@[TOC](Mit6.S081-实验1-Xv6 and Unix utilities)

# 前言

# 一、Boot xv6

## 1，实验目的

利用qemu启动xv6

## 2，操作流程

### 1）切换到xv6-labs-2020代码库的lab1分支

```bash

git checkout util

```

参考：https://pdos.csail.mit.edu/6.828/2020/labs/util.html

### 2）启动xv6

```bash

make qemu

```

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200919145316582.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

参考：https://pdos.csail.mit.edu/6.828/2020/labs/util.html

### 3）测试xv6

```bash

ls

```

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200919145410932.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

参考：https://pdos.csail.mit.edu/6.828/2020/labs/util.html

### 4）过程分析

xv6通过qemu启动完成后，启动了shell进程。通过shell，启动子进程ls，显示了xv6目录下的文件。

### 5）其他操作

查看xv6中的进程：Ctrl+p（xv6没有实现ps程序）

退出qemu启动的xv6：Ctrl+a x

# 二、在xv6中添加一个自己编写的程序

## 1，源码准备

在user目录下创建copy.c

```c

// copy.c: 将控制台输入内容输出到控制台

#include "kernel/types.h"

#include "user/user.h"

int

main()

{

char buf[64];

while(1){

//从console读取输入，通过system call的read函数实现

int n = read(0, buf, sizeof(buf));

//无输入结束程序

if(n <= 0)

break;

//将console输入输出到控制台，通过system call的write函数实现

write(1, buf, n);

}

exit(0);

}

```

参考：https://pdos.csail.mit.edu/6.828/2020/lec/l-overview/copy.c

## 2，编译配置

在Makefile 152行添加配置

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200919151509556.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 3，测试添加程序

启动xv6后，执行copy

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200919152013683.png#pic\_center)

## 4，过程分析

xv6通过qemu启动完成后，启动了shell进程。通过shell，启动子进程copy。

# 三、xv6中shell简析

shell也是一个用户进程，不过在xv6启动后就运行了。源码为user/sh.c，入口函数在145行。

在shell下输入命令行，shell程序读取输入内容，通过调用fork（system call）开启一个shell的子进程，

shell进程利用wait（system call），等待子进程执行完后继续执行。

```c

//创建子进程

//父进程中fork1()返回创建的子进程pid；子进程中fork1执行成功返回0。所以子进程才会执行runcmd()

if(fork1() == 0)

//在子进程中执行shell中输入的程序

runcmd(parsecmd(buf));

//父进程等待子进程执行结束

wait(0);

```

# 四、ping pong练习

## 1，实验要求

写一个程序，使用unix system calls在两个进程间”ping-pong“一个字节，使用一对pipe，一个pipe对应一个方向，另外一个pipe对应另外一个方向。

## 2，源码

```c

#include <kernel/types.h>

#include <user/user.h>

/\*

xv6可运行

chapter01: ping pong练习程序

\*/

int main(){

//pipe1(p1)：写端父进程，读端子进程

//pipe2(p2)；写端子进程，读端父进程

int p1[2],p2[2];

//来回传输的字符数组：一个字节

char buffer[] = {'X'};

//传输字符数组的长度

long length = sizeof(buffer);

//父进程写，子进程读的pipe

pipe(p1);

//子进程写，父进程读的pipe

pipe(p2);

//子进程

if(fork() == 0){

//关掉不用的p1[1]、p2[0]

close(p1[1]);

close(p2[0]);

//子进程从pipe1的读端，读取字符数组

if(read(p1[0], buffer, length) != length){

printf("a--->b read error!");

exit(1);

}

//打印读取到的字符数组

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

//子进程向pipe2的写端，写入字符数组

if(write(p2[1], buffer, length) != length){

printf("a<---b write error!");

exit(1);

}

exit(0);

}

//关掉不用的p1[0]、p2[1]

close(p1[0]);

close(p2[1]);

//父进程向pipe1的写端，写入字符数组

if(write(p1[1], buffer, length) != length){

printf("a--->b write error!");

exit(1);

}

//父进程从pipe2的读端，读取字符数组

if(read(p2[0], buffer, length) != length){

printf("a<---b read error!");

exit(1);

}

//打印读取的字符数组

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

//等待进程子退出

wait(0);

exit(0);

}

```

## 3，辅助图

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200923171420335.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 4，执行效果

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200923171524970.png#pic\_center)

## 5，测试效果

在xv6-labs-2020中，执行下面指令，测试程序

```bash

./grade-lab-util pingpong

```

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200923171631229.png#pic\_center)

# 四、sleep实验

## 1，实验要求

写一个用户程序，调用sleep system call实现，执行sleep 10，表示程序等待10个时钟周期。

## 2，源码

```c

#include "kernel/types.h"

#include "user/user.h"

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

if(argn != 2){

fprintf(2, "must 1 argument for sleep\n");

exit(1);

}

int sleepNum = atoi(argv[1]);

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

sleep(sleepNum);

exit(0);

}

```

## 3，测试结果

在xv6-labs-2020中，执行下面指令，测试程序

```bash

./grade-lab-util sleep

```

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200923134048926.png#pic\_center)

# 五、primes实验

## 1，实验要求

将2-35中的素数打印出来，要求利用管道理念。

## 2，源码

```c

#include "kernel/types.h"

#include "user/user.h"

void func(int \*input, int num){

if(num == 1){

printf("prime %d\n", \*input);

return;

}

int p[2],i;

int prime = \*input;

int temp;

printf("prime %d\n", prime);

pipe(p);

if(fork() == 0){

for(i = 0; i < num; i++){

temp = \*(input + i);

write(p[1], (char \*)(&temp), 4);

}

exit(0);

}

close(p[1]);

if(fork() == 0){

int counter = 0;

char buffer[4];

while(read(p[0], buffer, 4) != 0){

temp = \*((int \*)buffer);

if(temp % prime != 0){

\*input = temp;

input += 1;

counter++;

}

}

func(input - counter, counter);

exit(0);

}

wait(0);

wait(0);

}

int main(){

int input[34];

int i = 0;

for(; i < 34; i++){

input[i] = i+2;

}

func(input, 34);

exit(0);

}

```

## 3，执行结果

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020092321294371.png#pic\_center)

## 4，测试结果

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200923213018843.png#pic\_center)

# 六、find实验

## 1，实验目的

实现find <path> <fileName>，即在某个路径中，找出某个文件

## 2，源码

```c

#include "kernel/types.h"

#include "kernel/fcntl.h"

#include "kernel/stat.h"

#include "kernel/fs.h"

#include "user/user.h"

/\*

将路径格式化为文件名

\*/

char\* fmt\_name(char \*path){

static char buf[DIRSIZ+1];

char \*p;

// Find first character after last slash.

for(p=path+strlen(path); p >= path && \*p != '/'; p--);

p++;

memmove(buf, p, strlen(p)+1);

return buf;

}

/\*

系统文件名与要查找的文件名，若一致，打印系统文件完整路径

\*/

void eq\_print(char \*fileName, char \*findName){

if(strcmp(fmt\_name(fileName), findName) == 0){

printf("%s\n", fileName);

}

}

/\*

在某路径中查找某文件

\*/

void find(char \*path, char \*findName){

int fd;

struct stat st;

if((fd = open(path, O\_RDONLY)) < 0){

fprintf(2, "find: cannot open %s\n", path);

return;

}

if(fstat(fd, &st) < 0){

fprintf(2, "find: cannot stat %s\n", path);

close(fd);

return;

}

char buf[512], \*p;

struct dirent de;

switch(st.type){

case T\_FILE:

eq\_print(path, findName);

break;

case T\_DIR:

if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){

printf("find: path too long\n");

break;

}

strcpy(buf, path);

p = buf+strlen(buf);

\*p++ = '/';

while(read(fd, &de, sizeof(de)) == sizeof(de)){

//printf("de.name:%s, de.inum:%d\n", de.name, de.inum);

if(de.inum == 0 || de.inum == 1 || strcmp(de.name, ".")==0 || strcmp(de.name, "..")==0)

continue;

memmove(p, de.name, strlen(de.name));

p[strlen(de.name)] = 0;

find(buf, findName);

}

break;

}

close(fd);

}

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

if(argc < 3){

printf("find: find <path> <fileName>\n");

exit(0);

}

find(argv[1], argv[2]);

exit(0);

}

```

## 3，执行结果

find当前目录下文件名为ls的文件

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200924140036428.png#pic\_center)

## 4，测试结果

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200924161956558.png#pic\_center)

# 七、xargs实验

## 1，实验目的

实现类似unix xargs类似功能，比如echo hello too|xargs echo bye，要输出bye hello too；

即等价于echo bye hello too，将上个命令输出的每行作为参数，拼接到xargs后面的指令后面。

echo hello too输出为hello too，将其拼接到echo bye后面，就是echo bye hello too。

## 2，源码

```c

#include "kernel/types.h"

#include "user/user.h"

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

int i;

int j = 0;

int k;

int l,m = 0;

char block[32];

char buf[32];

char \*p = buf;

char \*lineSplit[32];

for(i = 1; i < argc; i++){

lineSplit[j++] = argv[i];

}

while( (k = read(0, block, sizeof(block))) > 0){

for(l = 0; l < k; l++){

if(block[l] == '\n'){

buf[m] = 0;

m = 0;

lineSplit[j++] = p;

p = buf;

lineSplit[j] = 0;

j = argc - 1;

if(fork() == 0){

exec(argv[1], lineSplit);

}

wait(0);

}else if(block[l] == ' ') {

buf[m++] = 0;

lineSplit[j++] = p;

p = &buf[m];

}else {

buf[m++] = block[l];

}

}

}

exit(0);

}

```

## 3，执行结果

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200925012001954.png#pic\_center)

## 4，测试结果

在xv6中执行：sh <xargstest.sh

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200925012255846.png#pic\_center)

在xv6-labs-2020中执行：./grade-lab-util xargs

![在这里插入图片描述](https://img-blog.csdnimg.cn/20200925012050846.png#pic\_center)