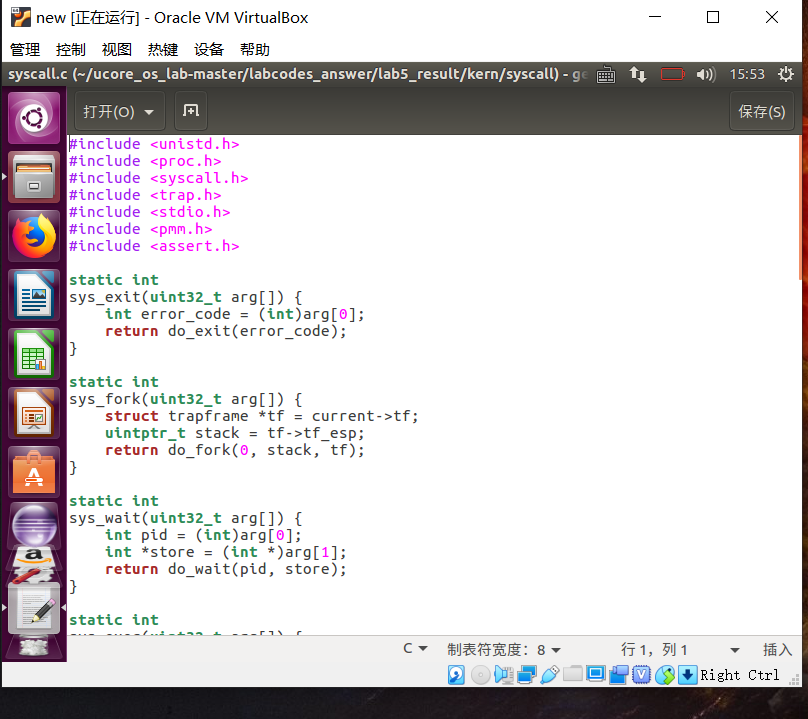
# 练习3：阅读分析源代码，理解进程执行 fork/exec/wait/exit 的实现，以及系统调用的实现（不需要编码）

fork：在执行了fork系统调用之后，会执行正常的中断处理流程，最终将控制权转移给syscall，之后根据系统调用号执行sys\_fork函数，进一步执行了上文中的do\_fork函数，完成新的进程的进程控制块的初始化、设置、以及将父进程内存中的内容到子进程的内存的复制工作，然后将新创建的进程放入可执行队列（runnable），这样的话在之后就有可能由调度器将子进程运行起来了；

exec：在执行了exec系统调用之后，会执行正常的中断处理流程，最终将控制权转移给syscall，之后根据系统调用号执行sys\_exec函数，进一步执行了上文中的do\_execve函数，在该函数中，会对内存空间进行清空，然后将新的要执行的程序加载到内存中，然后设置好中断帧，使得最终中断返回之后可以跳转到指定的应用程序的入口处，就可以正确执行了；

wait：在执行了wait系统调用之后，会执行正常的中断处理流程，最终将控制权转移给syscall，之后根据系统调用号执行sys\_wait函数，进一步执行了的do\_wait函数，在这个函数中，将搜索是否指定进程存在着处于ZOMBIE态的子进程，如果有的话直接将其占用的资源释放掉即可；如果找不到这种子进程，则将当前进程的状态改成SLEEPING态，并且标记为等待ZOMBIE态的子进程，然后调用schedule函数将其当前线程从CPU占用中切换出去，直到有对应的子进程结束来唤醒这个进程为止；

exit：在执行了exit系统调用之后，会执行正常的中断处理流程，最终将控制权转移给syscall，之后根据系统调用号执行sys\_exit函数，进一步执行了的do\_exit函数，首先将释放当前进程的大多数资源，然后将其标记为ZOMBIE态，然后调用wakeup\_proc函数将其父进程唤醒（如果父进程执行了wait进入SLEEPING态的话），然后调用schedule函数，让出CPU资源，等待父进程进一步完成其所有资源的回收；



# 练习1: 使用 Round Robin 调度算法（不需要编码）

第一部分Init：是初始化环节，初始化rq的进程队列，并将其进程数量置零。

第二部分enqueue：是一个进程入队的操作：进程队列是一个双向链表，一个进程加入队列的时候，会将其加入到队列的第一位，并给它初始数量的时间片；并更新队列的进程数量。第三部分dequeue：从就绪队列中取出这个进程，并将其调用list\_del\_init删除。同时，进程数量减一。

第四部分pick\_next：通过list\_next函数的调用，会从队尾选择一个进程，代表当前应该去执行的那个进程。如果选不出来有处在就绪状态的进程，那么返回NULL，并将执行权交给内核线程idle，idle的功能是不断调用schedule，直到整个系统出现下一个可以执行的进程。

第五部分：proc\_tick：产生时钟中断的时候，会触发tick函数的调用，对应于上图中调度点的第六种情况。

每次产生了时钟中断，代表时间片数量减一（因为中断和时间片的关系，在练习0的中断处理函数中修改，变得相关联）。

一旦时间片用完了，那么就需要把该进程PCB中的need\_resched置为1，代表它必须放弃对于CPU的占有，需要将别的进程调度进来执行，而当前进程需要等待了。

最后一步：在schedule初始化的时候，需要填写一个初始化信息，那么这里就填上我们所实现的类函数，那么系统就可以按照这个方式去执行了。