## Question

随着操作系统相关理论的研究和开源社区的贡献，独立开发一款相应量级的操作系统已不再困难。困难在于大量历史应用的移植或适配——这将很大程度上决定一款操作系统的市场前景。目前各家操作系统之间为了扩大或稳定自己的用户群体，都采取了集成对方操作系统的“应用子系统”、模拟器或兼容层（如Wine、Android for win）的方式来尝试适配更多应用。

1. 尝试结合开源代码来分析这些子系统的关键工作原理(3~5个关键点)

2. 尝试结合开源代码分析总结 Linux 和 Windows (ReactOS项目)两种系统在\*文件操作\*上的异同。

3. 基于上述 2 点分析，请在 Linux 系统上使用 C 语言，尝试编写一个简单的\*文件操作\*兼容层(模拟层)\*将 Windows 的文件操作转换或翻译为 Linux 的文件操作\*，并使用文档说明实现的机制和你的设计特点。

4. 基于业界现状和你的上述研究与模拟操作实现，你认为上述技术路线存在哪些问题或缺陷？请选择2~3个方面详细阐述。

<!-- 题目要求: 请将源代码、可执行文件、分析笔记、分析报告和文档打包发至: lzfls050409@qq.com -->

## Solution

\*By Bai Jiale - id: 2023090913021\*

### 一、关键工作原理

#### 以 [Wine](https://github.com/wine-mirror/wine) 为例：

1. Wine 包含了大量的库，这些库是 Windows 操作系统中API函数的重新实现。这些库使得 Windows 应用程序可以在 Unix 环境中调用这些 API，从而执行各种操作。

2. Wine 对 Windows 的内存管理和文件系统进行了模拟。例如，它会创建一个虚拟的C盘，并提供与 Windows 类似的文件路径结构。

3. Wine 能够动态链接和加载 Windows 的 [Dynamic-link library](https://zh.wikipedia.org/wiki/动态链接库)（动态链接库）。这些 DLL 文件包含了应用程序所需的功能，而Wine通过其实现的Windows API来满足这些功能的需求。

4. Wine 能够解析和加载 Windows 应用程序分发的的 [Portable Executable](https://zh.wikipedia.org/wiki/可移植可执行) 文件，提供一个兼容的运行时环境，让这些应用程序可以在 Unix 系统上运行。

5. Wine 还能够仿真实现 Windows 的注册表以储存配置信息和其他数据。

##### 再看 [windows-subsystem-for-android](https://github.com/microsoft/WSA)：

1. WSA 的原理是在 Windows 上创建一个轻量级的虚拟机，运行一个经过优化的 Android 内核。

2. 在这个虚拟机中运行的是一个完整的 Android 系统，包括 Android 的核心组件和系统服务。

3. WSA 包含一个兼容层，将 Android 应用程序的 API 调用翻译成 Windows 系统可以理解的调用。这使得 Android 应用程序可以与 Windows 进行交互。

相比 Wine，WSA更加集成化，表层化。

### 二、Linux 和 Windows 在文件操作上的异同。

https://github.com/torvalds/linux

https://github.com/reactos/reactos

#### 相同之处：

1. 两个操作系统都支持基本的文件操作，如打开、读取、写入、关闭、删除、重命名等。

#### 不同之处：

1. Linux 使用 POSIX 标准的系统调用，例如 open(), read(), write(), close() 等；而 Windows 使用 Win32 API，例如 CreateFile(), ReadFile(), WriteFile(), CloseHandle() 等。

2. Linux 使用正斜杠 / 作为路径分隔符，路径是区分大小写的；而 Windows 使用反斜杠 \ 作为路径分隔符，路径不区分大小写（但文件名区分大小写）。

3. Linux 使用权限位和用户/组/其他（user/group/others）模式（例如，rwx 权限）来控制文件访问；而 Windows 使用访问控制列表（ACL）来管理文件权限。

### 三、在 Linux 上编写文件操作的模拟层

\*\*Attention: 编译环境为 MacOS\*\*

#### 重现方式

1. 在当前目录 make

```sh

$ make

```

2. Usage：

```sh

$ main <(char\*)data>

```

#### 学习笔记

##### 1. 打开/创建文件

\*\*open() function in Linux\*\*

open 函数是 Unix 和 Linux 系统中的系统调用，用于打开或创建文件。

原型：

```c

int open(const char \*pathname, int flags);

int open(const char \*pathname, int flags, mode\_t mode);

```

参数：

open() 函数有两种形式。第一种形式接受两个参数，第二种形式接受三个参数。

1. `pathname`：指针类型，指向要打开或创建的文件的路径。

2. `flags`：整数类型，指定文件应如何打开。flags 参数可以是以下常量之一，或者是它们的组合：

\* `O\_RDONLY`：以只读方式打开文件。

\* `O\_WRONLY`：以只写方式打开文件。

\* `O\_RDWR`：以读写方式打开文件。

\* `O\_CREAT`：如果文件不存在，则创建新文件。

\* `O\_TRUNC`：如果文件已存在，并且是以写入或读写方式打开的，则其长度将被截断为 0。

\* `O\_APPEND`：在每次写入时，都将文件的偏移量设置为文件的末尾。

3. `mode`：这是一个可选参数，只有当 flags 参数包含 O\_CREAT 时才需要。mode 参数指定了新创建的文件的权限。0666 表示新文件的用户、组和其他用户都有读写权限。

函数返回一个文件描述符，这是一个用于引用打开的文件的小整数。如果出现错误，返回 -1，并设置全局变量 errno 以指示错误的类型。

\*\*CreateFile() function in Windows\*\*

`CreateFile` 函数是 Windows 系统中的API调用，用于创建或打开一个文件、文件夹、命名管道、设备等。

原型：

```c

HANDLE CreateFile(

LPCTSTR lpFileName,

DWORD dwDesiredAccess,

DWORD dwShareMode,

LPSECURITY\_ATTRIBUTES lpSecurityAttributes,

DWORD dwCreationDisposition,

DWORD dwFlagsAndAttributes,

HANDLE hTemplateFile

);

```

参数：

1. `lpFileName`：指针类型，指向要打开或创建的文件的路径。

2. `dwDesiredAccess`：整数类型，指定文件应如何打开。

3. `dwShareMode`：整数类型，指定文件的共享模式。

4. `lpSecurityAttributes`：这是一个指向`SECURITY\_ATTRIBUTES`结构的指针，该结构决定了返回的句柄是否可以被子进程继承，以及对于新创建的文件或目录的安全描述符。

5. `dwCreationDisposition`：整数类型，决定如何创建或打开文件。例如，`CREATE\_NEW`表示创建一个新文件，如果文件已存在则失败，`OPEN\_EXISTING`表示打开一个存在的文件，如果文件不存在则失败。

6. `dwFlagsAndAttributes`：整数类型，指定文件或设备的属性和标志。

7. `hTemplateFile`：这是一个句柄，指向一个模板文件。如果该参数不为NULL，那么该模板文件提供了创建新文件时所需的文件属性和扩展属性。

函数返回一个句柄，这是一个用于引用打开的文件的值。如果出现错误，返回`INVALID\_HANDLE\_VALUE`，并设置全局变量`GetLastError`以指示错误的类型。

##### 2. 读文件

\*\*read() function in Linux\*\*

read() 用于从打开的文件或其他输入源（如套接字）中读取数据。

原型：

```c

ssize\_t read(int fd, void \*buf, size\_t count);

```

参数：

1. `fd`：这是一个文件描述符，它引用了要从中读取数据的文件或其他输入源。

2. `buf`：指针类型，指向一个缓冲区，用于存储从文件中读取的数据。

3. `count`：整数类型，指定要从文件中读取的字节数。

read 函数\*返回实际读取的字节数\*。如果到达文件末尾，或者没有更多的数据可读，返回0。如果出现错误，返回 -1，并设置全局变量 errno 以指示错误的类型。

\*\*ReadFile() function in Windows\*\*

ReadFile 函数用于从打开的文件或其他输入源中读取数据。

原型：

```c

BOOL ReadFile(

HANDLE hFile,

LPVOID lpBuffer,

DWORD nNumberOfBytesToRead,

LPDWORD lpNumberOfBytesRead,

LPOVERLAPPED lpOverlapped

);

```

参数：

1. `hFile`：句柄类型，它引用了要从中读取数据的文件或其他输入源。

2. `lpBuffer`：指针类型，指向一个缓冲区，用于存储从文件中读取的数据。

3. `nNumberOfBytesToRead`：整数类型，指定要从文件中读取的字节数。

4. `lpNumberOfBytesRead`：指针类型，指向一个变量，该变量在读取操作完成后被赋值为实际读取的字节数。

5. `lpOverlapped`：这是一个指向`OVERLAPPED`结构的指针，该结构包含了用于处理异步I/O的信息。如果文件打开时指定了`FILE\_FLAG\_OVERLAPPED`，那么读取操作将是异步的，否则读取操作将是同步的。

函数返回一个布尔值，成功返回值非 0，失败返回值 0。

##### 3. 写文件

\*\*write() function in Linux\*\*

write 函数用于向打开的文件或其他输出源写入数据。

原型：

```c

ssize\_t write(int fd, const void \*buf, size\_t count);

```

参数：

1. `fd`：一个文件描述符，它引用了要向其中写入数据的文件或其他输出源。

2. `buf`：指针类型，指向某个缓冲区，该缓冲区包含要写入文件的数据。

3. `count`：整数类型，指定要写入文件的字节数。

函数返回实际写入的字节数。如果出现错误，返回-1，并设置全局变量`errno`以指示错误的类型。

\*\*WriteFile() function in Windows\*\*

WriteFile 函数用于向打开的文件或其他输出源写入数据。

原型：

```c

BOOL WriteFile(

HANDLE hFile,

LPCVOID lpBuffer,

DWORD nNumberOfBytesToWrite,

LPDWORD lpNumberOfBytesWritten,

LPOVERLAPPED lpOverlapped

);

```

参数：

1. `hFile`：句柄类型，它引用了要向其中写入数据的文件或其他输出源。

2. `lpBuffer`：指针类型，指向包含要写入文件数据的缓冲区。

3. `nNumberOfBytesToWrite`：整数类型，指定要写入文件的字节数。

4. `lpNumberOfBytesWritten`：指针类型，指向一个变量，该变量在写入操作完成后包含了实际写入的字节数。

5. `lpOverlapped`：这是一个指向`OVERLAPPED`结构的指针，该结构包含了用于处理异步I/O的信息。如果文件打开时指定了`FILE\_FLAG\_OVERLAPPED`，那么写入操作将是异步的，否则写入操作将是同步的。

函数返回一个布尔值，成功返回非 0，失败返回 0。

### 四、存在的问题或缺陷

1. 命令行工具较为复杂（如：cp、mv、find等），对于初学者不是那么友好。对于大规模文件操作和管理，命令行工具的效率和易用性可能不如图形界面工具。

2. Linux 和 Windows 之间的相互兼容性还需要提高，跨平台开发目前还是存在过于麻烦的问题