**实验报告**

**Lab 0**

**Exercise1: Implement the UNIX program sleep for xv6; your sleep should pause for a user-specified number of ticks. A tick is a notion of time defined by the xv6 kernel, namely the time between two interrupts from the timer chip. Your solution should be in the file user/sleep.c.**

**完成sleep程序。**

**实验原理:**

利用system call **sleep**即可完成。

int sleep(int n)：**Pause for n clock ticks**

sleep函数的声明位于user.h中:

// system calls

...

int sleep(int);

在sysproc.c中提供了xv6内核实现sleep system call的方式:

int

sys\_sleep(void)

{

  int n;

  uint ticks0;

  if(argint(0, &n) < 0)

    return -1;

  acquire(&tickslock);

  ticks0 = ticks;

  while(ticks - ticks0 < n){

    if(myproc()->killed){

      release(&tickslock);

      return -1;

    }

    sleep(&ticks, &tickslock);

  }

  release(&tickslock);

  return 0;

}

在usys.S中提供了将用户代码转换内核实现的汇编码:

#define SYSCALL(name) \

.globl name; \

name: \

movl $SYS\_ ## name, %eax; \

int $T\_SYSCALL; \

ret

**实验过程:**

根据提示，参考echo.c文件的编写：

#include "types.h"

#include "stat.h"

#include "user.h"

int

main(int argc, char \*argv[])

{

  int i;

  for(i = 1; i < argc; i++)

    printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");

  exit();

}

需要注意的点如下:

1. main函数接收2个参数，第一个argc是实际提供的参数个数，第二个argv是传给主函数的字符串数组。通过命令行提供的参数就以这样的方式传递给main函数。

2.printf与寻常C语言中使用的不同，第一个参数需要接受一个int类型变量，查看printf.c可知：

// Print to the given fd. Only understands %d, %x, %p, %s.

void

printf(int fd, const char \*fmt, ...)

该int变量为fd(file descriptor)，即文件描述符，是一个可以代表进程可以对其读写的内核管理对象的整数。一般来讲，整数0代表标准输入，1代表标准输出，2代表标准错误。

3.在完成功能后需要调用exit()函数。

仿照echo.c，从而我们可以写出sleep.c的内容：

#include "types.h"

#include "stat.h"

#include "user.h"

int

main(int argc, char \*argv[])

{

  if(argc<=1)            //没有输入参数

  {

    printf(2,"error:argument numbers not match\n");

    exit();

  }

  int time=atoi(argv[1]);    //用atoi函数转换字符串为整数

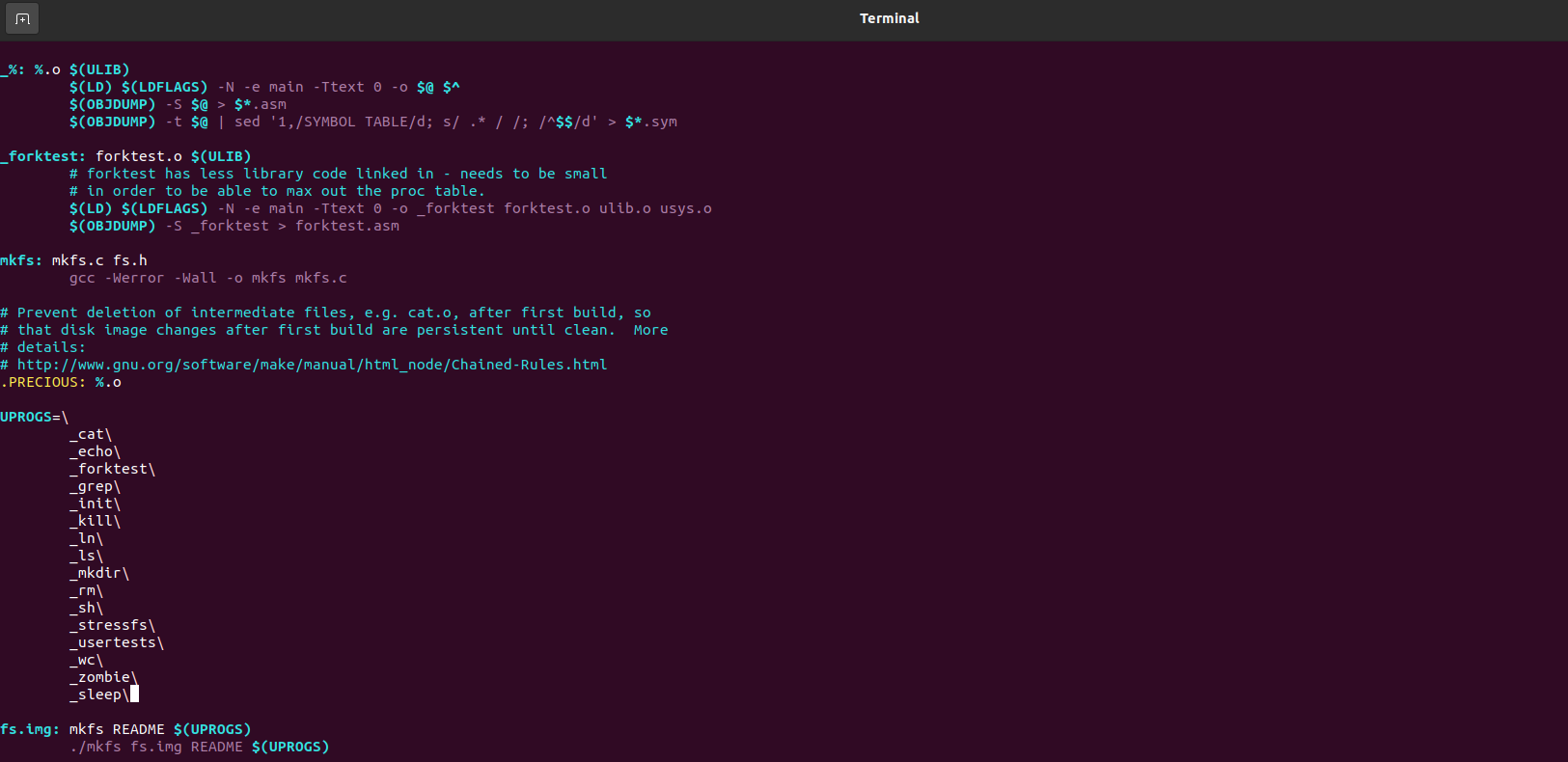
  sleep(time);               //调用sleep函数

  printf(1,"(nothing happens for a little while)\n");

  exit();

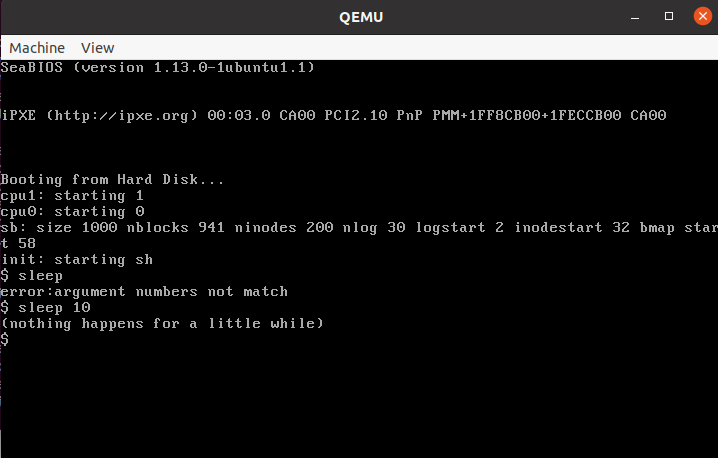
}

写完后若直接在qemu中输入”sleep 10”则会提示没有这条指令。因为我们还没有在makefile中的UPROGS加入该程序。找到makefile，用vim进行修改，在UPROGS末尾添加\_sleep\:



在插入模式下修改完成后回到命令模式，输入:wq退出并保存。

之后就能在qemu中运行sleep程序了，效果如下。



**Exercise2: Write a program that uses UNIX system calls to ''ping-pong'' a byte between two processes over a pair of pipes, one for each direction. The parent should send a byte to the child; the child should print "<pid>: received ping", where <pid> is its process ID, write the byte on the pipe to the parent, and exit; the parent should read the byte from the child, print "<pid>: received pong", and exit. Your solution should be in the file user/pingpong.c.**

**写一个程序使其能让两个进程间通过一对pipe互相发送字节。父进程发送给子进程一个字节，子进程需要打印出received ping，并写字节到pipe上给父进程再退出；父进程读取子进程发送的字节，打印received pong，并退出。**

**实验原理:**

1.fork()函数能创建一个当前进程的拷贝（称为子进程），并返回子进程的PID。对于父进程来说，返回的就是其子进程的PID；对于子进程来说，因为其没有子进程，返回的是0（失败返回-1）。注意fork函数调用一次，但要返回两次（分别在父进程和子进程中）。

2.pipe（管道）是一个小的内核缓存，它是一对暴露给进程的文件描述符，一个负责读一个负责写。向pipe的一端写使得在另一端可以读这些数据。pipe提供了进程间通信的一种方式。

3.read(int fd, char \*buf, int n) 从文件描述符fd读取n个字节到buf中，返回读取成功的字符数。

4.write(int fd,char \*buf,int n) 从buf写入n个字节到文件描述符fd，返回n。

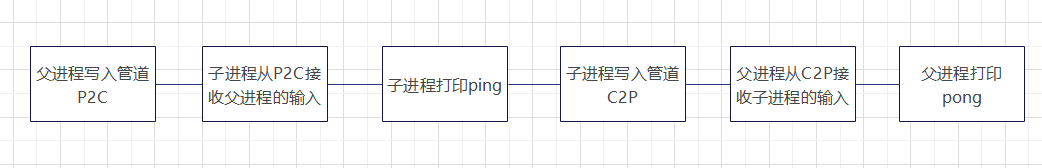
5.pipe(int p[]) 创建一个pipe，并将p[0]和p[1]分别作为读取和写入文件描述符。

6.getpid()能获取当前进程的PID。

**实验过程**:

关键在于如何正确理解fork函数的用途。实际上，它相当于将原本的进程一分为二，且两个进程基本一致（存在父子关系），二者同步运行。因此，为完成题目的要求我们只需要分别在父进程和子进程中规定各自需要做的事即可。

梳理执行逻辑如下:



代码:

#include "types.h"

#include "stat.h"

#include "user.h"

int

main(int argc, char \*argv[])

{

  int P2C[2];               //the pipe that parent send byte to child

  int C2P[2];               //the pipe that child send byte to parent

  char buffer[1]={'1'};    //one byte

  pipe(P2C);

  pipe(C2P);

  if(fork()==0)             //child process

  {

    close(P2C[1]);

    close(C2P[0]);

    read(P2C[0],buffer,1);    //step2

    close(P2C[0]);

printf(1,"%d:received ping\n",getpid());    //step3

    write(C2P[1],buffer,1);        //step4

    close(C2P[1]);

    exit();

  }

  else                      //parent process

  {

    close(C2P[1]);

    close(P2C[0]);

write(P2C[1],buffer,1);    //step1

close(P2C[1]);

    read(C2P[0],buffer,1);    //step5

    close(C2P[0]);

    printf(1,"%d:received pong\n",getpid());      //step6

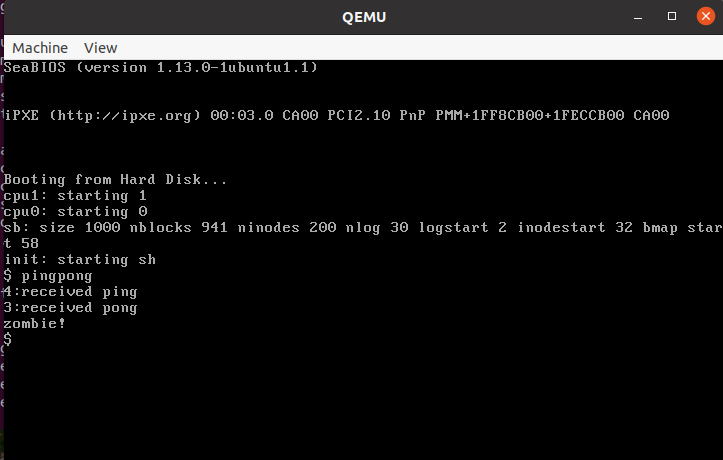
    exit();

  }

}

通过fork函数的返回值是否为0来判断是子进程还是父进程。这里注意close函数的使用。根据说明，close(int fd)释放文件描述符fd。如果不及时释放有可能产生堵塞。虽然本程序并不涉及堵塞的情况，不写这些close也可以，但最好还是及时释放掉不用的文件描述符。

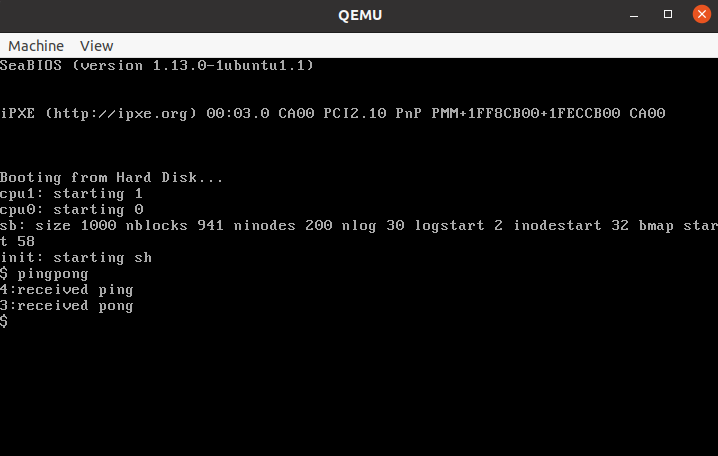
结果如下:



其中4代表了子进程的PID，3代表了父进程的PID。这是符合预期的，因为我们按Ctrl+P就能看到，1代表init进程，2代表sh进程。



我们注意到最后多出了一句”zombie!”，这实际上代表了僵死进程，因为我们的子进程比父进程更早退出，使子进程没有彻底被销毁。为解决这个问题，我们应该在父进程（即else分支）退出前加一句wait()。wait函数使父进程等待直到子进程退出，并将其彻底销毁。修改后的实验结果如下:

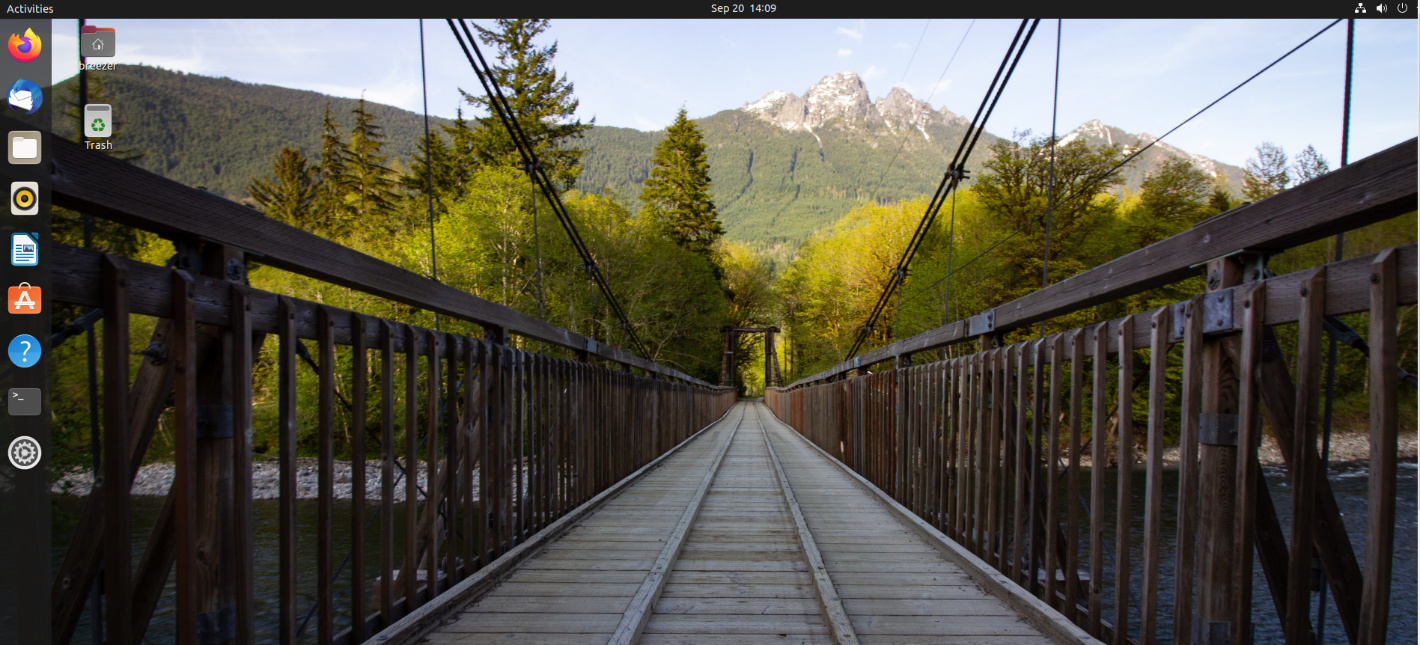


可以看到zombie语句消失了。

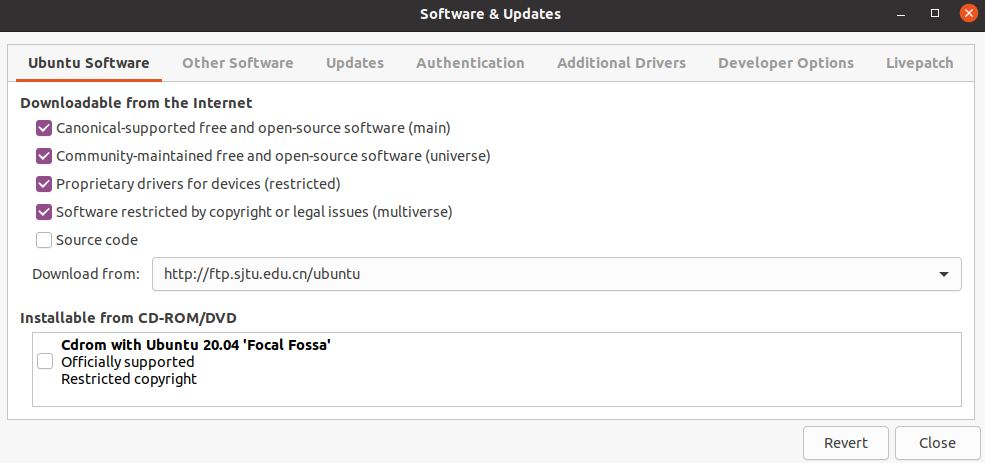
**Part 1：Environment Configuration**

**Exercise 1：**Please install your own environment, attach a screenshot of the process and describe it in detail.

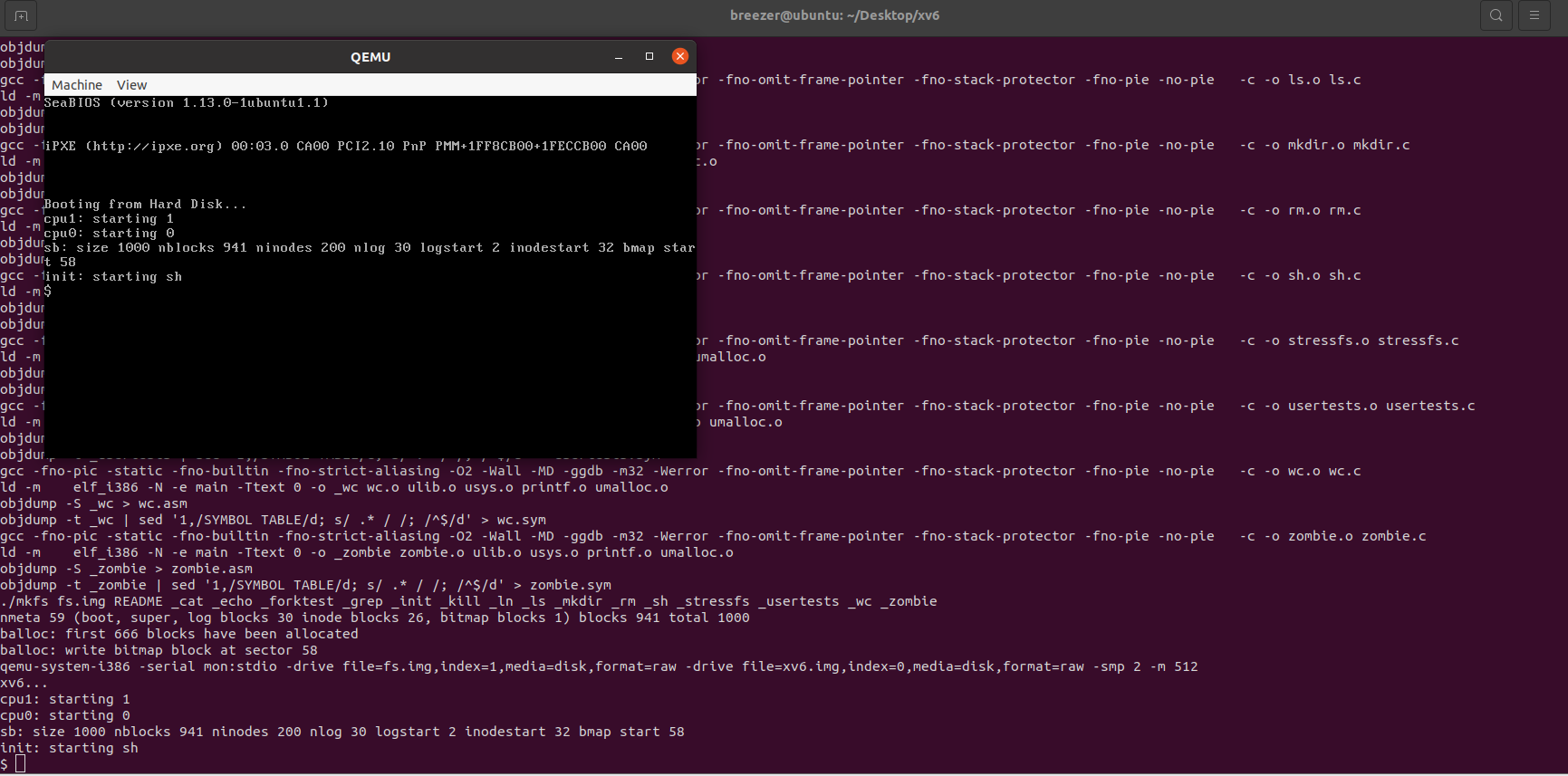
在装好Vmware WorkStation后，下载Ubuntu20.04镜像，并创建新虚拟机。装好后如图。



先在Software&Updates中换源，否则下载软件非常慢。可以通过select best server的方法自动选择。选好后退出并在命令行中用sudo apt-get update更新。



按照PPT给出的指示安装qemu，再在命令行中编译启动xv6系统，完成后结果如下。



**Exercise 2 ：**What problems did you encounter, how did you solve them, and if you didn't encounter any problems, what did you learn during the installation process？

遇到的一些问题如下：

①使用的printf等函数与C语言中常用的定义不一致，需要自行查找相关定义理解。

②pingpong部分涉及到的fork函数、父进程、子进程等概念较为复杂，程序跑错了很多次，也查阅了很多相关资料。

③gdb的使用方法没有看懂，在网上看了一些教程后逐渐掌握了。

**Part 2：Some More Questions about Configuration**

**Exercise 3 ：**What does “operating system virtualization” mean and what important uses does it have？

虚拟化操作系统指的是在物理计算机系统上通过软件模拟虚拟的计算机系统。它允许多个操作系统独立并行存在，彼此不会产生影响。因此可以在虚拟操作系统中进行各种操作而不用担心影响物理系统。

**Exercise 4：**What is Qemu? What are its advantages and disadvantages? What are xv6 and jos?

Qemu是一套硬件模拟器。其优点在于能方便地模拟多种硬件架构，缺点是对某些操作系统支持不够完善，模拟速度一般。

xv6和jos都是用于教学的实验性质的操作系统。xv6比较完整全面，jos更简单。

**Exercise 5：**What's the difference between Qemu and VMware or Virtual Box?

VMware和Virtual Box都是虚拟机软件，用于制造虚拟操作系统。而Qemu是硬件模拟器，是运作在操作系统上模拟硬件架构的。这两类软件的模拟内容不一样。

**Exercise 6：**What does the "make" directive mean? What is a makefile？

make指令读入makefile文件，执行makefile中的指令。

makefile是这样一个文件，它指定了一系列文件的编译、链接等的规则，如编译的顺序、编译的方式等。

**Part 3：The Ending**

**Exercise 7：**Please tell me what you learned from this experiment, or give a summary of the experiment.

从本次实验中，我了解了xv6系统的一些基本概念，如文件描述符(File Descriptor)、PID、fork函数等；熟悉了Linux命令行的使用，掌握了如何利用gdb进行调试；通过阅读系统源码，也对系统内核有了进一步的认知。