1.关于滚屏实现

对于 shell 滚屏实现我们不做要求，如果没有实现的话，在写到最底的时候，同学手动执行或者 shell 自动执行 clear 清屏即可。关于滚屏和清屏的部分函数，我们在 screen.c 中已经提供给了大家，但是有同学反映可能存在 bug，这里我们就不修改了，想做相关功能的同学自己 debug 出来或者重新实现就行。

2 测试用例中的 INIT

在进行条件变量测试中，我们给出的代码里没有进行 condition variables 的初始化，请同学们自行添加初始化的调用。此外关于 IPC 通信的全局信箱的初始化，请同学们不要忘记，初始化的地方同学们请注意自行选择。

3.进程在调用同步原语后访问临界区时被 kill

在之前的 review 中，我们对进程被 kill 时应该进行什么操作进行了考核，大部分同学都知道要进行：释放 PCB、释放栈空间、释放锁、释放被阻塞进程。但对于信号量的处理，有一些同学对进程在调用同步原语访问临界区被杀死时的操作存在疑问，在这里我们给出三种方案，大家只要实现任意一种即可：

（1）什么都不做，直接杀死进程即可。

（2）被杀死进程所持有的信号量失效，当其他任务再次请求该信号量的时候，返回失败。

（3）进行信号量的回滚，既消除被杀死进程对信号量的影响，使得其他的进程可以继续使用该信号量并不出错。

对于回滚操作，不同的同步原语采取的措施不同，参考如下：

同步原语	回滚操作
条件变量	如果进程持有临界区，则通知并释放所有等待进程
信号量	记录该进程对信号量的操作（比如减了多少），在杀死时对这个值进行还原（加回去）
屏障	如果进程已进入等待队列，则直接将进程从等待队列中删除，并释放相应资源；否则，正常释放进程资源

上述三个措施，第一个无疑是最简单的，但如果学有余力的同学可以挑战一下第二、第三个措施。

4.shell 的实现

在昨天 review 的过程中很多同学反映 shell 的实现过程比较“困难”，我们和大家进行讨论了之后认为很多同学考虑问题过于复杂了，针对 shell 的实现，大家不需要考虑比较复杂的情况，比如：自动补全、容错性。大家只需要从我们提供的串口读取函数读取字符，然后存储到缓冲区，当读到一个回车（Enter）后，将缓冲区存储的字符和我们的命令进行解析，如果符合，就去调用相关的系统调用即可。关于 shell 的更多特性，比如滚屏等，我们不做要求。有兴趣的同学，我们鼓励去发挥并完善更多特性。