**操作系统原理实验报告**

**实验一 小型shell模拟**

**3022206045-陆子毅**

**实验内容：**

实现运行原生的linux程序，实现输入输出重定向，管道符

**实验过程：**

**一、执行简单命令**

根据手册指引，找到了runcmd所在位置，并用man 3 exec 查看了exec函数原型。在runcmd函数中，编写了代码来处理简单命令。使用execv函数来执行用户输入的命令，并在执行失败时打印错误消息。

查阅资料得知，

execv 和 execvp 都是用于在一个进程中执行另一个程序的函数，它们属于类Unix操作系统的系统调用。它们之间的主要区别在于参数的传递方式和搜索可执行文件的方式。

**execv 函数：**

原型：int execv(const char \*path, char \*const argv[]);

接受两个参数，第一个参数是要执行的程序的路径名，第二个参数是一个字符串数组，其中包含了传递给新程序的命令行参数。

需要明确指定可执行文件的路径，即必须提供完整的路径，否则需要确保当前工作目录包含可执行文件。不会搜索 PATH 环境变量中指定的目录，因此需要提供完整路径或者在当前目录中执行可执行文件。

**execvp 函数：**

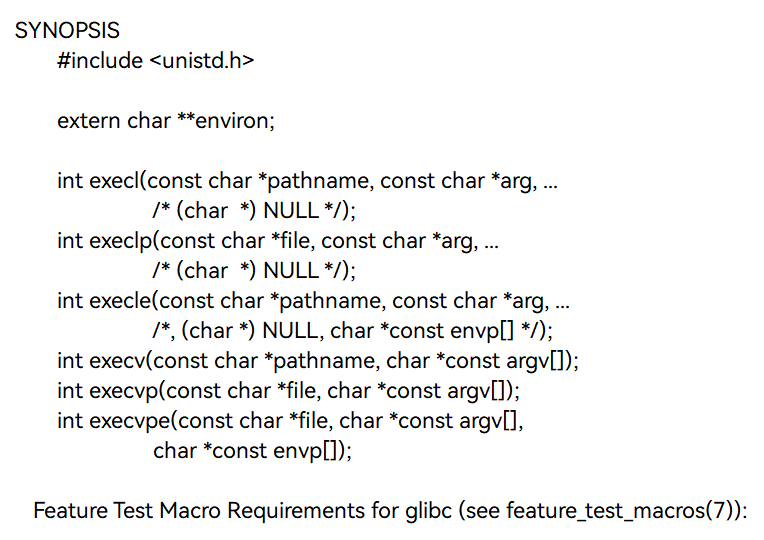
原型：int execvp(const char \*file, char \*const argv[]);

接受两个参数，第一个参数是要执行的程序的名称，第二个参数是一个字符串数组，其中包含了传递给新程序的命令行参数。

会搜索 PATH 环境变量中指定的目录，以查找可执行文件。因此，你可以只提供可执行文件的名称而无需提供完整路径。

如果可执行文件的路径可以在 PATH 中找到，execvp 会自动查找并执行它。

总之，主要区别在于 execv 需要提供完整路径，而 execvp 可以根据 PATH 环境变量中的目录来查找可执行文件。通常情况下，execvp 更为常用，因为它更方便，不需要显式指定完整路径，但如果你需要精确控制可执行文件的路径，可以使用 execv。



图片包含 室内, 桌子, 房间

描述已自动生成

文本

描述已自动生成

测试成功，并且在运行命令时不用写/bin

**二、输入输出重定向**

解析器已经能够识别>和<符号，并构建了redircmd。只需要填写runcmd函数中>和<的代码部分，根据实验提示，应该使用open和close等系统调用。我们确保在系统调用失败时打印错误消息。

使用命令查看open，close的函数原型使用说明。文本, 信件

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

open 和 close 是Unix-like操作系统中的两个系统调用，它们用于文件操作。它们的作用如下：

open 函数：

open 用于打开文件或创建新文件，并返回一个文件描述符，该文件描述符是一个非负整数，代表打开的文件。

open 函数的原型为：int open(const char \*path, int flags, mode\_t mode);

参数说明：

path：指定要打开的文件的路径或名称。

flags：指定文件打开的方式和选项，例如只读、只写、追加等。常见的标志包括 O\_RDONLY（只读）、O\_WRONLY（只写）、O\_RDWR（读写）、O\_CREAT（创建文件）、O\_APPEND（在文件末尾追加）等。

mode：通常用于指定新文件的访问权限，例如 0666 表示文件的读写权限。

open 函数返回一个文件描述符，可以在后续的文件操作中使用这个描述符来标识和访问文件。

close 函数：

close 用于关闭一个已打开的文件，释放相关资源，并使文件描述符不再有效。

close 函数的原型为：int close(int fd);

参数 fd 是要关闭的文件描述符。

当你完成了对文件的操作后，应该调用 close 函数来释放文件描述符，以避免资源泄漏和确保文件被正确关闭。

dup2 是一个Unix-like操作系统中的系统调用，用于复制文件描述符（file descriptor）。它的作用是创建一个新的文件描述符，使其与已有的文件描述符指向相同的文件或流。dup2 允许你将一个文件描述符的副本与另一个文件描述符相关联，以便在后续的文件操作中同时使用这两个描述符。

dup2 函数的原型如下：

int dup2(int oldfd, int newfd);

oldfd 是要复制的旧文件描述符，它必须是有效的文件描述符。

newfd 是要创建的新文件描述符，它通常是一个未使用的文件描述符。如果 newfd 已经被占用，dup2 会首先关闭它，然后将其重定向到 oldfd 指向的文件或流。

在 dup2(file\_fd, rcmd->fd) 中，dup2 函数用于复制文件描述符 file\_fd 到 rcmd->fd。这通常是用于文件描述符的重定向操作，将一个文件描述符与另一个文件描述符关联，以实现输入/输出重定向或者管道操作。

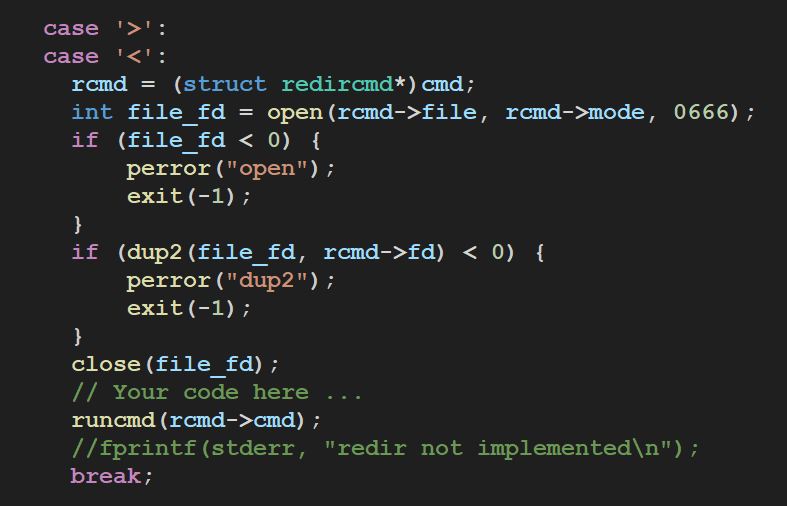
具体来说，这行代码的作用是将 file\_fd 指向的文件描述符的内容复制到 rcmd->fd 指向的文件描述符中。这可能会涉及以下操作：

如果 rcmd->fd 已经打开并与某个文件或流相关联，那么 dup2 会首先关闭 rcmd->fd，然后将其重定向到 file\_fd 指向的文件或流，使得 rcmd->fd 成为 file\_fd 的一个副本。

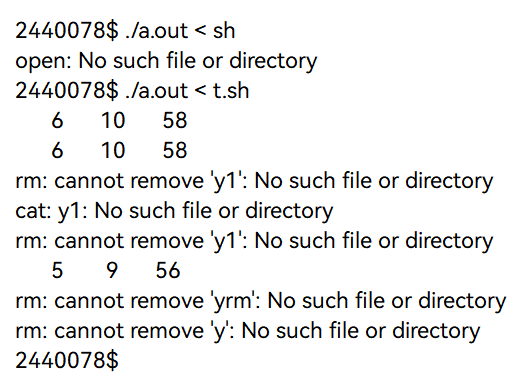
如果 rcmd->fd 未打开，dup2 会简单地将其指向 file\_fd 指向的文件或流，使得 rcmd->fd 现在与 file\_fd 具有相同的文件内容。

这种操作通常用于重定向进程的标准输入、标准输出或标准错误，或者在创建管道时将一个进程的输出与另一个进程的输入相连。

需要注意的是，在执行 dup2 后，rcmd->fd 将继续指向与 file\_fd 相同的文件或流，因此所有的读写操作将影响到这个文件或流。



测试运行，成功



**三、管道操作**

管道操作需要用到使用了pipe、fork、close和dup等系统调用。使用man命令查询具体函数。

pipe、fork、close 和 dup 是在Unix-like操作系统中常用于创建管道和进程间通信的系统调用，它们通常一起使用。

**pipe 函数：**

pipe 用于创建一个管道，它可以在两个进程之间传递数据。管道有两个文件描述符，一个用于读取数据，另一个用于写入数据。

pipe 函数的原型为：int pipe(int pipefd[2]);

pipefd 是一个长度为2的整数数组，用于存储管道的两个文件描述符，pipefd[0] 用于读取数据，pipefd[1] 用于写入数据。

**fork 函数：**

fork 用于创建一个新的进程，新进程是调用进程的副本。这两个进程几乎完全相同，但有不同的进程ID（PID）。

子进程通常用于执行不同的任务，例如在管道通信中，一个进程用于写入数据，另一个进程用于读取数据。

父进程和子进程都会继续执行后续代码，但可以通过返回值来区分哪个进程是父进程，哪个是子进程。

**close 函数：**

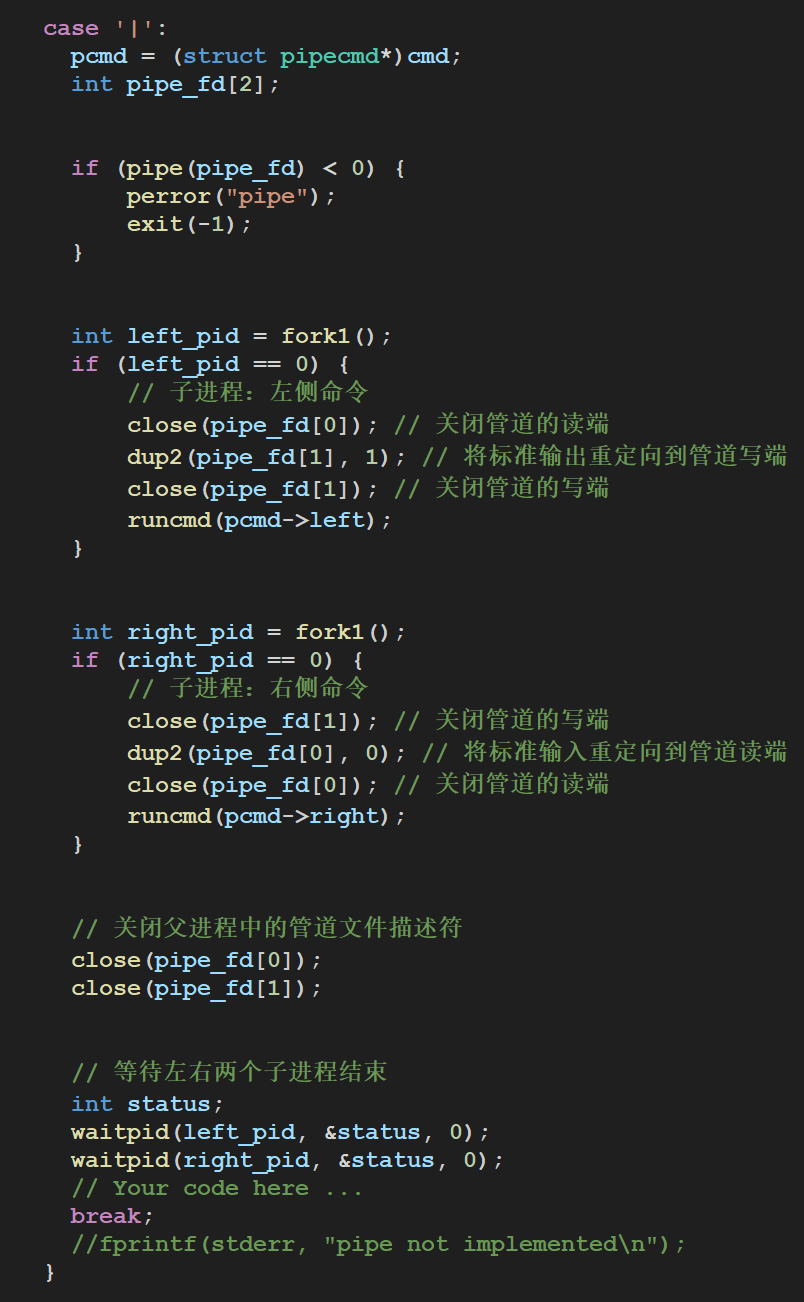
close 用于关闭一个文件描述符，释放相关资源。在管道通信中，通常在父子进程中使用 close 来关闭不需要的文件描述符，以防止资源泄漏。

**dup 函数：**

dup 用于复制文件描述符，创建一个副本，该副本与原始文件描述符指向相同的文件或流。这可以用于在不同的文件描述符上进行读写操作，以实现进程间通信。

例如，在子进程中，可以使用 dup 复制标准输出文件描述符，然后使用 close 关闭原始标准输出文件描述符，以将子进程的标准输出重定向到管道。

综合使用这些系统调用，可以在父子进程之间创建管道，将数据从一个进程传递到另一个进程，通过 fork 创建子进程，使用 close 关闭不需要的文件描述符，以及使用 dup 进行输入/输出重定向。这是实现进程间通信的常见方法之一。



**实验总结：**

实验不是很难，但是需要我们去认证阅读linux编程手册，在这期间，准确快速地阅读英文手册是一个难点，希望能够通过不断的锻炼渐渐摆脱翻译软件。

通过本次实验，我们成功实现了一个基本的Unix Shell，具备了运行原生Linux程序、输入输出重定向和管道操作的功能。我们熟悉了系统调用接口和Shell的工作原理，提高了操作系统原理的实际编程能力。本次实验使我们更深入地理解了操作系统原理，特别是Shell的实现。我们成功地实现了所需的功能，同时也学到了如何查看系统调用文档和调试Shell程序。在接下来的实验中，我们将继续学习和扩展Shell的功能，以更好地理解操作系统的工作原理。