Exercise 2

1352841 洪庆

1.Is the COW technique same with running in the parent's address space ?

这两者并不一样。
COW(CopyOnWrite)内核只为新生成的子进程创建虚拟空间结构，它们复制父进程的虚拟空间结构，但是不为这些段分配物理内存，它们共享父进程的物理空间，当父子进程中有更改相应段的行为发生时，再为子进程相应的段分配物理空间，并将父进程的内容复制过来进行修改。为的是减少不必要的开销。

运行在父进程地址空间意味着父子进程都会使用同样的数据，任何一方修改，另一方也能觉察到。

2.What are the differences between fork and vfork ? Write a program to verify your answer.

区别：
1.vfork保证子进程先运行，在它调用exec或exit之后父进程才可能被调度运行。如果在调用这两个函数之前子进程依赖于父进程的进一步动作，则会导致死锁。而fork的子线程和父线程的执行顺序是不一定的。

2.fork要拷贝父进程的进程环境；而vfork则不需要完全拷贝父进程的进程环境，在子进程没有调用exec和exit之前，子进程与父进程共享进程环境，相当于线程的概念，此时父进程阻塞等待。

出现*vfork*的原因：
因为以前的fork当它创建一个子进程时，将会创建一个新的地址空间，并且拷贝父进程的资源，然后将会有两种行为：

1.执行从父进程那里拷贝过来的代码段
2.调用一个exec执行一个新的代码段

因此，如果创建子进程是为了调用exec执行一个新的程序的时候，就应该使用vfork

验证代码运行如下：
运行如下命令：

(1)执行fork
```
gcc -o fork fork.c
./fork
```

```
before fork

parent process
pid=1849, global=1314, local=520

child process
pid=1850, global=1315, local=519
```

(2)执行vfork
gcc -o vfork vfork.c
./vfork

```
before vfork

child process
pid=1867, global=234, local=665

parent process
pid=1866, global=234, local=665
```

由运行结果可知，执行fork程序的时候先执行的是父进程，再执行的是子线程（由shell中显示的顺序可知），而执行vfork程序后，是先执行子线程，后执行父线程，符合区别1；并且从global和local的值可以看出来，执行vfork后，先执行的子线程改变的global和local的值也在父线程中显示出来，说明，vfork与父线程共享进程环境，而fork则不会，因为它是拷贝的父进程的资源数据。

3.Write a program to show the child process wait the parent process with a block style and a noblock style to terminate.Then show the changing of memory layout.

with a block style:

```
before waiting
start child process
child process: pid = 2246, ppid = 2245, pgid = 2245
after waiting
child process exited.parent process: pid = 2245, ppid = 1130, pgid = 2245
```

with a no-block style:

```
before waiting
the child process has not exited!
start child process
the child process has not exited!
the child process has not exited!
the child process has not exited!
the child process has not exited!
the child process has not exited!
child process: pid = 2251, ppid = 2250, pgid = 2250
after waiting
child process exited.parent process: pid = 2250, ppid = 1130, pgid = 2250
```

从以上实验可以看出两个概念的区别：

阻塞操作是指在没有执行设备操作时，如果没有获得资源，则进程挂起，直到满足可操作的条件再进行操作。非阻塞操作的进程在不能进行设备操作时，并不挂起。