Trabalho 3
Sistema de Arquivos

Trabalho 3: Sistema de Arquivos

- Objetivo: Implementar um sistema de arquivos simples do tipo Unix com uma estrutura hierárquica de diretórios.
- Leia a especificação do projeto para detalhes.
- O código fonte está no AVA.
- Comece o quanto antes. Este é um projeto longo e você terá de escrever bastante código.

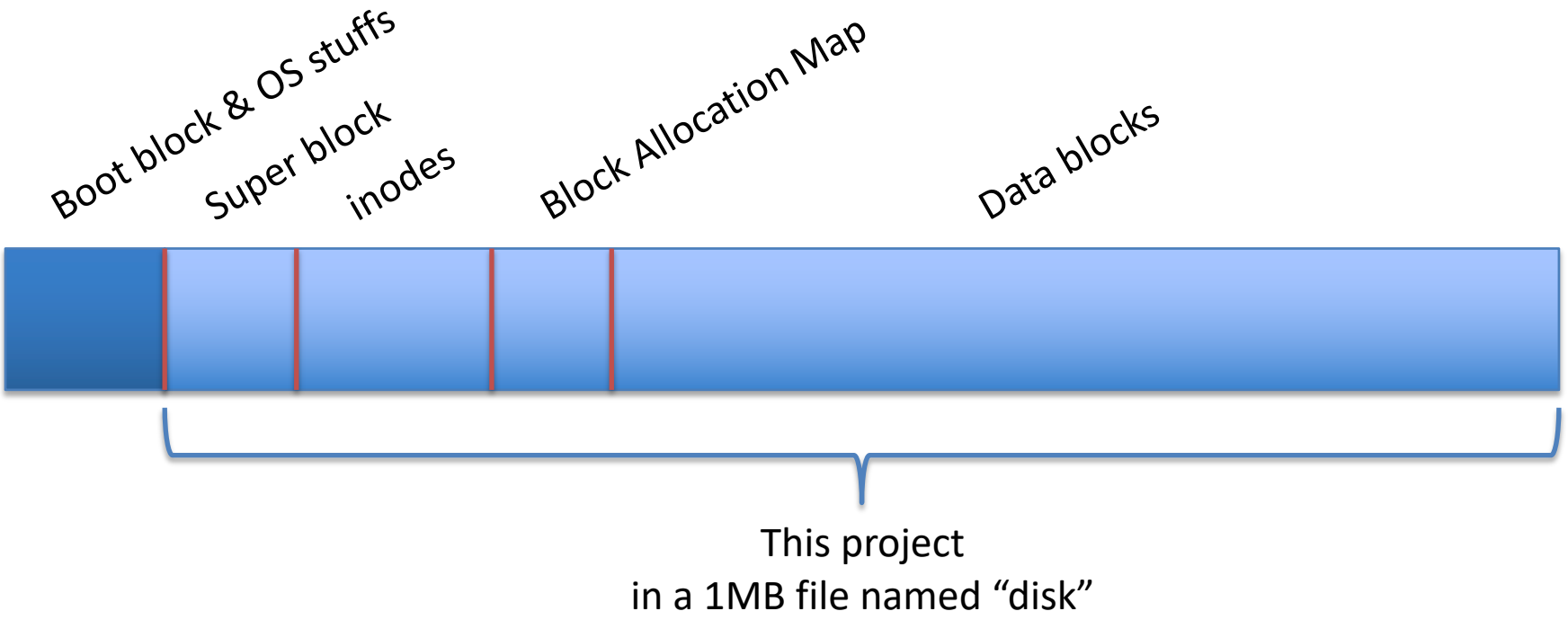
Trabalho 3: Visão Geral

- Implementar um sistema de arquivos simples do tipo Unix com uma estrutura hierárquica de diretórios.
- Gerenciar espaço em disco, pois arquivos e diretórios podem crescer e diminuir.
- Implementar comandos e chamadas de sistema para navegar na estrutura de diretórios, criar novos arquivos e diretórios, removê-los, etc.
- Não precisa se preocupar com concorrência, permissões ou alto desempenho.

API

- Formatação do disco
- Arquivo
  - open, close, read, write, seek
  - link e unlink (apaga um arquivo)
  - stat
- Diretório
  - make, remove, stat, etc.
- Comandos da shell
  - ls e chdir (cd)

Disk Layout

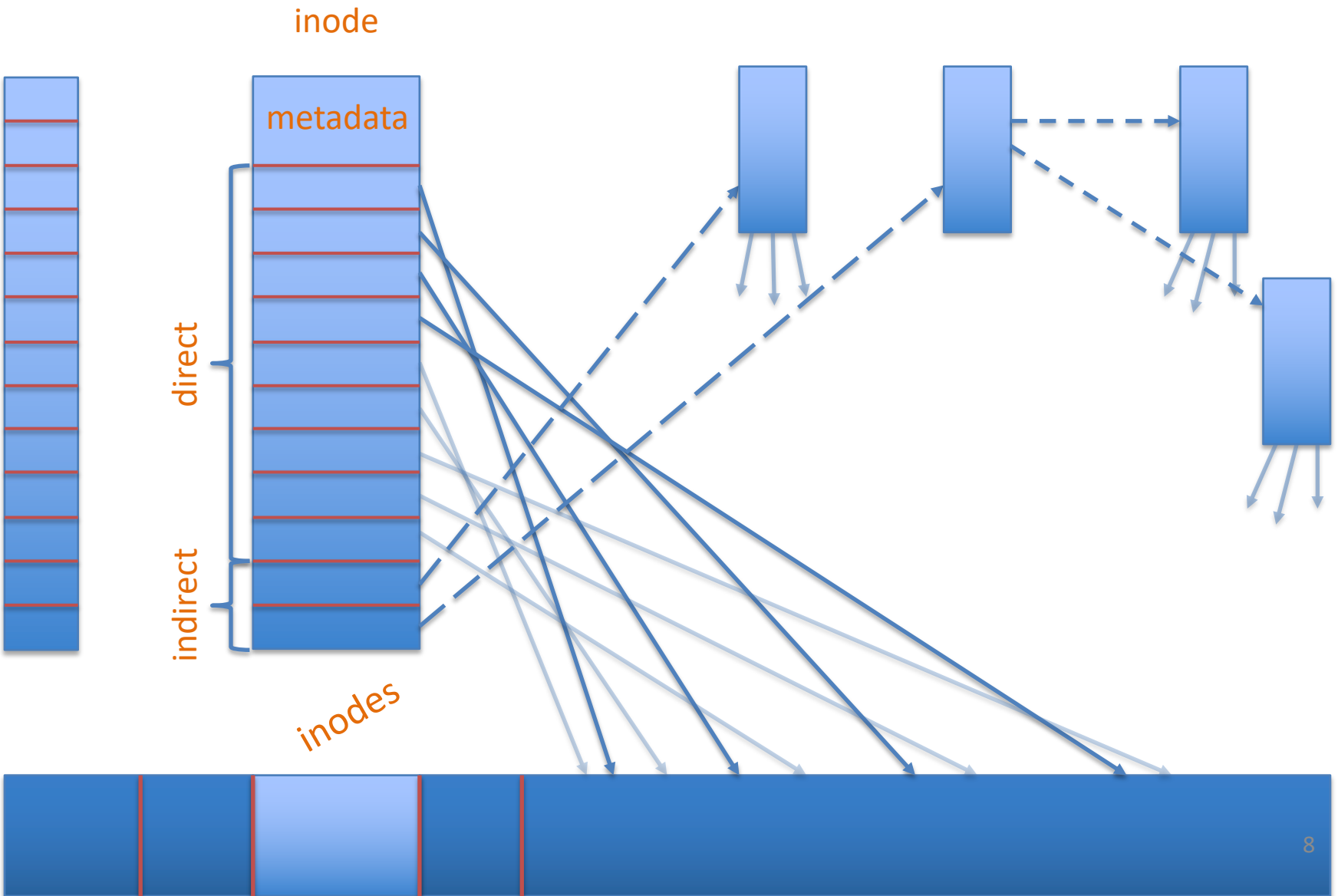


Space between divisions is not representative of actual size.

Superblock – Disk Metadata

- Exemplos:
  - Tamanho
  - # de inodes
  - # de blocos de dados
  - Onde começam os inodes
  - Onde começam os blocos de dados
  - Número mágico (magic number)

Inodes



Inode – Metadata

- Exemplos:
  - Arquivo ou diretório?
  - Contador de links
  - Tamanho
  - etc..

fs_init

- Um “constructor” para o código do FS.
- Chame block_init() para inicializar o “dispositivo” de bloco.
- Inicialize estruturas de dados e recursos usados pelo sistema de arquivos.
- Formate o disco ou monte-o se ele não estiver formatado (crie um mecanismo para detectar se o disco está formatado).

fs_mkfs

- “Cria” um sistema de arquivo:
  - Escreve o superbloco;
  - Marca inodes e blocos de dados como livres;
  - Cria o diretório raiz;
  - Inicializa a tabela de descritores de arquivo.

Criação e Remoção de Arquivos

- fs_open(), fs_link(), fs_unlink().
- open: cria um novo arquivo se ele não existe.
- link: hard link para um arquivo
  - Cria um link para um arquivo existente
- unlink:
  - Apaga o arquivo se o contador de links == 0;
  - Apaga entrada de diretório;
  - Comportamento especial se o arquivo ainda estiver aberto (veja a descrição do projeto).

Acesso a Arquivo

- open: abre um arquivo existente (aloque descritor de arquivo).
- read: lê bytes de um arquivo aberto.
- write: escreve bytes para um arquivo aberto.
- lseek: muda posição em um arquivo.
- close: libera descritor de arquivo.

Semântica de fs_lseek()

- Neste projeto fs_lseek() recebe apenas dois argumentos:
  - Descritor de arquivo e offset.
- Unix lseek() recebe três argumentos:
  - Descritor de arquivo, offset, whence.
- Whence: SEEK_SET, SEEK_CUR, SEEK_END.
- ls_lseek() assume SEEK_SET.
- O que acontece se lseek() quiser posicionar além do final do arquivo? (veja a descrição do projeto para o comportamento esperado)

Diretórios – Parte 1

- Como um arquivo: lista de arquivos e diretórios:
  - Mapeamento de nome para inode.
- Pode ser lido como um arquivo:
  - Use as suas funções de E/S (fs_*) para manipulação de diretórios.
- Sempre tem pelo menos duas entradas:
  - “.” diretório corrente;
  - “..” diretório pai.

Diretórios – Parte 2

- mkdir: cria um diretório.
  - Cria uma entrada no diretório pai;
  - Cria dois diretório: “.” e “..”.
- rmdir: remove diretório se estiver vazio.
- cd: muda o diretório corrente
  - Para nomes de caminho relativos apenas.

Exemplo – fs_mkdir

```
int fs_mkdir(char *file_name)
{
     if (file_name exists) return ERROR;
     // allocate inode
     // allocate data blocks
     // set directory entries for “.” and
“..”
     // set inode entries appropriately
     // update parent
     return SUCCESS
}
```

Diversos

- Você não precisa implementar nomes de caminho absolutos.
- Você não precisa implementar remoção recursiva de diretórios.
- Implemente uma ferramenta para verificar a integridade do sistema de arquivo (fsck) para depuração que verifica a integridade de:
  - Número mágico do superbloco;
  - Alocações de blocos;
  - Alocações de inodes;
  - Mapa de alocação de blocos;
  - Conteúdo de diretório;
  - etc.

Implementação

- Em Linux:
  - Usa um arquivo para simular um disco
  - O código para isso é fornecido
  - Execute ./lnxsh
- Shell suporta:
  - Chamadas de sistema para sistema de arquivos
  - Comandos: “ls”, “cat foo”, “create foo 200”
- Você terá de escrever bastante código:
  - Em torno de 1000 linhas de código.

Teste

- Um script em python é fornecido.
- Vários testes que:
  - Executa a shell;
  - Abre um sistema de arquivos existente (ou formata um novo);
  - Escreve comandos para a shell (cat foo);
  - Lê saída da shell (ABCDEF);
  - exit.
- Você também deve escrever seus próprios casos de teste:
  - Submeta-os com o seu código.