**task1：**



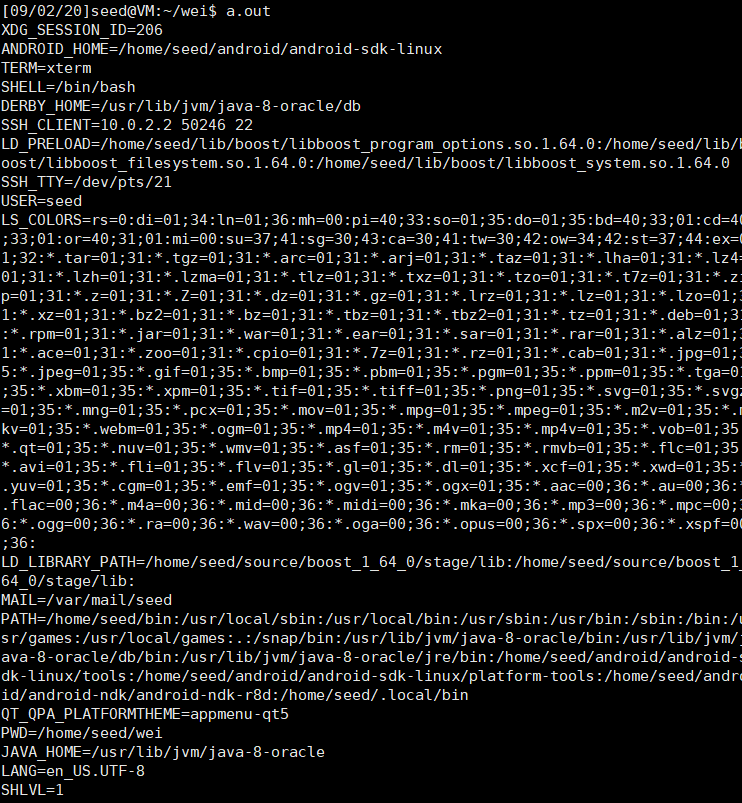






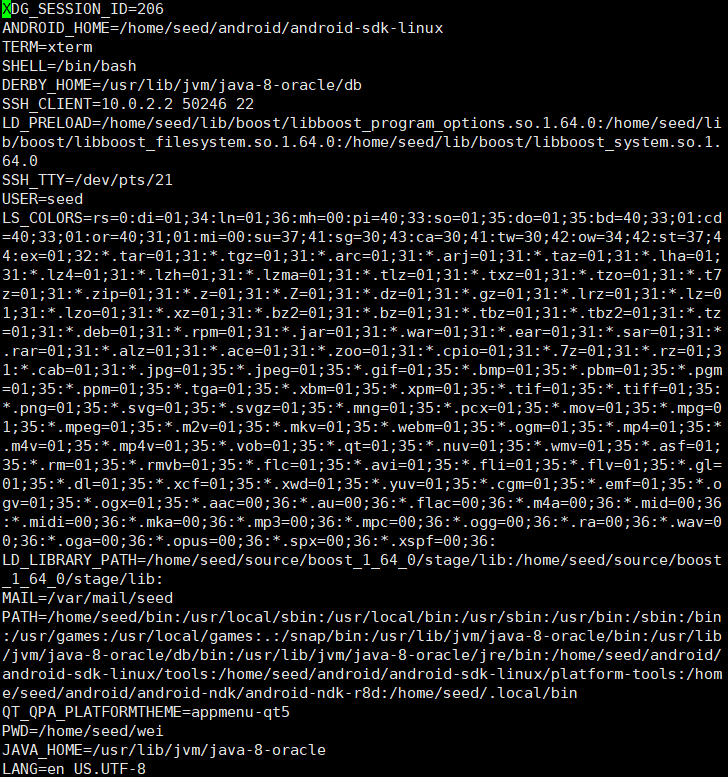
**task2：**

**step1：**程序运行结果如下



**step2：**

程序运行结果如下 输出保存在father文件中

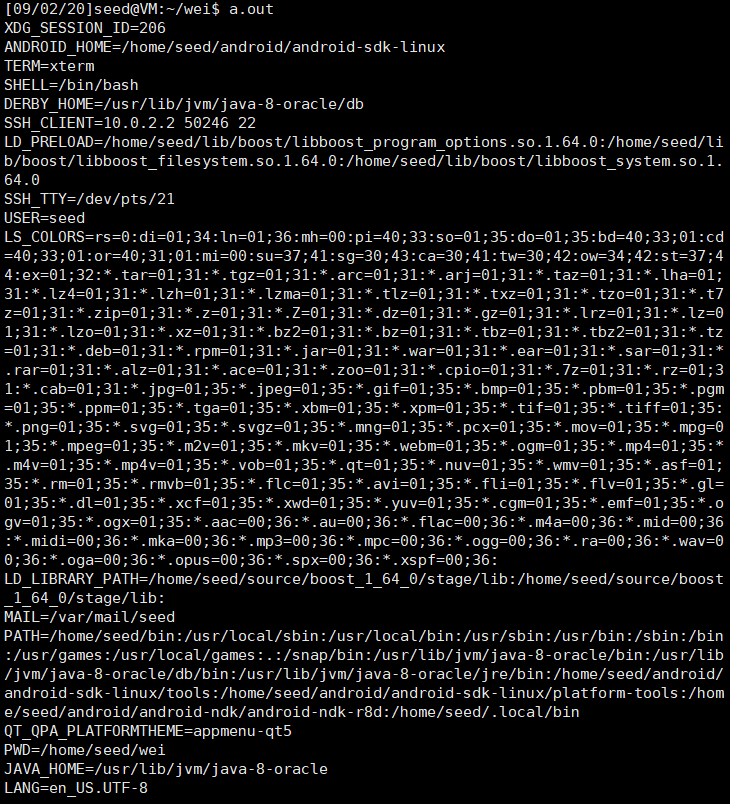


**step3：**通过比较两次的运行结果发现这两个文件输出的环境变量完全相同。说明原环境变量被子进程完全继承。

**task3：**

**step1：**程序无输出

**step2**：程序运行结果如下

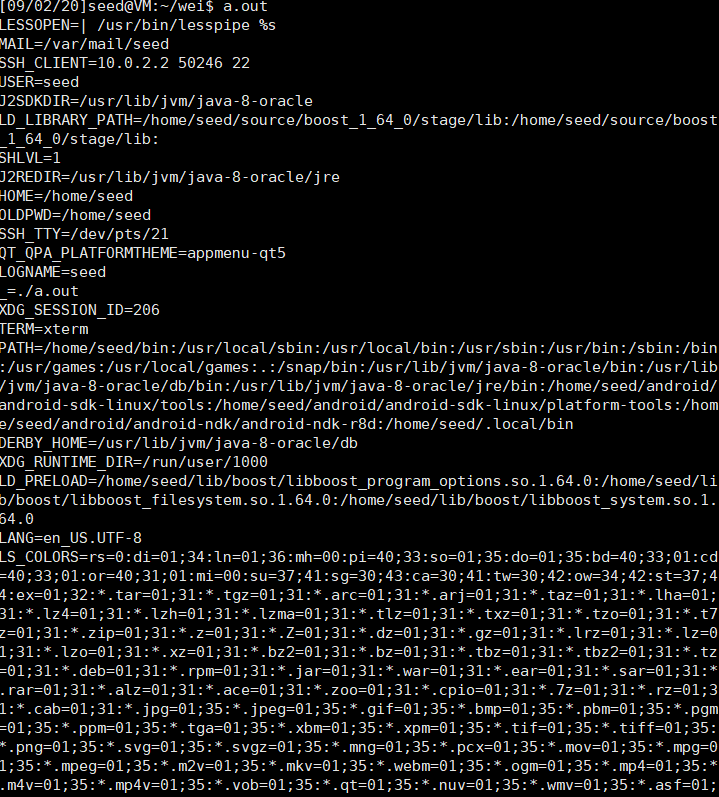


**step3：**

结论：execve函数创建的进程需要通过execve函数的第3个参数来获得环境变量，新进程不会自动继承父进程的环境变量。

**task4：**

程序运行结果如下：



**task5：**

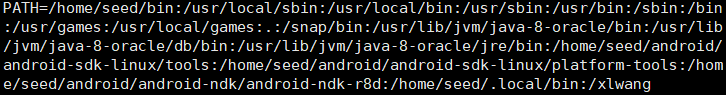
**step2：**



**step3：**



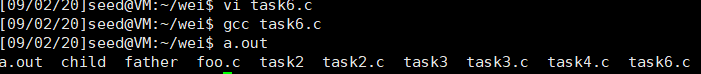




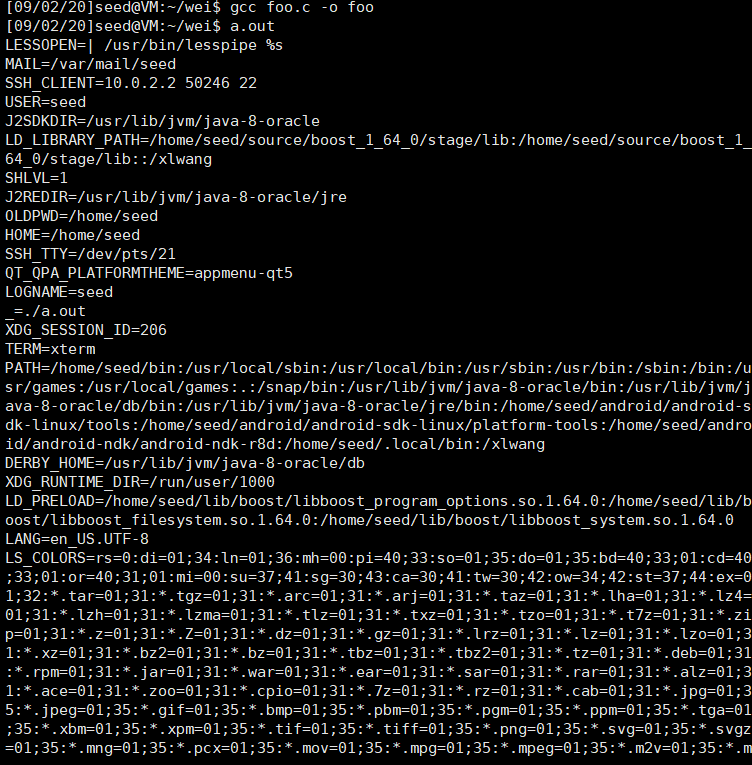
结论：被定义的3个环境变量全部被包括在shell中

**task6：**

编译执行程序发现程序执行了ls命令



将task5中的foo.c编译到foo，并将task6.c设置为SET-UID程序将system的参数改为foo，再将foo所在路径添加到SHELL变量开头，task6.c程序运行结果如下



可以看出task6.c执行了task5中的foo。

**task7：**

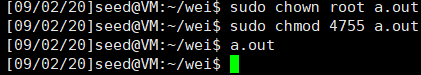
**step1：**

**step2：**

1）在普通用户身份下运行



2）在普通用户身份下运行SET-UID的myprog.c 无输出



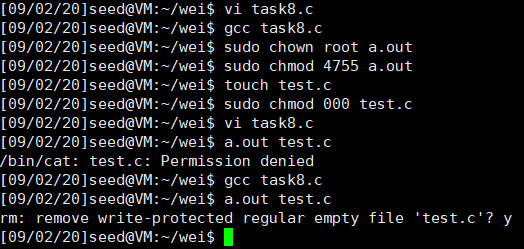
3）在user2用户中再次设置LD\_PRELOAD环境变量并运行myprog程序时，输出：I am not sleeping!

结论：第一次和第三次实myprog程序均具有seed用户权限，而在seed用户的LD\_PRELOAD环境变量中也添加了libmylib.so.1.0.1链接库。因此，这两个实验有输出。但第2次实验并未继承LD\_PRELOAD环境变量，所以无法调用动态链接库。

**task8：**

可以利用task8.c的SET-UID程序来删除本来没有权限删除的文件

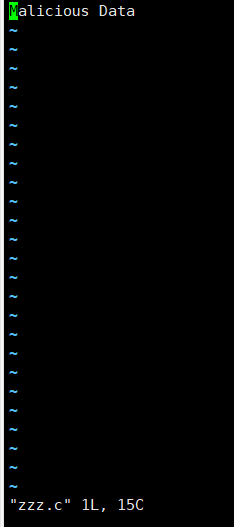
将v[0]改为rm命令再编译task8.c即可通过task8.c的可执行程序删除无写权限的文件



task9：







结论：task.9的可执行程序依然会修改zzz.c文件