



Simulador de Sistemas de Arquivos

Trabalho Prático 1

Disciplina: Sistemas Operacionais – 2008/1
Professor Humberto César Brandão de Oliveira

O trabalho possui dois requisitos fundamentais:

- Entrega do código fonte (zipado): *nomeDoAluno.zip*;
- Entrega do relatório (pdf): *nomeDoAluno.pdf*;

Ambos devem ser entregues no e-mail: humberto@dcc.ufmg.br

Título do e-mail: [SO_2008_1] TP1 – Entrega Final – “Nome do aluno”

Exemplo: [SO_2008_1] TP1 – Entrega Final – Humberto Brandão

Data da entrega final: 18/04/2008

Data das entregas parciais: 04/04/2008 e 11/04/2008 (detalhes das entregas parciais estão no site)

Não serão aceitas entregas em material impresso.

Observações sobre o trabalho:

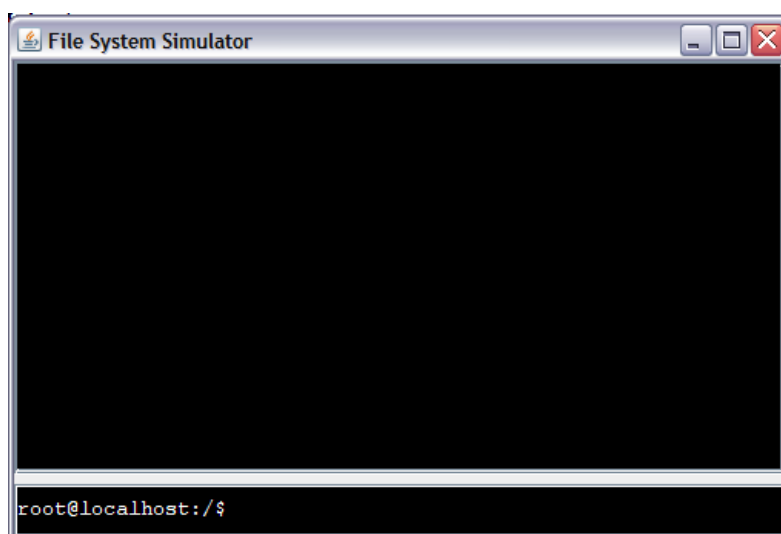
- Todo aluno usará o código disponibilizado no site como base de seu trabalho;
- O código disponibilizado no site da disciplina fornece a interface gráfica para utilização do simulador de sistema de arquivos;
- O aluno deve implementar as funções do *minikernel* do S.O. simulado. Tais funcionalidades serão alcançadas implementando as funções de interface da classe *Kernel.class* (*operatingSystem.Kernel.class*), também disponibilizada no site. O nome da classe do aluno que implementa *Kernel.class* deve ser *MyKernel.java*; Já existe um protótipo desta classe no código disponibilizado. Opcionalmente, o aluno poderá utilizar a classe incompleta disponibilizada, implementando apenas as funcionalidades do *minikernel*.
- A organização do sistema de arquivos simulado, classes utilizadas, etc, é de total responsabilidade do aluno. Tal organização, incluindo estrutura, decisões de projeto, etc, serão base para avaliação do trabalho;
- Para ligação entre o *minikernel* do aluno (*MyKernel.java*), e a classe de interface *kernel.class* disponibilizada, o aluno deve em sua classe principal do projeto, denominada *Main.java*, instanciar um objeto de sua classe *MyKernel* (que implementa *kernel.java* do pacote) e passar no construtor do sistema de arquivos (*FileSytemSimulator.java*); A classe *Main.java* também está disponível no código para *download* no site.



Main.java

```
import operatingSystem.fileSystem.FileSystemSimulator;  
import operatingSystem.Kernel;  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        //instanciando o kernel definido pelo aluno  
        Kernel k = new MyKernel();  
  
        //instanciando o simulador de Sistema de Arquivos  
        new FileSystemSimulator( k ).setVisible( true );  
  
    }  
  
}
```

- A execução da classe *Main.java*, utilizando o código disponibilizado deve abrir o Simulador de Sistema de Arquivos:



- A execução correta das funcionalidades, depende exclusivamente da implementação da classe *MyKernel* e de outras classes que o aluno julgar implementar;

Chamadas de Sistema que devem ser implementadas na classe *MyKernel*:

Documentação *javadoc*:

Resumo dos Métodos

java.lang.String	batch (java.lang.String parameters) Funcao para executar um arquivo de lote.
java.lang.String	cat (java.lang.String parameters) Funcao para navegação entre diretorios.



java.lang.String	cd (java.lang.String parameters) Funcao para navegacao entre diretorios.
java.lang.String	chmod (java.lang.String parameters) Funcao para definir permissao de arquivos e diretorios
java.lang.String	cp (java.lang.String parameters) Funcao para copiar arquivos e diretorios.
java.lang.String	createfile (java.lang.String parameters) Funcao para criar arquivos.
java.lang.String	info () Função para indicar informacoes do simulador.
java.lang.String	ls (java.lang.String parameters) Funcao para listar diretorios e arquivos.
java.lang.String	mkdir (java.lang.String parameters) Funcao para criar diretorio.
java.lang.String	mv (java.lang.String parameters) Funcao para mover arquivos e diretorios.
java.lang.String	rm (java.lang.String parameters) Funcao para remover arquivos e diretorios.
java.lang.String	rmdir (java.lang.String parameters) Funcao para remover diretorio vazio.

```
public String batch(String parameters)
```

O comando `batch` recebe como parâmetro o endereço completo de um arquivo situado no sistema de arquivos real do computador, e executa todos os comandos deste arquivo texto no sistema de arquivo simulado. Os comandos são separados pelo caractere quebra de linha `\n`.

A string retornada apenas indicará se o arquivo passado por parâmetro existe ou não. Se existir, a chamada de sistema deverá executar todos os comandos do arquivo texto indicado.

Exemplo de comando:

```
//exemplo de execucao de arquivo em lote  
aluno@localhost:/$ batch c:\humberto\comandos.txt  
Comandos executados.
```

```
//exemplo de falha na execucao de arquivo em lote  
aluno@localhost:/$ batch c:\humberto\ocmandos.txt  
Arquivo não existe.
```

Este comando será utilizado para executar um conjunto de testes para avaliação final do trabalho.

Um exemplo de arquivo de lote é disponibilizado na página da disciplina para testes. O aluno poderá criar outros arquivos para validar seu trabalho.



A variável parâmetro da função (*parameters*) recebe apenas a string com o caminho do arquivo. O tratamento da string é de responsabilidade do aluno.

```
public String cat(String parameters)
```

O comando `cat` recebe como parâmetro o endereço de um arquivo do sistema de arquivos simulado, e lista o conteúdo deste arquivo texto, se o mesmo existir.

Exemplo de comando: caminho completo e caminho relativo.

```
//exemplo de visualização utilizando caminho completo
aluno@localhost:/$ cat /home/aluno/disciplina.txt
Sistemas Operacionais
Trabalho Prático 1
aluno@localhost:/home/aluno$
```

```
//exemplo de visualização utilizando caminho relativo
aluno@localhost:/home/aluno$ cat disciplina.txt
Sistemas Operacionais
Trabalho Prático 1
aluno@localhost:/home/aluno$
```

```
//exemplo de erro
aluno@localhost:/home/aluno$ cat disciplina2.txt
cat: Arquivo não existe.
aluno@localhost:/home/aluno$
```

Observação: Todos os arquivos do TP1 são do tipo texto. Não existe arquivo binário no simulador.

A variável parâmetro da função (*parameters*) recebe apenas a string com o caminho do arquivo (completo ou relativo). O tratamento da string é de responsabilidade do aluno.

```
public String cd(String parameters)
```

O comando `cd` é responsável pela navegação na estrutura de árvore do sistema de arquivos simulado.

Exemplo de comandos:

```
//exemplo de falha na navegação.
aluno@localhost:/$ cd /aloMundo
/aloMundo: Diretório não existe.
```

```
//exemplo de navegação com sucesso. String vazia é retornada.
aluno@localhost:/$ cd /home
aluno@localhost:/home$ cd ./aluno
aluno@localhost:/home/aluno$ cd ../
aluno@localhost:/home/$ cd ./
aluno@localhost:/home/$
```

A chamada de sistema `cd` recebe apenas um parâmetro. O diretório a ser acessado. Este pode possuir caminho completo, começando por “/”, ou apenas o caminho relativo. Como no comando apresentado anteriormente: `cd aluno`. Esta opção efetua acesso de acordo com o local atual do apontador de diretório, que neste caso era na pasta `/home`. Portanto, o diretório acessado foi o `/home/aluno`.



É necessária a implementação dos ponteiros `.` e `..`, que apontam para o diretório corrente, e para o diretório pai, respectivamente.

Para alterar o nome do diretório atual, impresso no *shell*, utilize a variável estática:

```
operatingSystem.fileSystem.FileSystemSimulator.currentDir
```

Exemplo:

```
operatingSystem.fileSystem.FileSystemSimulator.currentDir = "/home/aluno";
```

O comando `cd` será utilizado para verificar a corretude do trabalho, efetuando a navegação na árvore de diretórios do simulador de sistema de arquivos.

A variável parâmetro da função (*parameters*) recebe apenas a string com o caminho para navegação (completo ou relativo). O tratamento da string é de responsabilidade do aluno.

```
public String chmod(String parameters)
```

O comando `chmod` é responsável pela definição de permissões de arquivos e ou diretórios.

A string retornada pela função deve ser o conteúdo do diretório a ser listado, ou o indicativo que o diretório não existe.

Exemplo de comandos `chmod`:

```
//exemplo de listagem do conteúdo do diretório, para visualizar as permissões.
aluno@localhost:/home/aluno$> ls -l
-rw-r--r--  aluno Jan 24 14:41 NewMain.java

//alterando a permissão do arquivo NewMain.java.
aluno@localhost:/home/aluno$> chmod 777 ./NewMain.java

//listando novamente o conteúdo do diretório, agora com a permissão alterada.
aluno@localhost:/home/aluno$> ls -l
-rwxrwxrwx  aluno Jan 24 14:41 NewMain.java
```

Esta chamada de sistema pode ter 2 ou 3 parâmetros neste simulador.

Exemplo com dois parâmetros:

```
aluno@localhost:/home/aluno$> chmod 777 ./NewMain.java
1º parâmetro: permissões (usuário, grupo, todos)
2º parâmetro (arquivo ou diretório)
```

Exemplo com 3 parâmetros:

```
aluno@localhost:/home/aluno$> chmod -R 744 ./myFolder
1º parâmetro: indica recursividade no comando. Todos os arquivos e pastas dentro
do diretório myFolder receberão a permissão indicada.
2º parâmetro: permissões (usuário, grupo, todos)
3º parâmetro (diretório)
```

O parâmetro da função `chmod` da classe *MyKernel* denominado "parameters", possuirá em uma única String os parâmetros da chamada de sistema `chmod`. O aluno deverá efetuar o tratamento – separação da String – para verificar a validade dos dois ou três parâmetros passados no *shell*.



public String cp(String parameters)

O comando `cp` copia um arquivo ou um diretório para outro local especificado.

O comando `cp` pode receber dois ou três parâmetros:

1. `-R`, se for copiar um diretório, e todo seu conteúdo para outro local;
2. Diretório ou arquivo de origem;
3. Diretório ou arquivo de destino;

```
//exemplo de copia, mudando o nome do arquivo, seguida de listagem de arquivo.  
aluno@localhost:/home/aluno$> cp ./NewMain.java ../NewMain2.java  
aluno@localhost:/home/aluno$> cd ../  
aluno@localhost:/home/$> ls -l  
-rw-r--r--  aluno Jan 24 14:41 NewMain2.java
```

```
//exemplo de copia, sem mudar o nome, seguida de listagem de arquivo.  
aluno@localhost:/home/aluno$> cp ./NewMain.java ../  
aluno@localhost:/home/aluno$> cd ../  
aluno@localhost:/home/$> ls -l  
-rw-r--r--  aluno Jan 24 14:41 NewMain.java
```

```
//exemplo de copia com 3 parâmetros  
aluno@localhost:/home/aluno$> cp -R ./humberto /  
aluno@localhost:/home/aluno$> ls /  
home    humberto    etc
```

```
//exemplo de erro ao copiar  
aluno@localhost:/home/aluno$> cp -R ./humberto /pastaQueNaoExiste  
cp: O directorio de destino não existe. (Nada foi copiado)
```

```
//exemplo 2 de erro ao copiar  
aluno@localhost:/home/aluno$> cp -R /pastaQueNaoExiste ./humberto  
cp: O directorio de origem não existe. (Nada foi copiado)
```

```
//exemplo 3 de erro ao copiar  
aluno@localhost:/home/aluno$> cp -R ./disciplina2.txt /home  
cp: O arquivo não existe. (Nada foi copiado)
```

A string `parameters` neste caso deverá ter os dois ou três parâmetros informados no *shell*. Exemplo: `"-R ./disciplina2.txt /home"`. O tratamento da string é de responsabilidade do aluno.

public String createfile(String parameters)

O comando `createfile` é utilizado para criar arquivos texto. Recebe como entrada dois parâmetros. O primeiro indica o caminho do arquivo, podendo ser caminho completo ou caminho relativo. O segundo parâmetro é o conteúdo do arquivo texto.

Exemplo de comando: caminho completo e caminho relativo.

```
//criando arquivo texto usando caminho completo  
aluno@localhost:/ $ createfile /home/aluno/disciplina.txt Sistemas  
Operacionais\nTrabalho Pratico 1
```

```
//visualizando arquivo criado  
aluno@localhost:/ $ cat /home/aluno/disciplina.txt  
Sistemas Operacionais
```



Trabalho Prático 1

--

```
//criando arquivo texto usando caminho relativo
aluno@localhost:/home/aluno$ createfile ./disciplina.txt Sistemas
Operacionais\nTrabalho Pratico 1
aluno@localhost:/home/aluno$

//visualizando arquivo criado utilizando caminho completo
aluno@localhost:/home/aluno$ cat /home/aluno/disciplina.txt
Sistemas Operacionais
Trabalho Prático 1
aluno@localhost:/home/aluno$

//visualizando arquivo criado utilizando caminho relativo
aluno@localhost:/home/aluno$ cat ./disciplina.txt
Sistemas Operacionais
Trabalho Prático 1
aluno@localhost:/home/aluno$

//exemplo de erro ao criar arquivo
aluno@localhost:/home/aluno$ createfile ./disciplina.txt Sistemas
Operacionais\nTrabalho Pratico 1
createfile: Arquivo já existe. Não foi possível cria-lo
aluno@localhost:/home/aluno$
```

Se o arquivo já existir, uma mensagem de erro deve ser retornada ao usuário. Neste sistema de arquivos simulado, não é possível editar arquivos criados.

A variável parâmetro da função (*parameters*) recebe apenas a string com o caminho do arquivo e o conteúdo textual do novo arquivo. Exemplo: “./disciplina.txt Sistemas Operacionais\nTrabalho Pratico 1”. O tratamento da string é de responsabilidade do aluno.

```
public String info()
```

O comando `info` é responsável por imprimir informações do trabalho no *shell*:

- Nome do aluno;
- Número de Matrícula;
- Versão do kernel (para controle individual do aluno);

```
public String ls(String parameters)
```

O comando `ls` é responsável pela visualização do conteúdo de um diretório.

A string retornada deve ser o conteúdo do diretório a ser listado, ou o indicativo que o diretório não existe.

Exemplo de comandos `ls`:

```
//conteúdo do diretório atual
root@localhost:~/ $ ls
NewMain.java      index.htm          rcssserver-11.1.2
OCI.zip           mail              suzuki.lapo
dead.letter       monitor

//listando o conteúdo do diretório atual
root@localhost:~/ $ ls -l
```



```
-rw-r--r-- Jan 24 14:41 NewMain.java
-rw-r--r-- Dec 20 15:06 OCI.zip
-rw----- Mar 18 2007 dead.letter
-rw-r--r-- Sep 14 2007 index.htm
drwx----- Mar 16 2007 mail
drwxr-xr-x Nov 26 02:01 monitor
drwxrwxrwx Nov 25 20:38 rcssserver-11.1.2
-rw-r--r-- Jan 24 14:45 suzuki.lapo
```

```
//listando o conteúdo do diretório especificado
root@localhost:/$ ls /home
func grad pos prof vip aluno
```

O comando `ls` pode receber dois, um ou nenhum parâmetro(s):

Nenhum: visualizar conteúdo do diretório atual;

Um: Pode ser a opção de listar o conteúdo `-l`, ou poder ser o caminho do diretório a ser listado.

Dois: `-l` e caminho do diretório a ser listado. Exemplo: `ls -l /home`

O parâmetro da função `ls` denominado “parameters”, possuirá em uma única string os parâmetros opcionais da função `ls`. Esta única string receberá, por exemplo, os dois parâmetros “`-l /home/alunos`”. O tratamento da string é de responsabilidade do aluno.

O comando `ls` será utilizado para verificar a corretude do trabalho, após uma sequência de comandos efetuados no *shell*.

```
public String mkdir(String parameters)
```

O comando `mkdir` é responsável pela criação de diretórios. A criação pode ser feita de duas formas. Utilizando o caminho completo, e o caminho relativo.

```
//exemplo de criação e acesso do diretório humberto
root@localhost:/$ mkdir ./humberto
root@localhost:/$ ls
humberto home etc
root@localhost:/$ cd ./humberto
root@localhost:/humberto$
```

```
//exemplo de erro ao criar diretório
root@localhost:/$ mkdir home
mkdir: home: Diretório já existe (Nenhum diretório foi criado).
```

```
//exemplo de criação de uma pasta dentro de outra já existente.
root@localhost:/home/aluno/$ mkdir ./humberto/notas
root@localhost:/home/aluno/$ cd ./humberto
root@localhost:/home/aluno/humberto$ cd ./notas
root@localhost:/home/aluno/humberto/notas$
```

```
//exemplo de criação e acesso utilizando o caminho completo.
root@localhost:/home/aluno/$ mkdir /home/aluno/humberto/notas
root@localhost:/home/aluno/$ cd /home/aluno/humberto/notas
root@localhost:/home/aluno/humberto/notas$
```

A string `parameters` neste caso deverá ter somente o nome do diretório a ser criado. O tratamento da string é de responsabilidade do aluno.

```
public String mv(String parameters)
```



O comando `mv` move um arquivo ou um diretório para outro local especificado.

O comando `mv` recebe dois parâmetros.

1. Diretório ou arquivo de origem;
2. Diretório ou arquivo de destino.

O comando `mv` pode ser utilizado também para renomear arquivos e diretórios.

Exemplos:

```
//exemplo de movimentação de diretório
aluno@localhost:/home/aluno$> mv ./humberto ../
aluno@localhost:/home/aluno$> cd ../
aluno@localhost:/home/$> ls
aluno  humberto
aluno@localhost:/home/aluno$>

//renomeando arquivo
aluno@localhost:/home/aluno/$> mv ./disciplina.txt ./disc.txt
aluno@localhost:/home/aluno$> ls
disc.txt
aluno@localhost:/home/aluno$>

//erro ao mover diretorio
aluno@localhost:/home/aluno/$> mv ./humberto ./diretorioQueNaoExiste
mv: Diretorio destino nao existe (Nenhuma alteracao foi efetuada)
aluno@localhost:/home/aluno$>
```

Assim como em outros comandos, podem ser utilizados os caminhos completo e relativo.

A string `parameters` neste caso deverá ter os dois parâmetros informados no *shell*. Exemplo: “./disciplina.txt ./disc.txt”. O tramento da string é de responsabilidade do aluno.

```
public String rm(String parameters)
```

O comando `rm` remove um arquivo ou um diretório.

O comando `rm` pode receber um ou dois parâmetros.

1. -R, se o objeto a ser removido for um diretório; (opcional)
2. Diretório ou arquivo a ser removido;

```
//removendo o arquivo disciplina.txt
aluno@localhost:/home/aluno/humberto$> rm ./disciplina.txt
aluno@localhost:/home/aluno/humberto$>

//removendo o diretório e todo o seu conteúdo
aluno@localhost:/home/aluno$> rm -R ./humberto
aluno@localhost:/home/aluno$>

//exemplo de erro ao remover arquivo
aluno@localhost:/home/aluno/humberto$> rm ./disciplina2.txt
rm: Arquivo nao existe (Nenhum arquivo ou diretório foi removido)
aluno@localhost:/home/aluno/humberto$>
```



```
//exemplo 2 de erro ao tentar remover diretório sem o -R
aluno@localhost:/home/aluno$> rm ./humberto
rm: humberto: é um diretório (Nenhum arquivo ou diretório foi removido)
aluno@localhost:/home/aluno/ $>
```

A string `parameters` neste caso deverá ter os um ou dois parâmetros informados no *shell*. Exemplo: “-R ./humberto”. O tratamento da string é de responsabilidade do aluno.

```
public String rmdir(String parameters)
```

O comando `rmdir` remove diretórios vazios. Diretórios com conteúdo não podem ser removidos por este comando no nosso simulador de sistema de arquivos.

```
//exemplo de remoção de diretório vazio utilizando caminho completo
root@localhost:/home/aluno/$ rmdir /home/aluno/humberto
root@localhost:/home/aluno/$
```

```
//exemplo de erro ao remover diretório
root@localhost:/home/aluno/$ rmdir /home/aluno/mariane
rmdir: Diretório /home/aluno/mariane não existe. (Nada foi removido)
root@localhost:/home/aluno/$
```

```
//exemplo de erro ao remover diretório
root@localhost:/home/aluno/$ rmdir /home/aluno/
rmdir: Diretório /home/aluno/ possui arquivos e/ou diretorios. (Nada foi removido)
root@localhost:/home/aluno/$
```

```
//exemplo de remoção de diretório vazio utilizando caminho relativo
root@localhost:/home/aluno/$ rmdir ./humberto
root@localhost:/home/aluno/$
```

O comando `rmdir` recebe apenas um parâmetro no *shell*.

A string `parameters` neste caso receberá apenas um parâmetro: O diretório a ser removido. Exemplo: “./humberto”. O tratamento da string é de responsabilidade do aluno.



Relatório

- O relatório deve conter informações como:
 - a. Detalhes das Estruturas utilizadas para armazenamento do Sistema de Arquivos;
 - b. Dificuldade no desenvolvimento do trabalho. Teoria e/ou prática;
 - c. Complexidade das operações das chamadas de sistema implementadas (utilizando notação O);
 - d. Sugestões, considerando as estruturas utilizadas, para a atualização que especifica um sistema de arquivos multiusuário;
 - e. Sugestões, considerando as estruturas utilizadas, para a atualização que especifica um sistema de arquivos com acesso concorrente;
 - f. Comentários para as perguntas:
 - i. O sistema de arquivos implementado é eficiente?
 - ii. O sistema de arquivos implementado é confiável?
 - iii. Quais testes foram executados para 'garantir' a confiabilidade do software?
 - iv. Você enxerga diferentes implementações de sistemas de arquivos para diferentes hardwares? Exemplifique.

Considerações:

- Qualquer tipo de consulta a livros, colegas, internet e outras fontes, deve ser incluída no relatório;
- O aluno eventualmente poderá ser chamado para comentar seu trabalho na sala do professor.