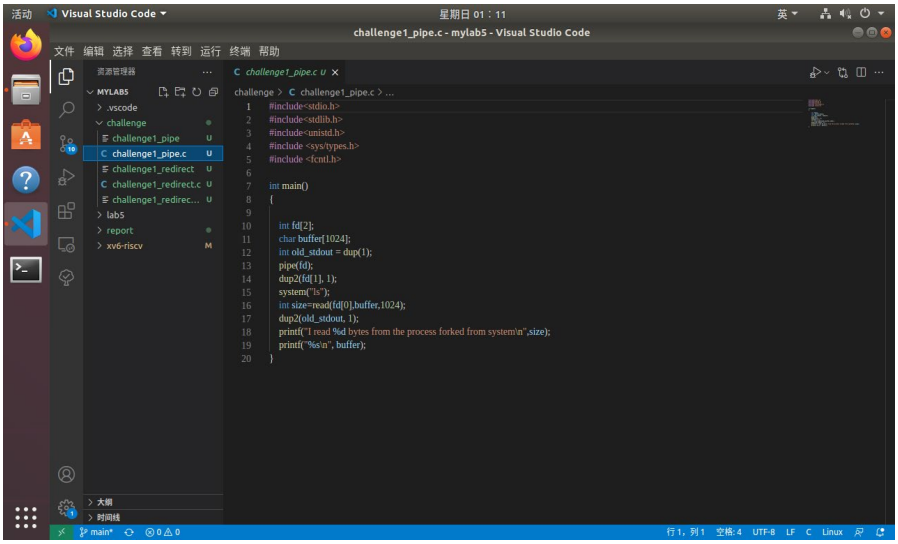
201220180 李全昊 OSlab5 实验报告

1. exercise1：想想为什么我们不使用文件名而使用文件描述符作为文件标识。
   int read(const char *filename, void *buffer, int size);
   int write(const char *filename, void *buffer, int size);
   答：首先，文件名是一个字符串，操作速度慢且占空间大，而文件描述符为一整数，其处理效率明显高于字符串。其次，文件被打开后其控制信息(FCB)被缓冲到内存系统空间，文件描述符作为用户打开文件表中的入口地址直接与内存 FCB 建立起联系，而文件名无法做到这一点。
2. exercise2：为什么内核在处理 exec 的时候，不需要对进程描述符表和系统文件打开表进行任何修改。（可以先往下看看再回答，或者**阅读一下 Xv6 的 shell**）
   答：因为 exec 只是把别的地址空间的代码替换了当前进程的而已，并不对当前进程打开的文件进行任何操作，系统也没有新的打开（或者关闭的）文件，因此这两个表都不会被修改。
3. challenge1：system 函数（自行搜索）通过创建一个子进程来执行命令。但一般情况下，system 的结果都是输出到屏幕上，有时候我们需要在程序中对这些输出结果进行处理。一种解决方法是定义一个字符数组，并让 system 输出到字符数组中。如何使用重定向和管道来实现这样的效果？
   Hint：可以用 pipe 函数（自行搜索）、read 函数（你们都会）.....
   （在 Linux 系统下自由实现，不要受约束）
   1. 重定向：

```c
int main()
{
    write(1,"111\n",4);
    close(1);
    open("challenge1.txt",O_RDWR+O_CREAT);
    system("ls");
    return 0;
}
```

   代码见上图，通过这样的操作，我们编译这个.c 文件运行之后可以发现 111 可以被输出到终端，但是 system 执行的 ls 指令输出的信息就被重定向到了 a.txt 文件里面了，剩下的我们只需要简单的文件操作就可以获取得到 system 输出的信息，把他们存到自己想存入的地方（比如手册里写的字符数组）
   2. 管道



代码见上图，我们首先将 stdout 通过 dup 函数备份到 old_stdout 这个文件描述符中。然后我们调用 pipe 函数创建管道，通过 dup2 函数把 fd[1]对应的描述符复制给 stdout，然后 system 函数创建的子进程就会朝 fd[1]写入信息了。我们在主进程里面就可以调用 read 函数从 fd[0]里面接收到 system 的输出。最后我们用 dup2 函数把我们一开始的备份 old_stdout 复制给 stdout，这样就恢复正常，调用 printf 就朝着标准输出上面输出信息。

4. exercise3：我们可以通过 which 指令来找到一个程序在哪里，比如 which ls，就输出 ls 程序的绝对路径（看下面，绝对路径是/usr/bin/ls）。那我在/home/yxz 这个目录执行 ls 的时候，为什么输出/home/yxz/路径下的文件列表，而不是/usr/bin/路径下的文件列表呢？（请根据上面的介绍解释。）

   答：因为每个进程都应该拥有它的工作目录,我在/home/yxz 这个目录执行 ls 程序的时候,当前进程的工作目录调整为了/home/yxz 了,那么这个时候 ls 列出的就是/home/yxz/路径下的文件列表，而不是/usr/bin/路径下的文件列表。