivo, ou truncar um existente, para input e output.
  - "a+" : criar um novo arquivo, ou append um existente, para input e output.
  - "rb", "wb", "ab" , "r+b" , "w+b" , "a+b" : mesmos modos descritos acima, porém o arquivo é aberto em modo binário.

UnB

Exemplo para Abrir um arquivo em C

```
/* list.c--program to read characters from a file and write them
**          to the terminal screen
*/
#include <stdio.h>
#include <fcntl.h>

main( )
{

        char c;
        int  fd;    /* file descriptor */
        char filename[20];

        printf("Enter the name of the file: ");     /* Step 1 */
        gets(filename);                             /* Step 2 */
        fd =open(filename, O_RDONLY);               /* Step 3 */

        while (read(fd, &c, 1) != 0)                /* Step 4a */
              write(STDOUT, &c, 1);                 /* Step 4b */

        close(fd);                                  /* Step 5 */
```

UnB

Exemplo para Abrir um arquivo em Pascal

```
PROGRAM list (INPUT, OUTPUT);

{ reads input from a file and writes it to the terminal screen }

VAR

      c         : char;
      infile    : file of char;                  { logical file name }

      filename : packed array [1..20] of char; { physical file name }

BEGIN {main}

      write('Enter the name of the file: ');     { Step 1 }
      readln(filename);                          { Step 2 }
      reset(infile, filename);                   { Step 3 }
      while not (eof(infile)) DO

      BEGIN

            read(infile,c);                      { Step 4a }
            write(c)                             { Step 4b }

      END;

      close(infile)                              { Step 5 }

END.
```

UnB

Fechando arquivos

- Fechando arquivos
- Ao fechar arquivos, o nome do arquivo lógico fica disponível para ser associado a outro arquivo físico.
- Fechar um arquivo que foi usado como output, assegura que eventuais dados do buffer sejam gravados no arquivo físico [uma vez que os dados são escritos antes num buffer, e depois gravados em blocos no arquivo].
- O S.O. normalmente fecha os arquivos após o término do programa [a menos que o programa seja interrompido de forma anormal].
- Comando para fechar o arquivo em C: **fclose(fd)**;
- Utilizand System Call: **close(fd)**;

UnB

Manipulando arquivos

- Manipulando arquivos
- São operações fundamentais no processamento de arquivos:
  - Abrir um arquivo
  - Escrever num arquivo
  - Ler de um arquivo
  - Posicionar o ponteiro do arquivo
- A sintaxe de linguagem para linguagem
- Funcionalidades
  - Variam de linguagem para linguagem: algumas linguagens provêm acesso de alto nível cuidando dos detalhes para o programador
  - Outras tem um acesso de baixo nível, deixando os detalhes para o programador

UnB

Lendo arquivos

- Lendo arquivos: Syscall

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t nbytes);
```

O syscall **read()** recebe vários argumentos:

- **fd**: Descritor do arquivo.
- **buf**: ponteiro da área de memória de onde os dados lidos do dispositivo serão escritos.
- **nbytes**: numero de bytes do dispositivo a serem lidos e depois guardados em **buf**.

UnB

Lendo arquivos

- Lendo arquivos: C

```
#include <stdio.h>
size_t fread (void * ptr, size_t size, size_t count, FILE * stream);

infile = fopen("myfile.txt","r");
fread(a,1,10,infile); /*Coloca 10 bytes a partir do endereco apontado de a*/
```

A função **fread()** é composta de vários parâmetros:

- ptr: Ponteiro para a área de memoria para onde os dados lidos serao guardados
- size: Tamanho de cada elemento a ser lido (em bytes)
- count: Quantidade de elementos a serem lidos
- stream: Arquivo lógico onde será feito a leitura

- A linguagem C, tem diversos outros comandos de leitura de mais alto nível que o syscall, conforme podemos ver em:
- http://www.cplusplus.com/reference/cstdio/

UnB

Escrevendo em arquivos

- Escrevendo em arquivos: Syscall

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t nbytes);
```

O syscall **write()** recebe vários argumentos:

- **fd**: Descritor do arquivo
- **buf**: ponteiro da área de memória de onde os dados serão coletados para serem escritos em um dispositivo.
- **nbytes**: numero de bytes de **buf** a serem lidos da memoria para serem escritos em um dispositivo.

UnB

Escrevendo em arquivos

- Escrevendo em arquivos: C

```
#include <stdio.h>
size_t fwrite (const void * ptr, size_t size, size_t count, FILE * stream);

outfile = fopen("mynew.txt","w");
fwrite(a,1,10,outfile); /*pega 10 bytes do endereco apontado por a
                          e escreve em outfile*/
```

A função **fwrite()** é similar a **fread()** quanto aos parâmetros:

- ptr: Ponteiro para a área de memoria de onde os dados serão lidos
- size: Tamanho de cada elemento a ser lido (em bytes)
- count: Quantidade de elementos a serem lidos
- stream: Arquivo lógico onde será feito a escrita

- A linguagem C, tem diversos outros comandos de escrita de mais alto nível que o syscall, conforme podemos ver em:
- http://www.cplusplus.com/reference/cstdio/

UnB

Acessando uma posição específica do arquivo

- Um programa não necessariamente deve ler um arquivo sequencialmente. Ele pode acessar um local específico diretamente, ou ir para o fim do arquivo. Esta ação de acessar diretamente uma posição em um arquivo é chamada de *seeking*.

Seeking: Syscall

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

Seeking: *C*

```
#include <stdio.h>
int fseek (FILE * stream, long int offset, int whence);
```

Ambos recebem o arquivo lógico, a quantidade de deslocamento a ser feita e o ponto de onde o deslocamento deve ser feito (começo do arquivo(SEEK_SET), posição atual (SEEK_CUR), ou fim do arquivo (SEEK_END), indicado pelo inteiro **whence**.

UnB

Detectando o fim do arquivo

- Detectando o fim do arquivo
- Os programas precisam saber quando a condição de parada dos *loops* de leitura são atingidas.
- Conforme um arquivo é lido, o SO controla a posição “atual” através de um ponteiro pro arquivo.
- Esse ponteiro é necessário para o sistema controlar qual o próximo byte a ser acessado por um comando de leitura.
- Na linguagem C, a função **fread()** retorna o número de bytes lidos. Se retornar zero, indica que o ponteiro chegou ao final do arquivo. Além disso, existe a função **feof()** que indica se um arquivo chegou ao fim.

UnB

Exemplo

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main(void){
        char c;
        int     fd; /*descritor de arquivo*/
        char filename[20];

        printf ("Enter the name of the file: ");
        scanf("%s",filename);
        fd = open (filename, O_RDONLY);
        lseek (fd, 501, 0);/*Desloca o ponteiro do arquivo em 501 bytes*/
        while (read(fd, &c, 1) != 0) /*enquanto read nao retornar 0*/
                write (1, &c, 1); /*escreve o byte na tela*/
        close (fd); /*fecha o arquivo*/
        return(0);
}
```

UnB

Unix

- Estrutura dos arquivos do Unix
- Em qualquer sistema de computação, existem muitos arquivos. Esses arquivos precisam ser organizados de alguma maneira, no Unix esta estrutura é chamada de *Unix File System*.
- É organizado numa estrutura de árvore (como o MS-DOS).
- O diretório raiz é identificado como “/”.
- Todos os diretórios, incluindo a raiz, contém dois tipos de arquivos, arquivos regulares de dados e programas, e diretórios.

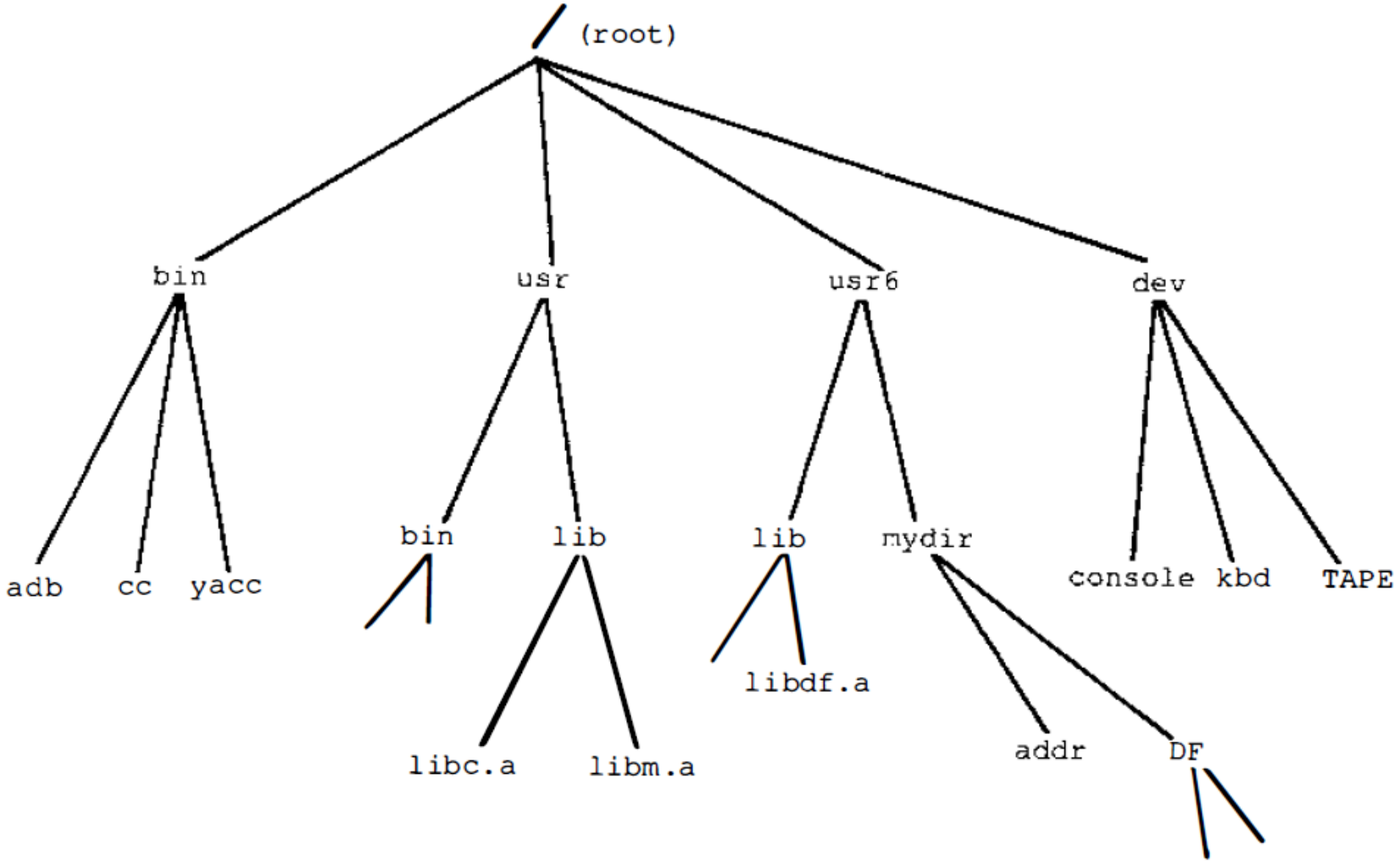
UnB

Unix

- Estrutura de arquivos do Unix
- Qualquer arquivo pode ser identificado unicamente fornecendo o seu caminho absoluto (*absolute path*). Ex: /usr6/mydir/addr.
- Dispositivos E/S também são tratados como arquivos, e seus drivers estão no diretório “/dev”
- O nome armazenado no diretório do Unix corresponde ao nome físico.
- O diretório corrente é identificado por “.”
- O diretório pai é identificado por “..”

UnB

Estrutura de diretórios no Unix



Unix

- Visão do Unix sobre arquivos
- O Unix tem uma visão bem genérica do que é um arquivo. Ele corresponde a uma sequência de bytes, sem preocupações sobre onde os bytes estão armazenados ou de onde eles vêm.
- Discos magnéticos (ou fitas) podem ser vistos como arquivos, e da mesma forma, o teclado e o controle (/dev/controle or /dev/kbd).
- Não importa qual a forma física de um arquivo do Unix (arquivo real ou dispositivo), a visão lógica do arquivo é a mesma.

- Flexibilidade!

UnB

Unix: arquivos especiais

- Saída padrão: STDOUT => Vídeo
- Entrada padrão: STDIN => Teclado
- Saída de erro padrão: STDERR => Vídeo (Quando o compilador detecta um erro, a mensagem de erro é escrita neste arquivo, por exemplo).

UnB

Unix: redirecionando arquivos

- “<” nomearq (redireciona STDIN para “nomearq”, isto é, a entrada padrão agora é o arquivo nomearq).
- “>” nomearq (redireciona STDOUT para “nomearq”, isto é, a saída padrão agora é o arquivo nomearq).

- Exemplo
- Suponha que prog.exe é um programa executável:
- Redirecionando Input (o input padrão passa a ser o arquivo in.txt):
- Prog.exe < in.txt
- Redirecionando Output (o output padrão passa a ser o arquivo out.txt):
- Prog.exe > out.txt

- Faça um teste num ambiente Unix-like, digite na linha de comando:
  - Print “Helllo World” > arq.txt

UnB

Sistema Unix-Like: Comandos Básicos

- Comando básicos de Sistemas Unix-Like
- cat filenames => Imprime o conteúdo dos arquivos.
- tail filenames => Imprime as últimas 10 linhas do arquivo.
- cp arquivo1 arquivo2 => copia arquivo1 em arquivo2.
- mv arquivo1 arquivo2 => move arquivo1 para arquivo2 (renomeia).
- rm filenames => remove arquivos
- chmode mode filename => modifica as permissões do arquivo.
- ls => Lista o conteúdo do diretório.
- mkdir name => Cria um diretório.
- rmdir name => Remove o diretório.
- cd diretory => Muda de diretório.

UnB

DOS

- Estrutura do DOS
- A estrutura em DOS também é uma árvore, porém uma unidade representa o topo da árvore. (ex: C:\ )
- O diretório corrente é identificado por “.”
- O diretório pai é identificado por “..”
- Os dispositivos **NÂO** são tratados como arquivos
- Não tem redirecionamento de E/S

    - Menos Flexível que o Unix

UnB