GSI018 – Sistemas Operacionais

Sistemas de Arquivos no Linux

Universidade Federal de Uberlândia
Faculdade de Computação
Prof. Dr. rer. nat. Daniel D. Abdala

Na Aula Anterior...

Nesta Aula

- O Sistema de Arquivos EXT e suas versões;
- Característica Gerais do EXT2;

Sistemas de Arquivos – Linux

- A primeira versão do Linux utilizava o sistema de arquivos do MINIX;
- A medida que o Linux foi se desenvolvendo o EXT – Extended File System foi introduzido;
- Ele implementava várias otimizações em relação ao sistema de arquivos do MINIX, mesmo assim seu desempenho era insatisfatório;
- O EXT2, introduzido em 1994, incluia várias novas características;
- Ele era eficiente e robusto e se tornou o sistema de arquivos mais amplamente utilizado em sistemas baseados no Linux;

EXT2 e Suas Versões Posteriores

- Como veremos o EXT2 foi idealizado com o objetivo de maximizar a consistência do sistema de arquivos;
- Duas novas versões do EXT foram lançadas:
  - EXT3 – 2001 – implementa **journaling**;
  - EXT4 – 2008 – expande alguns limites de tamanho de arquivos e número de diretórios do EXT3;

Características Gerais do EXT2

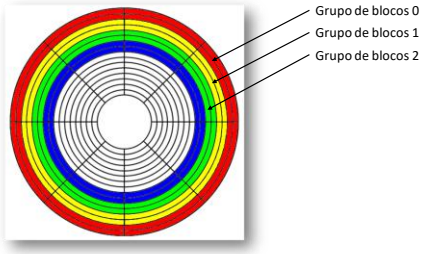
- Implementa muitas das boas práticas encontradas em diversos sistemas de arquivos modernos;
- Ele implementa a interface de Syscalls proposta no padrão POSIX;
- O tamanho dos blocos de dados pode ser configurado pelo gerente do sistema no momento da criação do sistema de arquivos;
- Os blocos podem ser de 1 a 4kB (1.024 a 4.096 bytes);

Características Gerais do EXT2

- Os blocos de dados são organizados lógica e físicamente em grupos;
- Cada grupo inclui blocos de dados e i-nodes armazenados contíguamente em trilhas adjacentes;
- O sistema de arquivos busca alocar um arquivo, ou menor, os blocos que compõem o arquivo no mesmo grupo de blocos;
- Esta forma de organização em que um arquivo é armazenado no mesmo grupo de blocos o que diminui o tempo médio de busca do arquivo;

7

Grupos de Blocos



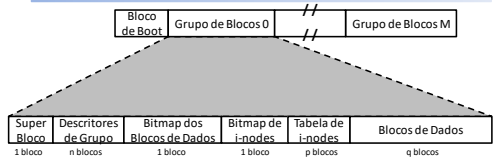
8

Características Gerais do EXT2

- O sistema de arquivos pré-aloca blocos de dados para arquivos regulares antes que eles sejam efetivamente utilizados;
- Quando o tamanho do arquivo aumenta, já há blocos pré-alocados em posições adjacentes;
- O EXT2 implementa uma estratégia de atualização de arquivos com o objetivo de minimizar o impacto de falhas do sistema;

9

Estrutura Geral de uma Unidade de Armazenamento Usando EXT2



- Todos os grupos de blocos têm o mesmo tamanho (nº de blocos) e são armazenados sequencialmente;
- Ambos **superbloco** e **descritores de grupos** são replicados em todos os grupos;
- Apelas os do grupo zero são usados pelo SO;
- Os demais atual como cópias de segurança. Programas de verificação de consistência os usam para consistência do sistema de Arquivos;

10

Superbloco

- Um superbloco do sistema de arquivos EXT2 utiliza um bloco;
- Este é um dos motivos pelo qual os blocos não podem ser menores que 1024 bytes;
- A informação contida no superbloco é representada por uma estrutura de dados chamada **`ext2_super_block`**;
- Ela contém informação de configuração e manutenção do sistema de arquivos;
- A estrutura é extensa e complexa, a seguir são listados alguns dos principais campos;

11

Superbloco

tipo	campo	função
uint32	s_inodes_count	nº total de i-nodes
uint32	s_blocks_count	nº total de blocos
uint32	s_free_blocks_count	contador de blocos livres
uint32	s_free_inodes_count	contador de i-nodes livres
uint32	s_log_block_size	tamanho do bloco
uint32	s_blocks_pre_group	nº de blocos por grupo
uint32	s_inodes_per_group	nº de i-nodes por grupo
uint32	s_mtime	hora da última vez que o sa foi montado
uint32	s_wtime	hora da última vez que o sa foi escrito
uint16	s_mnt_count	nº de vezes que o sa foi montado antes da checagem de consistência
uint32	s_lastcheck	hora em que o sa foi checado pela última vez

12

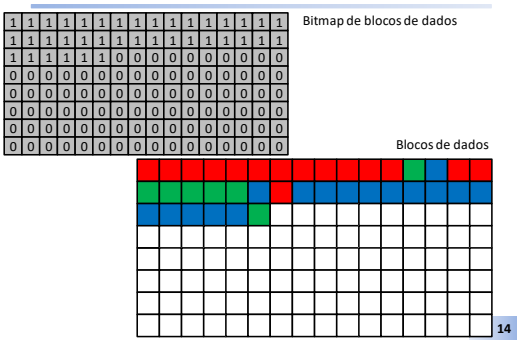
Descritores de Grupos

- Cada grupo de blocos tem seu próprio descritor de grupos;
- O descritor de grupos é representado pela estrutura de dados **ext2_group_desc**;

tipo	campo	função
uint32	Bg_block_bitmap	Nº do bloco do bitmap de blocos
uint32	Bg_inode_bitmap	Nº do bloco do bitmap de inodes
uint32	Bg_inode_table	Nº do bloco do 1º bloco da tabela de i-nodes
Uint16	Bg_free_blocks_count	Nº de blocos livres no grupo
Uint16	Bg_free_inodes_count	Nº de i-nodes livres no grupo
Uint16	Bg_used_dirs_count	Nº de diretórios no grupo
Uint16	Bg_pad	Alinhamento para palavra de 32 bits
Uint32[3]	Bg_reserved	-----

13

Bitmap de Blocos de Dados



14

Quantidade de Grupos de Blocos

- Depende do tamanho da partiçãoe do tamanho do bloco de dados;
- A maior restrição é o bitmap de blocos que é usado para identificar os blocos usados e livres dentro de um grupo;
- Um bit por bloco de dados;
- Deve ser armazenado em um único bloco;

  |blocos no grupo| = 8*b (blocos)
- Onde **b** é o tamanho do bloco em bytes;

15

Exemplo

- Partição: 8 GB
- Bloco: 4 KB
- |blocos no grupo| = $8 * 4 * 2^{10}$
- |blocos no grupo| = 32.768 (blocos)
- Total de bytes de dados em um grupo é igual a $32.768 * 4 * 2^{10}$ = 134.217.728 bytes = 128 MB
- 8GB/128MB = $8*2^{30}/128*2^{20}$ = 64 grupos

16

Tabela de i-nodes

- Série de blocos consecutivos cada um dos quais contém uma quantidade pré-definida de i-nodes;
- O nº do 1º bloco da tabela de i-nodes é armazenado no campo **bg_inode_table** da estrutura de dados **ext2_group_desc**;
- Todos os i-nodes têm o mesmo tamanho, 128 bytes;
- Um bloco de 1KB contém 8 inodes e um de 4k contém 32 i-nodes;

17

Tabela de i-nodes

- O nº de blocos utilizados pela tabela de i-nodes é computado de acordo com a fórmula abaixo:

$$|Table| = \frac{|i - nodes_{grupo}|}{|i - nodes_{bloco}|}(bloco s)$$

- O nº de i-nodes por grupo é um parâmetro definido no superbloco e define qual o número máximo de arquivos um grupo pode conter. Ele pode ser acessado no campo **s_inodes_per_group** do superbloco;

18

i-nodes

- Cada i-node do EXT2 é representado pela estrutura **`ext2_inode`**;
- Compatível com a especificação POSIX;
- O Campo **`i_size`** de 32 bits limita o tamanho máximo a 4GB; (**Na verdade 2GB pois o bit mais significativo não é usado**)
- O campo **`i_block`** é um array de ponteiros **`EXT2_N_BLOCKS`** (usualmente 15) para blocos usados para identificar blocos de dados do arquivo;

19

i-node

tipo	campo	função
Uint16	i_mode	Tipo de arquivo e direitos de acesso
Uint16	i_uid	Identificador do proprietário
Uint32	i_size	Tamanho do arquivo em bytes
Uint32	i_atime	Hora do último acesso ao arquivo
Uint32	i_ctime	Hora em que o i-node foi alterado (última)
Uint32	i_mtime	Hora que o conteúdo do arquivo foi alterado (última)
Uint32	i_dtime	Hora em que o arquivo foi deletado
Uint16	i_gid	Identificador de grupo
Uint16	i_links_count	Contador do nº de links
Uint32	i_blocks	Nº de blocos de dados do arquivo
Uint32	i_flags	Flags do arquivo

20

i-node

tipo	Campo	função
Union	Osd1	
Uint32 [EXT2_N_BLOCKS]	i_block	Ponteiros para os blocos de dados
Uint32	i_version	Versão do arquivo (para NFS)
Uint32	i_file_acl	Lista de controle de acesso ao arquivo
Uint32	i_dir_acl	Lista de controle de acesso ao diretório
Uint32	i_faddr	Endereço do fragmento
Union	Odl2	

21

Utilização de Blocos por Diferentes Tipos de Arquivos

- Diferentes tipos de arquivos utilizam blocos de dados de maneira distinta:

tipo	Descrição
0	Desconhecido
1	Arquivo regular
2	diretório
3	Dispositivo de caracteres
4	Dispositivo de blocos
5	Pipe nomeado
6	Socket
7	Link simbólico

22

Arquivos Regulares

- Necessita de blocos de dados apenas quando dados começam a ser inseridos;
- Quando criado, um arquivo está vazio e não necessita de blocos de dados;
- Arquivos podem ser esvaziados pela chamada do sistema **`truncate()`**;

23

Diretórios

- Diretórios são um tipo especial de arquivo;
- Blocos de dados armazenam nomes de arquivos associados aos ponteiros para os i-nodes destes;
- Tais blocos de dados contêm estruturas do tipo **`ext2_dir_entry_2`**;

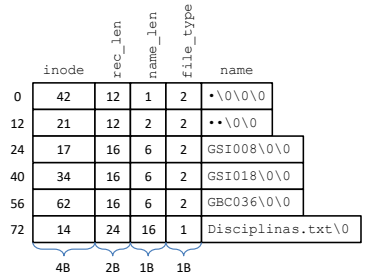
24

Diretórios

tipo	Campo	função
uint32	inode	nº do i-node
uint16	rec_len	comprimento da entrada do diretório
uint8	name_len	comprimento do nome do arquivo
uint8	file_type	tipo do arquivo
char[EXT2_NAME_LEN]	name	nome do arquivo

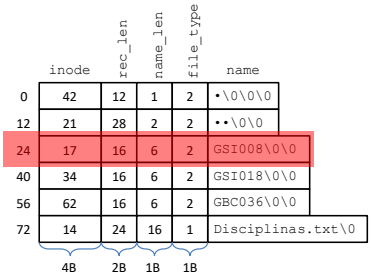
- `EXT2_NAME_LEN` → usualmente 255;
- `name` → sempre múltiplo de 4 por razões de eficiência;

Exemplo Diretório



	inode	rec_len	name_len	file_type	name
0	42	12	1	2	.\0\0\0
12	21	12	2	2	..\0\0
24	17	16	6	2	GSI008\0\0
40	34	16	6	2	GSI018\0\0
56	62	16	6	2	GBC036\0\0
72	14	24	16	1	Disciplinas.txt\0
	4B	2B	1B	1B	

Exemplo Diretório Deletado



	inode	rec_len	name_len	file_type	name
0	42	12	1	2	.\0\0\0
12	21	28	2	2	..\0\0
24	17	16	6	2	GSI008\0\0
40	34	16	6	2	GSI018\0\0
56	62	16	6	2	GBC036\0\0
72	14	24	16	1	Disciplinas.txt\0
	4B	2B	1B	1B	

Links Simbólicos

- Se o caminho do link simbólico tem até 60 caracteres ele é salvo no i-node;
- Se o caminho do link simbólico tem mais de 60 caracteres um bloco de dados é requerido;

Arquivo de Dispositivo, Pipes e Sockets

- Nenhum bloco de dados é requerido para arquivos de dispositivo, piles ou sockets;
- Toda a informação necessária é salva diretamente no i-node;

Bibliografia

- Capítulo 17
- Páginas 451 – 475