一、描述以下代码为Linux/Unix创建进程的例子，其中

\* fork克隆出一个进程

\* exec覆盖掉当前进程

|  |
| --- |
| If ((pid = fork()) == 0)  {  /\* child process \*/  exec(“foo”); /\* does not return \*/  else  /\* parent \*/  wait(pid); /\* wait for child to die \*/  } |

fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两个进程也可以做不同的事。

一个进程调用fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

但是，在不同的进程中，fork的返回值不同。

 1）在父进程中，fork返回新创建子进程的进程ID；  
    2）在子进程中，fork返回0；  
    3）如果出现错误，fork返回一个负值；

exec函数族提供了一个在进程中启动另一个程序执行的方法。它可以根据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的数据段、代码段和堆栈段，在执行完之后，原调用进程的内容除了进程号外，其他全部被新程序的内容替换了。

wait函数提供了多进程之间跳转的一种方式。进程一旦调用了wait，就立即阻塞自己，由wait自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个已经变成僵尸的子进程，wait就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait就会一直阻塞在这里，直到有一个出现为止。（僵尸进程是当子进程比父进程先结束，而父进程又没有回收子进程，释放子进程占用的资源，此时子进程将成为一个僵尸进程。）

在示例程序中，首先执行pid = fork()建立一个子进程。建立成功后，在父进程中pid返回值为子进程的进程号，在子进程中返回0。随后执行如下语句：

1. 在父进程中，执行语句wait(pid)，等待子进程运行结束并返回。
2. 在子进程中，执行exec(“foo“)，用foo覆盖掉当前子进程。

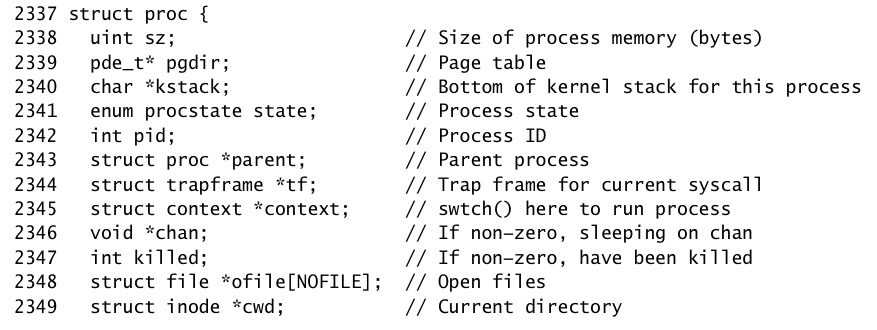
wait函数不断扫描子进程的PCB，确定子进程的状态。如果子进程仍然在运行，就调用sleep函数暂时进入睡眠状态。如此重复，直到子进程进入僵尸状态，wait函数会把子进程的栈和函数释放，初始化PCB，并杀死进程。

二、请大家阅读xv6源码，撰写一份分析报告。报告需要给出以下信息：

1. 在xv6中，PCB信息是如何表示的？是用什么数据结构？存放在哪个文件中？

2. 上述代码中，修改了哪些PCB中的信息？

1. 在xv6中，PCB信息存储在结构体proc中。proc定义在proc.h文件下，各进程的结构体以数组方式统一保存在proc.c文件下进程表 ptable 内的数组 struct proc proc[NPROC] 中。



2350 char name[16]; //Process name (debugging)

2351 };

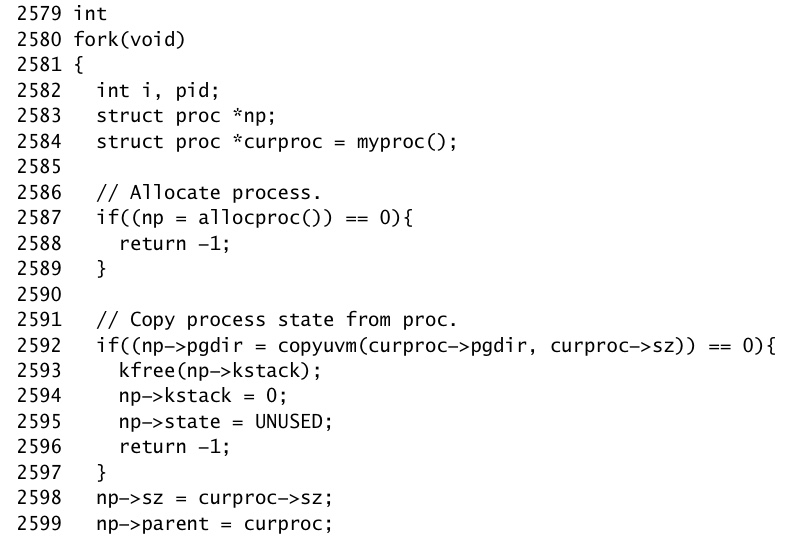
2409 struct{

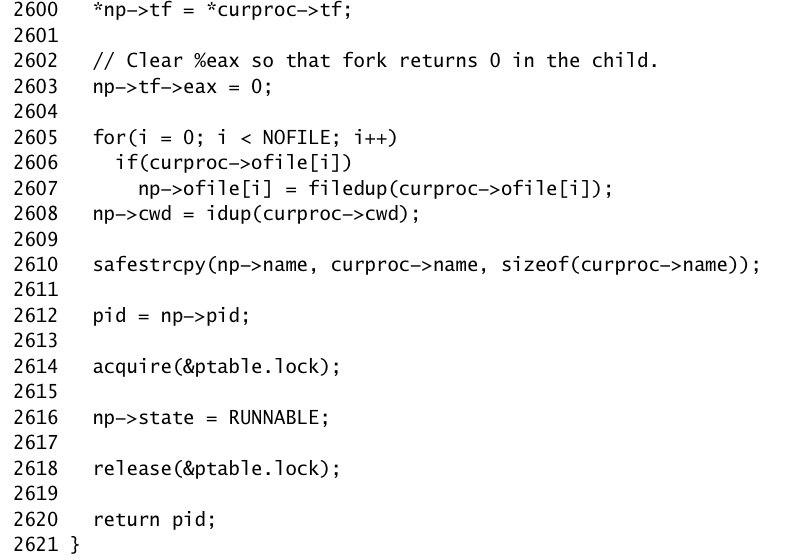
2410 struct spinlock lock;

2411 struct proc proc[NPROC];

2412 } ptable;

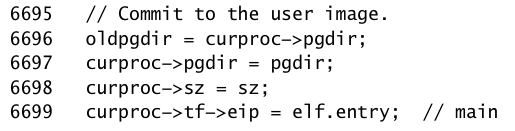
2.首先来看看fork函数在xv6中的源码：





fork（）首先调用 allocproc() 创建一个新的子进程，随后开始复制父进程的PCB。先使用cupyuvm赋值父进程的页表(\*pgdir)和内存空间大小(sz)到子进程的PCB中。如果复制成功，将父进程的curproc放入\*parent中，接着复制父进程的陷阱帧（\*tf）。之后，将%eax寄存器清零，从而确保子进程的返回值为0。最后复制打开的文件流（\*ofile），当前目录（\*cwd），进程名（name），在父进程中设置pid返回值为子进程ID，设置子进程状态为RUNNABLE。

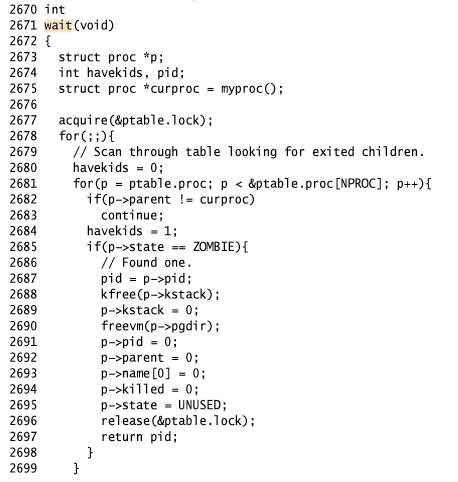
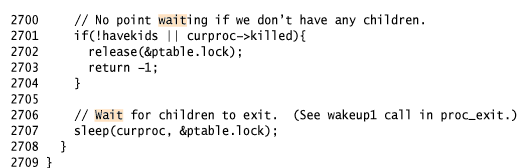
再来看exec（）函数的部分代码：



6700 curproc->tf->sep = sp;

exec函数在执行时会用foo覆盖掉当前的子进程。对PCB的操作主要为页表、内存空间大小、入口地址以及栈帧寄存器中值的替换。

最后参考wait（）函数的代码：



wait函数首先对进程表进行搜索，如果没找到子进程就返回-1。如果找到了就检测该子进程是不是僵尸状态。如果是，就释放子进程的栈空间、页表、ID、parent、name，杀死进程并修改状态为UNUSED，并返回被杀死的子进程的ID。如果没有找到处于僵尸态的子进程，就进入睡眠状态，直到进程表中有新的状态变化后重新执行上述过程。