**实验报告**

**Lab 0**

姓名：陈思芃

班级：20级人工智能

学号：20307140002

**实验报告填写要求**

1.请在每个exercise之后简要叙述实验原理，详细描述实验过程。可以使用中文进行描述，不对语言做要求。

2.请将你认为的关键步骤附上必要的截图。

3.有需要写代码的实验，必须配有代码、注释以及对代码功能的说明。

4.你还可以列举包括但不局限于以下方面:实验过程中碰到的问题你是如何解决的、实验之后你还留有哪些疑问和感想。

5.如果实验附有练习，请在每个练习之后作答，这是实验报告评分的重要部分。

6.Challenge为加分选作题。每个lab可能有多个challenge,我们会根据完成情况以及难度适当加分，具体情况会在课上说明。这部分的实验过程描述应该比exercise更加详细。（请注意，Lab0为基础环境配置，不设置挑战问题。）

7.切勿抄袭亦或是去互联网复制粘贴答案。

【练习题模板】

1. Question
2. Code
3. Screenshot
4. Difficulties and solutions

**Q1**

1. **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.**

#include "types.h"

#include "stat.h"

#include "user.h"

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

int arg;

if (argc < 2){

**printf**(2, "sleep ERROR: too few arguments.\n");

}

else{

arg = **atoi**(\*argv); *//Morph argv into integer type using function atoi*

**printf**(2, "(Nothing happens for a little while...)\n");

**sleep**(arg);

}

**exit**();

}

****

1. Figure out the function used in constructing sleep.

**Q2**

1. **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.**

#include "types.h"

#include "stat.h"

#include "user.h"

int **main**(){

int ptoc[2], ctop[2]; *//Array that is to store file descriptors in*

int pid = 0;

char bufa[512], bufb[512]; *//Content buffer*

**pipe**(ptoc); *//Generating a pair of pipes*

**pipe**(ctop);

if (**fork**() == 0){ *//Create a child process*

**close**(ptoc[1]); *//Close the writing end from parent to child...*

**close**(ctop[0]); *//and the reading end from child to parent*

**read**(ptoc[0], bufb, 1); *//Receive the signal from the parent...*

**write**(ctop[1], bufa, 1); *//and send out a signal*

**close**(ptoc[0]); *//Close the remaining ends*

**close**(ctop[1]);

pid = **getpid**();

**printf**(2, "%d: received ping\n", pid);

}

else{

**close**(ctop[0]); *//Close the writing end from child to parent...*

**close**(ptoc[1]); *//and the reading end from parent to child*

**read**(ctop[0], bufa, 1); *//Receive the signal from the child...*

**write**(ptoc[1], bufb, 1); *//and send out a signal*

pid = **getpid**();

**printf**(2, "%d: received pong\n", pid);

**close**(ctop[0]);

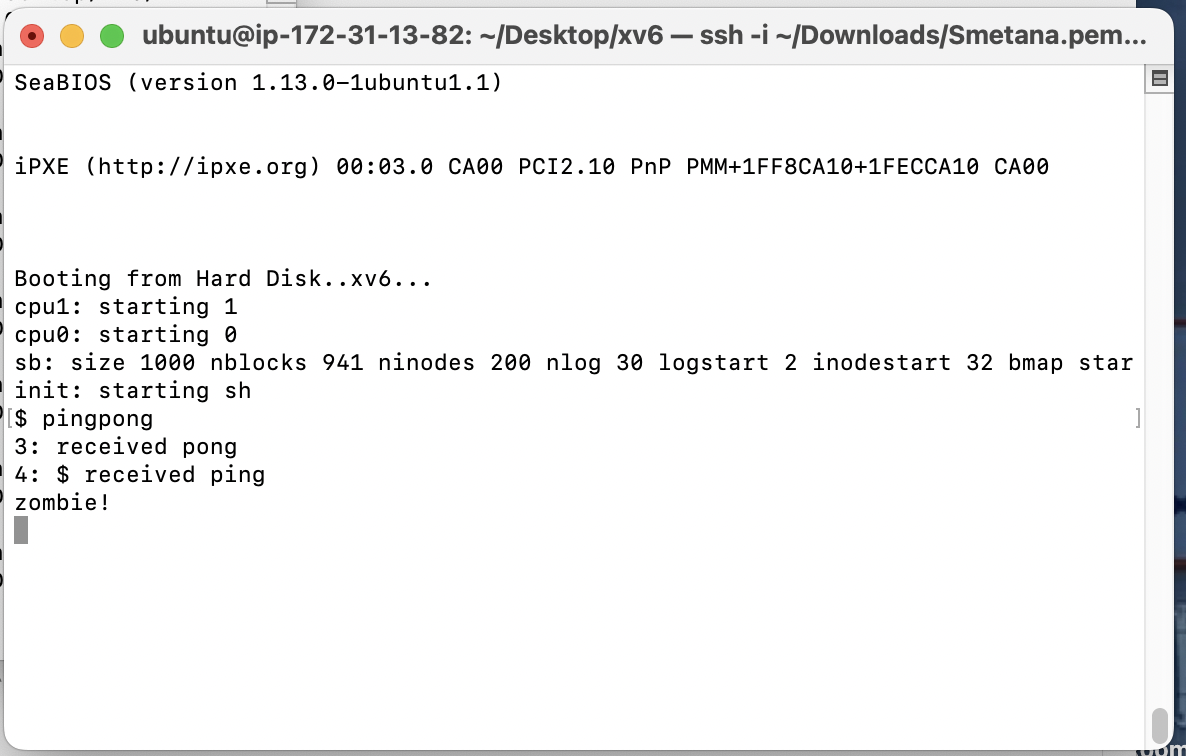
**close**(ptoc[1]);

}

**exit**();

}





1. It’s kind of hard to understand how the pipe works, like how pipe links its two ends with the parent and the child respectively. The first time I tried to do this I didn’t close the ends that will not be used in either process and ended up getting only one print from the two processes, so I tried to invalidate some of the codes to pin down the problem and managed to solve it.

**Part 1：Environment Configuration**

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

Basically I created an instance of ubuntu on an Amazon EC2 server and then I simply followed the instructions on how to install qemu. A few errors occurred and I will elaborate on how I solved them in Exercise 2. I pre-installed all the stuff required before the lab on Tuesday and unfortunately I wasn’t informed that I should take screenshots of the process of the installation.

**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？

The biggest problem I’ve encountered was that my macbook is equipped with M1 chip and the architecture of it is ARM64. This means that I wouldn’t be able to run x86 ubuntu even with VM. Fortunately I found out that Amazon Web Services were providing new users with a 12-month free trial which gave me access to linux servers for free. Apart from that, in the process of actual installation, when I ran “make qemu” on the terminal of the server, error report popped out indicating that I was “writing 1 byte into a region of size 0”. I tried a few solutions from stackoverflow, but they all failed, so my guess was that I had installed wrong version of ubuntu because it couldn’t have been because of conflicts between packages since there was literally nothing on that server before I had qemu installed. So I deleted this instance which had ubuntu 22.04 and opened up a new one with ubuntu 20.04. This time the terminal again reported error again yet with different info. I searched it up and found out that it was because the server I was using then didn’t have GUI installed, so I followed the instruction on how to use qemu on a non-GUI OS and added “-nographic” on some spots in Makefile, and qemu started to run properly.

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

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

OS virtualization creates segregated instances that interact directly with human users. Only the data inside these specific instances and the hardware devices assigned to them are visible to these instances. This allows different users to make the best of the hardware resources but at the same time not interrupting each other as their workspaces are separated.

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

Qemu is an emulator that emulates the processor, capacitating the machine to run a variety of guest operating systems. It supports a wide range of architectures and but it also suffer from disadvantages like lack of GUI and slow speed. Xv6 and jos are both reimplementation of UNIX-6.

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

Essentially, qemu is both an emulator and a virtualization software, while the latter two are mere virtualization software. An emulator is expected to emulate the physical signal of another machine with different hardware config, while virtualization software only needs to mimic logically.

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

“make” is a software that builds executable programs according to a makefile which is supposed to elucidate how to construct the target file.

**Part 3：The Ending**

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

The experiment guided me through the process of installation of ubuntu, qemu, xv6 and jos. I also get a quick glimpse of how some basic elements work in UNIX such as how to use fork() to create a child process, how the child and the parent proceed concurrently, and how pipes work when it comes to transmission of data in between different processes.