Chamadas de sistema de arquivos no Linux

MCTA026-13 - Sistemas Operacionais

Emilio Francesquini e Fernando Teubl Ferreira
e.francesquini@ufabc.edu.br / fernando.teubl@ufabc.edu.br

2019.Q1

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Disclaimer

UFABC

- Estes slides foram preparados para o curso de **Sistemas Operacionais na UFABC**.
- Este material foi preparado e gentilmente cedido pelo Prof. Gustavo Souza Pavani (UFABC)

Chamadas de sistema de arquivos no Linux

UFABC

- Chamadas de sistema relacionadas a arquivos

Chamada de sistema	Descrição
fd = creat(nome, modo)	Uma maneira de criar um novo arquivo
fd = open(arquivo, como, ...)	Abre um arquivo para leitura, escrita ou ambos
s = close(fd);	Fecha um arquivo aberto
n = read(fd, buffer, nbytes)	Lê dados de um arquivo para um buffer
n = write(fd, buffer, nbytes)	Escreve dados de um buffer para um arquivo
posição = lseek(fd, deslocamento, de-onde)	Move o ponteiro do arquivo
s = stat(nome, &buf)	Obtém a informação de estado do arquivo
s = fstat(fd, &buf)	Obtém a informação de estado do arquivo
s = pipe(&fd[0])	Cria um pipe
s = fcntl(fd, comando, ...)	Trava de arquivo e outras operações

Tabela 10.9 Algumas chamadas de sistema relacionadas a arquivos. O código de retorno *s* é –1 se ocorrer algum erro; *fd* é um descritor de arquivo e *position* é um offset de arquivo. Os parâmetros são autoexplicativos.

Chamadas de sistema de arquivos no Linux

UFABC

- Informações retornados pela chamada de sistema `stat`.

Dispositivo do arquivo
Número do i-node (qual arquivo do dispositivo)
Modo do arquivo (inclui informação de proteção)
Número de ligações para o arquivo
Identificação do proprietário do arquivo
Grupo ao qual pertence o arquivo
Tamanho do arquivo (em bytes)
Hora da criação
Hora do último acesso
Hora da última modificação

Tabela 10.10 Os campos retornados pela chamada de sistema *stat*.

- Todo processo UNIX tem 3 descritores de arquivos pré-definidos:
  - `0`: Entrada padrão (*standard input*).
  - `1`: Saída padrão (*standard output*).
  - `2`: Erro padrão (*standard error*).

- Chamadas relativas à diretórios

Chamada de sistema	Descrição
s = mkdir(caminho, modo)	Cria um novo diretório
s = rmdir(caminho)	Remove um diretório
s = link(caminho velho, caminho novo)	Cria uma ligação para um arquivo
s = unlink(caminho)	Remove a ligação para um arquivo
s = chdir(caminho)	Troca o diretório atual
dir = opendir(caminho)	Abre um diretório para leitura
s = closedir(dir)	Fecha um diretório
entradir = readdir(dir)	Lê uma entrada do diretório
rewinddir(dir)	Rebobina um diretório de modo que ele possa ser lido novamente

Tabela 10.11 Algumas chamadas de sistema relacionadas a diretórios. O código de retorno *s* é –1 se ocorrer algum erro; *dir* identifica uma cadeia de diretórios e *entradir* é uma entrada de diretório. Os parâmetros são autoexplicativos.

UFABC

- Exercício 1 Leitura e escrita em arquivos.
  1. Compile e execute o programa `prog1.c`. Digite sequências de caracteres no terminal seguido de ENTER. O que o programa faz? Por que ele não tem include? Por que não é preciso abrir os descritores da entrada e saída padrão?

UFABC

- Exercício 2 Remoção de arquivos.
  1. Compile e execute o programa `prog2.c`. Digite uma sequência de caracteres seguido de ENTER. O que acontece com o arquivo `temp.txt` no diretório corrente?
  2. Crie um *hard link* do arquivo `temp.txt` com o comando `ln temp.txt clone.txt`. Em seguida, entre com o comando `ls -i *.tx`. `temp.txt` e `clone.txt` São o mesmo arquivo? Justifique.
  3. Digite o comando `unlink` na entrada de `prog2.c`. O que aconteceu? O arquivo `clone.txt` ainda existe? Justifique.
  4. Digite uma sequência de caracteres seguida de ENTER. O que acontece com o arquivo `clone.txt` no diretório corrente? Por que é ainda possível ler o arquivo completo?
  5. Em que outras situações é interessante esse comportamento do `unlink`?

UFABC

- Exercício 3 Uso de buffer
  1. Compile e execute o programa `prog3.c` com o seguinte comando `/usr/bin/time -v CMD`, em que `CMD` é o arquivo a ser executado. Verifique os gastos de tempo de CPU de saídas para o sistema de arquivos, além do número de trocas de contexto voluntárias.
  2. Compile e execute o programa `prog4.c` com o seguinte comando `/usr/bin/time -v CMD`, em que `CMD` é o arquivo a ser executado. Verifique os gastos de tempo de CPU e de saídas para o sistema de arquivos, além do número de trocas de contexto voluntárias. Por que o tempo da execução desse programa é muito maior do que o tempo de execução do programa anterior?

UFABC

- Exercício 4 Obtenção dos atributos de um arquivo.
  1. Compile, ignorando os avisos da compilação, e execute o programa `prog5.c` passando como parâmetro um arquivo qualquer do sistema de arquivos. Que informações ele me retorna? Guarde a saída da execução em diferentes tipos de arquivos e compare o resultado.
- Exercício 5 Comando `stat`.
  1. O comando `stat` tem função similar ao `prog5.c`. Repita a execução com esse comando, usando como entrada os mesmos arquivos do Exercício 4.1.

UFABC

- Exercício 6 Lista de arquivos em um diretório.
  1. Compile e execute o programa `prog6.c` passando como parâmetro de entrada diversos diretórios do sistema de arquivos. Com base nas saídas, explique como funciona a organização dos arquivos em um diretório no Linux.