大多数文件系统实现将在Fs.c中进行。在这里，您可以定义自己的常量、结构和辅助函数。我们还提供了文件utility.c和utility.h，您可以在其中放置您认为应该放在单独文件中的数据结构和函数。所有代码必须在这三个文件中

**准备任务：**

**FsNew**

FsNew函数的签名如下:

Fs FsNew(void);

此功能应分配和初始化新的**struct FsRep**，创建文件系统的根目录，使根目录成为当前的工作目录。然后，它应返回指向分配的**struct FsRep** 的指针。

**FsGetCwd**

FsGetCwd函数的签名如下:

void FsGetCwd(Fs fs, char cwd[PATH\_MAX + 1]);

这个函数应该在给定的**cwd**数组中存储当前工作目录的规范路径。它可以假设当前工作目录的规范路径不超过**PATH\_MAX**字符。

**FsFree**

FsFree函数的签名如下:

void FsFree(Fs fs);

这个函数应该释放与给定**Fs**关联的所有内存。在处理每个阶段时，您可能需要更新这个函数，以释放您创建的任何新数据结构。

**任务一：**

**FsMkdir**

FsMkdir函数的签名如下:

void FsMkdir(Fs fs, char \*path);

该函数接受一个路径，并在给定文件系统中的该路径上创建一个新目录。**FsMkdir**执行的功能与Linux中的**mkdir**命令大致相同。

**Errors**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| 类型 | Error Message |
| 文件已存在于指定路径 | mkdir: cannot create directory '*path*': File exists |
| 路径的前缀是一个常规文件 | mkdir: cannot create directory '*path*': Not a directory |
| 路径的正确前缀不存在 | mkdir: cannot create directory '*path*': No such file or directory |

请注意，当打印错误消息时，应该使用给定的路径替换path。例如，如果给定的路径是 **cs/cs01**，并且一个名为**cs/cs01**的文件已经存在，那么错误消息应该是:

mkdir: cannot create directory 'cs/cs01': File exists

所有错误消息(包括其余函数中的错误消息)都应该打印到标准输出，这意味着应该使用printf打印它们。还要注意，当出现这些错误之一时，程序不应该退出—函数应该简单地返回文件系统，保持不变。

**例：**

**程序：**

int main(void) {

Fs fs = FsNew();

FsMkdir(fs, "/tmp");

FsMkdir(fs, "tmp");

FsMkdir(fs, "./tmp");

}

**期望结果：**

mkdir: cannot create directory 'tmp': File exists

mkdir: cannot create directory './tmp': File exists

**FsMkfile**

FsMkfile函数的签名如下:

void FsMkfile(Fs fs, char \*path);

该函数接受一个路径，并在给定文件系统中的该路径上创建一个新的空常规文件。这个函数在Linux中没有直接等效的命令，但最接近的命令是**touch**，它可以用来创建空的常规文件，但也有其他用途，如更新时间戳。

**Error**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| 类型 | Error Message |
| 文件已存在于指定路径 | mkfile: cannot create file '*path*': File exists |
| 路径的前缀是一个常规文件 | mkfile: cannot create file '*path*': Not a directory |
| 路径的正确前缀不存在 | mkfile: cannot create file '*path*': No such file or directory |

例：

程序：

int main(void) {

Fs fs = FsNew();

FsMkfile(fs, "hello");

FsTree(fs, NULL);

FsMkfile(fs, "hello/world");

FsMkdir(fs, "html");

FsMkfile(fs, "html/index.html");

FsMkfile(fs, "html/index.html/hi");

FsTree(fs, NULL);

}

期望结果：

/

hello

mkfile: cannot create file 'hello/world': Not a directory

mkfile: cannot create file 'html/index.html/hi': Not a directory

/

hello

html

index.html

**FsCd**

FsCd函数的签名如下:

void FsCd(Fs fs, char \*path);

该函数的路径可能为**NULL**。

如果路径不为**NULL**，函数应该将当前工作目录更改为该路径。

如果该路径为**NULL**，则默认为**root directory**  (而不是主目录（home directory），因为在这次任务中我们没有主目录)。

该函数大致相当于Linux中的**cd**命令。

Errors

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| 类型 | Error Message |
| 路径的前缀是一个常规文件 | cd: '*path*': Not a directory |
| 路径的前缀不存在 | cd: '*path*': No such file or directory |

例：

程序：

int main(void) {

Fs fs = FsNew();

FsMkdir(fs, "tmp");

FsCd(fs, "tmp");

FsCd(fs, NULL);

FsMkfile(fs, "hello.txt");

FsTree(fs, NULL);

}

期望输出：

/

hello.txt

tmp

**FsLs**

FsLs函数的签名如下:

void FsLs(Fs fs, char \*path);

该函数的路径可能为NULL。

如果路径不是NULL并且指向一个目录，那么函数应该打印该目录中所有文件的名称(除了**. and . .** )，按照ASCII顺序，每行一个。如果路径指向一个文件，那么该函数应该只打印文件系统中存在的给定路径。

如果路径为NULL，则默认为当前工作目录。

这个函数大致相当于Linux中的ls命令。

**Errors**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| 类型 | Error Message |
| 路径的正确前缀(proper prefix)是一个常规文件 | ls: cannot access '*path*': Not a directory |
| 路径的前缀不存在 | ls: cannot access '*path*': No such file or directory |

例

程序

int main() {

Fs fs = FsNew();

printf("---**\n**"); *// marker to separate output*

FsLs(fs, "/");

printf("---**\n**");

FsMkfile(fs, "hello.txt");

FsMkdir(fs, "tmp");

FsLs(fs, "/");

}

期待输出

---

---

hello.txt

tmp

**FsPwd**

FsPwd函数的签名如下:

void FsPwd(Fs fs);

该函数打印当前工作目录的规范路径。

该函数大致相当于Linux下的**pwd**命令。

例：

程序：

int main() {

Fs fs = FsNew();

FsPwd(fs);

FsMkdir(fs, "home");

FsCd(fs, "home");

FsPwd(fs);

FsMkdir(fs, "tim");

FsCd(fs, "tim");

FsPwd(fs);

}

期望输出：

/

/home

/home/tim

**FsTree**

FsTree函数的签名如下:

void FsTree(Fs fs, char \*path);

该函数的路径可能为**NULL**。

如果路径为NULL，则默认为根目录。

该函数以结构化的方式打印给定路径的目录层次结构(见下面)。

这个函数大致相当于Linux中的**tree**命令。

**输出格式：**

输出的第一行应该包含给定的路径，如果它存在并指向一个目录。下面的行应该按照ASCII顺序显示给定目录下的所有文件，每行一个，用缩进显示哪些文件包含在哪些目录下。每一级缩进增加4个空格。请参阅用法示例。

**Error:**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| 类型 | Error Message |
| 路径的前缀是一个常规文件 | tree: '*path*': Not a directory |
| 路径的前缀不存在 | tree: '*path*': No such file or directory |

例：

程序：

int main(void) {

Fs fs = FsNew();

FsMkfile(fs, "hello");

FsTree(fs, "hello");

FsTree(fs, "./hello/world");

}

期望输出：

tree: 'hello': Not a directory

tree: './hello/world': Not a directory

**任务2**

**FsPut**

FsPut函数的签名如下:

void FsPut(Fs fs, char \*path, char \*content);

该函数接受一个路径和一个字符串，并将该路径上的常规文件的内容设置为给定的字符串。如果文件已经有一些内容，那么它将被覆盖。

**Error**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| **类型** | **Error Message** |
| 该路径指的是一个目录 | put: '*path*': Is a directory |
| 路径的正确前缀是一个常规文件 | put: '*path*': Not a directory |
| 路径的前缀不存在 | put: '*path*': No such file or directory |

**例1**

**程序**

int main() {

Fs fs = FsNew();

FsMkfile(fs, "hello.txt");

FsPut(fs, "hello.txt", "hello**\n**");

FsPut(fs, "./hello.txt", "world**\n**"); *// overwrites existing content*

}

**此程序无期望输出**

**例2**

**程序**

int main(void) {

Fs fs = FsNew();

FsMkfile(fs, "hello");

FsPut(fs, "hello/world", "random-message**\n**");

}

**期望输出：**

put: 'hello/world': Not a directory

**FsCat**

FsCat函数的签名如下:

void FsCat(Fs fs, char \*path);

该函数接受一个路径，并在该路径上打印常规文件的内容。这个函数大致相当于Linux中的**cat**命令

**Errors**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| **类型** | **Error Message** |
| 该路径指的是一个目录 | cat: '*path*': Is a directory |
| 路径的正确前缀是一个常规文件 | cat: '*path*': Not a directory |
| 路径的前缀不存在 | cat: '*path*': No such file or directory |

**例**

**程序**

int main(void) {

Fs fs = FsNew();

FsMkdir(fs, "hello");

FsCat(fs, "hello");

FsCat(fs, ".");

FsCat(fs, "/");

}

**期望输出**

cat: 'hello': Is a directory

cat: '.': Is a directory

cat: '/': Is a directory

**FsDldir**

FsDldir函数的签名如下:

void FsDldir(Fs fs, char \*path);

该函数接受一个指向目录的路径，当且仅当该路径为空时删除该目录。这个函数大致相当于Linux中的**rmdir**命令。

为简单起见，可以假设给定路径不包含当前工作目录。注意，这意味着给定的路径永远不会是根目录。**如果您愿意(为了完整性起见)，您可以处理这种情况，但是不会对它进行测试。**

**Errors**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| **类型** | **Error Message** |
| 该路径指的是一个非空目录 | dldir: failed to remove '*path*': Directory not empty |
| 路径的前缀是一个常规文件 | dldir: failed to remove '*path*': Not a directory |
| 路径的前缀不存在 | dldir: failed to remove '*path*': No such file or directory |

**例**

**程序**

int main(void) {

Fs fs = FsNew();

FsMkdir(fs, "hello");

FsMkdir(fs, "hello/world");

FsDldir(fs, "hello");

FsTree(fs, NULL);

}

**期望输出**

dldir: failed to remove 'hello': Directory not empty

/

hello

world

**FsDl**

FsDl函数的签名如下:

void FsDl(Fs fs, bool recursive, char \*path);

该功能采取路径并删除该路径上的文件。默认情况下，该功能拒绝删除目录：它只会删除目录（及其所有内容递归），如果递归是真实的。如果路径指常规文件，则递归参数无关紧要。此函数大致对应于 Linux 中的 **rm** 命令，递归真实性与 **rm** 命令中使用的 **-r** 选项相对应。

为简单起见，可以假设给定路径不包含当前工作目录。注意，这意味着给定的路径永远不会是根目录。**如果您愿意(为了完整性起见)，您可以处理这种情况，但是不会对它进行测试。**

**Errors**

您必须处理以下错误并生成如下所示的错误消息。如果应用了多个错误，则只从应用的表中打印第一个错误的错误消息。

|  |  |
| --- | --- |
| 类型 | Error Message |
| 路径指向一个目录，但递归为false | dl: cannot remove '*path*': Is a directory |
| 路径的正确前缀是一个常规文件 | dl: cannot remove '*path*': Not a directory |
| 路径的前缀不存在 | dl: cannot remove '*path*': No such file or directory |

例

程序

int main(void) {

Fs fs = FsNew();

FsMkdir(fs, "hello");

FsDl(fs, false, "hello");

}

期望输出

dl: cannot remove 'hello': Is a directory

**任务3**

**FsCp**

FsCp函数的签名如下:

void FsCp(Fs fs, bool recursive, char \*src[], char \*dest);

该函数接受一个以**NULL**结尾的路径数组**src**和路径**dest**。如果**src**数组恰好包含一个路径，那么它应该将位于**src**的文件复制到**dest**。如果**src**数组包含多个路径，那么**dest**应该指向一个目录，函数应该将**src**数组中所有路径下的文件复制到**dest**目录下。默认情况下，函数不复制目录-只有当递归为**true**时，它才应该复制目录。

这个函数大致相当于Linux中的**cp**命令。

**Errors**

如果您在Linux中试验**cp**命令，您会发现可能出现许多不同的错误(超过10个)。因为我们不想把赋值的重点放在处理错误上，所以可以假设给**FsCp**的参数不会导致错误。然而，我们提供了一个可能的错误消息列表，以满足您的好奇心:

cp可能出现的错误：

|  |
| --- |
| cp: missing destination file operand after '*path*' |
| cp: -r not specified; omitting directory '*path*' |
| cp: cannot stat '*path*': Not a directory |
| cp: failed to access '*path*': Not a directory |
| cp: cannot stat '*path*': No such file or directory |
| cp: cannot create regular file '*path*': No such file or directory |
| cp: cannot create directory '*path*': No such file or directory |
| cp: '*path1*' and '*path2*' are the same file |
| cp: cannot overwrite non-directory '*path1*' with directory '*path2*' |
| cp: cannot copy a directory, '*path1*', into itself, '*path2*' |
| cp: target '*path*' is not a directory |

例

我们已经提供了**src**数组只包含一条路径的例子。

程序

int main() {

Fs fs = FsNew();

FsMkfile(fs, "hello.txt");

FsPut(fs, "hello.txt", "hello**\n**");

FsMkfile(fs, "world.txt");

FsPut(fs, "world.txt", "world**\n**");

FsCat(fs, "world.txt");

printf("---**\n**");

char \*src[] = { "hello.txt", NULL };

FsCp(fs, false, src, "world.txt");

FsCat(fs, "world.txt");

printf("---**\n**");

FsTree(fs, NULL);

}

期望输出

world

---

hello

---

/

hello.txt

world.txt

描述:如果**src**和**dest**路径都指向普通文件，那么**src**文件的内容应该简单地复制到**dest**文件，覆盖它的内容。

**任务4**

**FsMv**

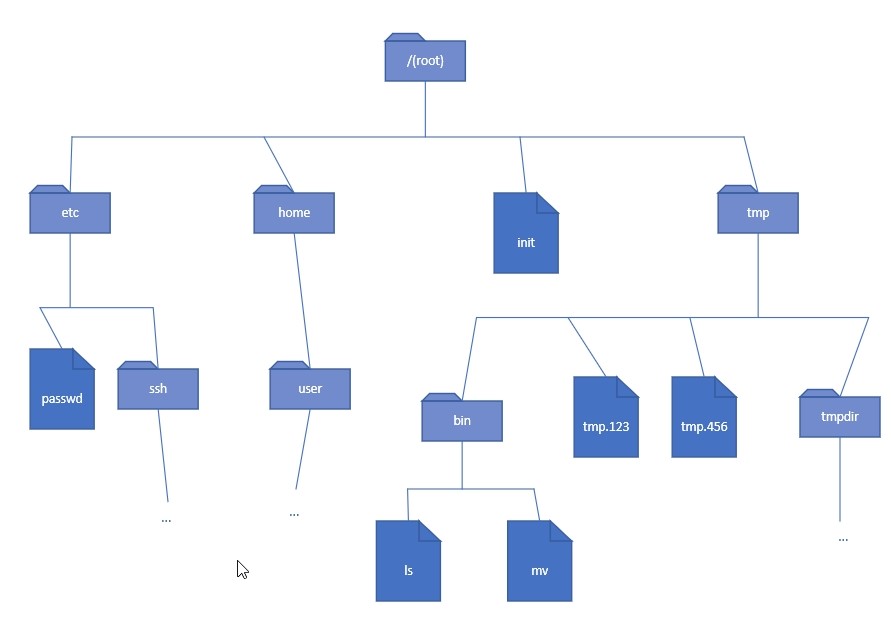
FsMv功能具有如下特征:

void FsMv(Fs fs, char \*src[], char \*dest);

该函数接受以**null**结尾的**src**路径数组和**dest**路径。它应该将**src**中所有路径所指向的文件移动到**dest**。

该函数大致相当于Linux中的**mv**命令。

例如，考虑背景部分中显示的文件系统。如果bin目录被移动到**tmp**目录，那么文件系统现在看起来像:



这个移动可以通过testFs.c中的以下代码实现:

char \*src[] = { "/bin", NULL };

FsMv(fs, src, "/tmp");

**Errors**

与任务3类似，为了将重点放在实现上，可以假设给**FsMv**的参数不会导致错误。

与任务3类似，我们已经提供了**src**数组只包含一条路径的示例。当**src**数组包含多个路径时，由您来决定正确的行为。

例

程序

int main() {

Fs fs = FsNew();

FsMkfile(fs, "hello.txt");

FsPut(fs, "hello.txt", "hello**\n**");

FsTree(fs, NULL);

printf("---**\n**");

char \*src[] = { "hello.txt", NULL };

FsMv(fs, src, "world.txt");

FsTree(fs, NULL);

printf("---**\n**");

FsCat(fs, "world.txt");

}

期待输出：

/

hello.txt

---

/

world.txt

---

hello

---

描述:如果**dest**路径不存在，但它的所有正确前缀都存在，那么**src**文件应该简单地移动到**dest**路径。