Lab1实验文档

软件13 杨楠

# 操作环境

VMware Workstation Pro: Ubuntu 22.04.2 LTS

# 配置

运行虚拟机，下载Labs.zip，解压，将Labs文件夹放在桌面。在Labs文件夹里运行终端，输入以下命令：

sudo apt-get install git build-essential gdb-multiarch qemu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu

这样就安装完成。

# Boot xv6

解压xv6\_for\_Lab1.zip安装包，在xv\_6\_for\_Lab1目录下，打开终端。

输入以下命令，开始编译运行。

make qemu

首次运行该命令可能会报错，说缺少make程序，提示使用apt安装。那么再输入以下命令：

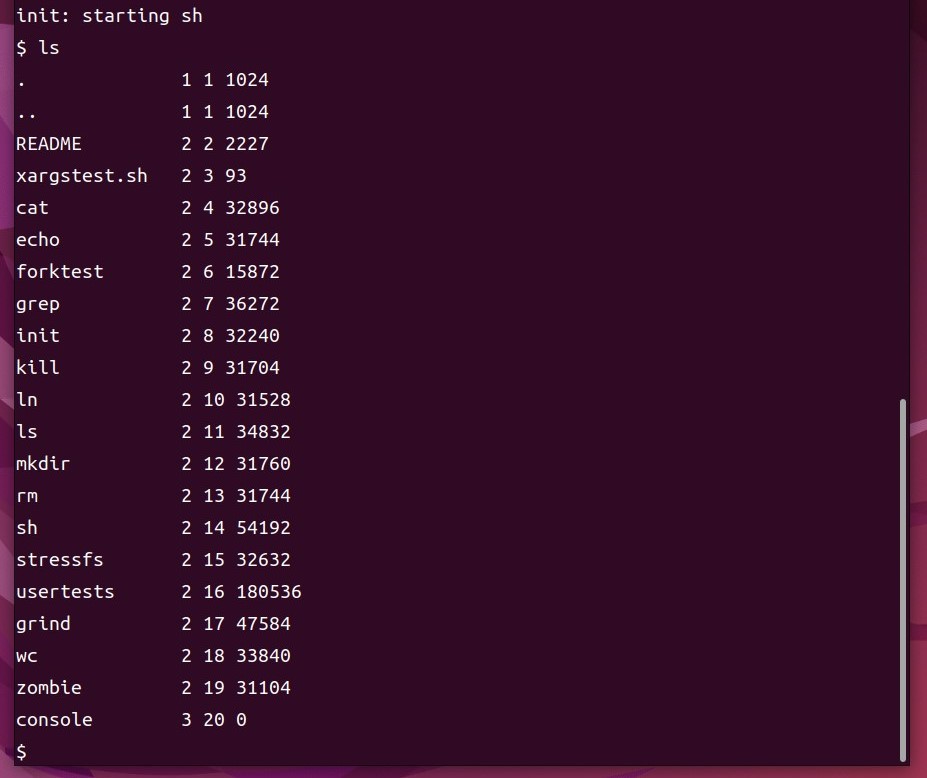
sudo apt install make

安装完成后，再次输入上述make命令，就成功运行。

输入ls命令：

ls

可以看到内容如下。



# sleep

在user文件夹里新建make.c文件，主要的代码如下。

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

{

if(argc < 2)

{

fprintf(2, "usage: sleep time\n");

exit(1);

}

int time = atoi(argv[1]);

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

sleep(time);

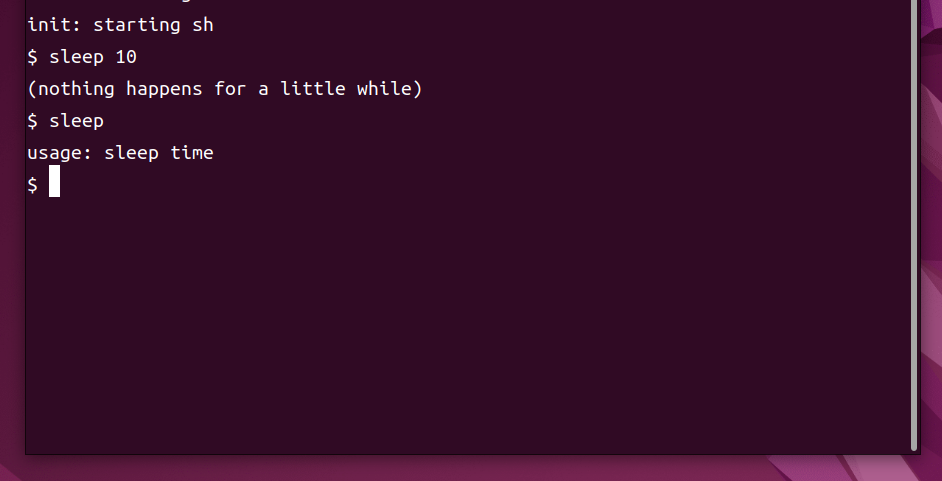
exit(0);

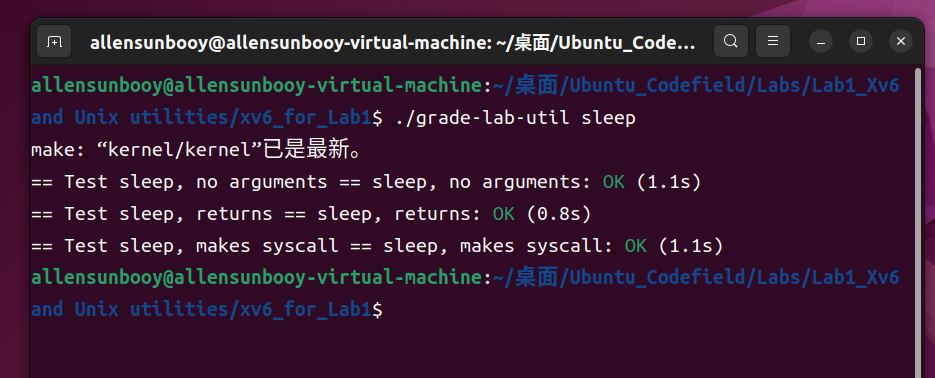
}

判断命令行参数个数，如果参数小于2，那么报错，exit(1)退出。将第二个参数转为int，调用sleep()，然后exit(0)退出。

在Makefile里添加sleep到UPROGS。

在终端进行测试，可以看到，能正确运行sleep命令。





# pingpong

在user文件夹里新建pingpong.c文件，主要的代码如下。

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

{

int p1[2], p2[2];

pipe(p1);

pipe(p2);

char buf;

if(fork() == 0)

{

close(p1[1]);

read(p1[0], &buf, sizeof(buf));

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

close(p2[0]);

write(p2[1], &buf, sizeof(buf));

}

else

{

close(p1[0]);

write(p1[1], &buf, sizeof(buf));

close(p2[1]);

read(p2[0], &buf, sizeof(buf));

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

}

exit(0);

}

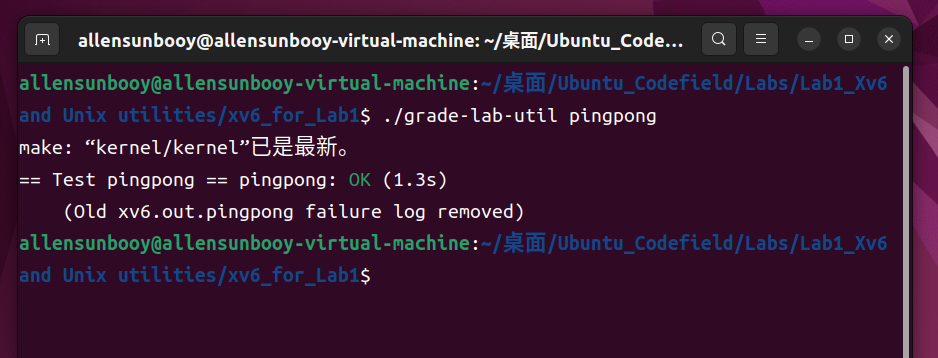
fork==0表示当前为子进程，否则表示当前为父进程。

设置两个管道，父进程将一byte写入管道1，子进程从管道1读取这个byte，输出pid，将这个byte写入管道2。父进程再读取这个byte，输出pid，将这个byte写入管道1。

在Makefile里添加pingpong到UPROGS。

在终端进行测试，可以看到，能正确运行pingpong。





# primes

在user文件夹里新建primes.c文件，主要的代码如下。

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

{

int list[40];

int i;

for(i = 0; i <= 33; i++)

list[i] = i + 2;

int p[2];

int buf;

int num = 34;

while(num > 0)

{

pipe(p);

int head = list[0];

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

if(fork() == 0)

{

close(p[0]);

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

write(p[1], &list[i], sizeof(int));

exit(0);

}

else

{

close(p[1]);

num = 0;

while(read(p[0], &buf, sizeof(buf)))

{

int tmp = buf;

if(tmp % head)

{

list[num] = tmp;

num++;

}

}

}

wait(0);

}

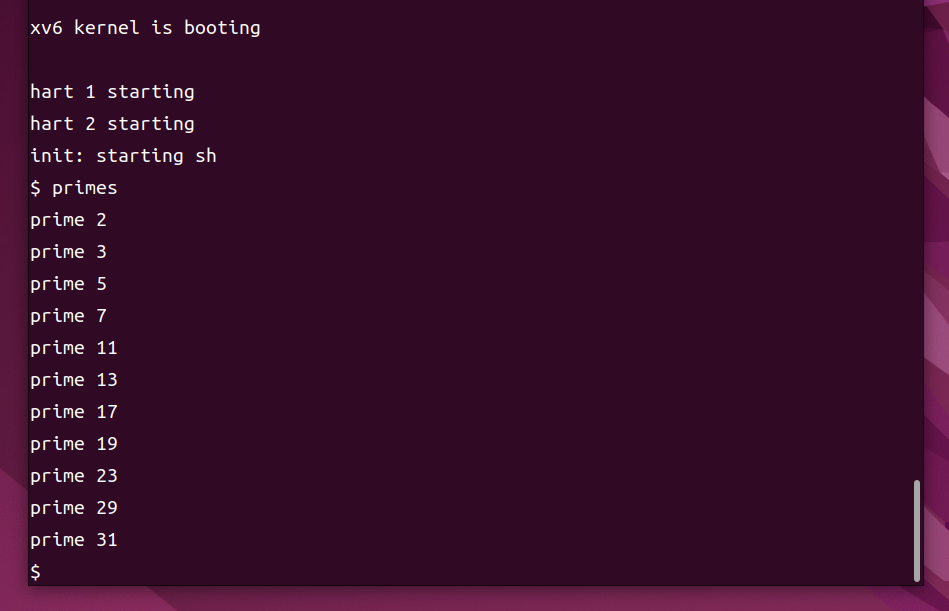
exit(0);

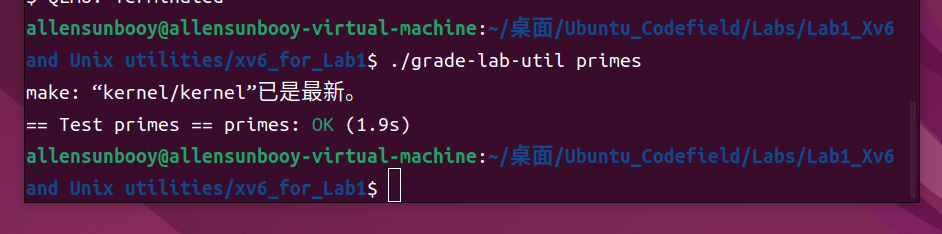
}

数组list初始存储的是2到35这几个整数，更新这个list直至长度为0。每次更新，先获取list[0]并且输出，然后往管道写入这组数，从管道依次读取数字，如果这个数不能整除list[0]，那么将其保留。整个过程类似于埃氏筛法。

在Makefile里添加primes到UPROGS。

在终端进行测试，可以看到，能正确运行primes。





# find

在user文件夹里新建find.c文件，主要的代码如下。

char\* fmtname(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++;

// Return blank-padded name.

if(strlen(p) >= DIRSIZ)

return p;

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

memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));

buf[strlen(p)] = '\0';

return buf;

}

void find(char \*path, char \*name)

{

char buf[512], \*p;

int fd;

struct dirent de;

struct stat st;

if((fd = open(path, 0)) < 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;

}

switch(st.type)

{

case T\_DEVICE:

case T\_FILE:

if(strcmp(fmtname(path), name) == 0)

{

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

}

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))

{

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

continue;

memmove(p, de.name, DIRSIZ);

p[DIRSIZ] = 0;

find(buf, name);

}

break;

}

close(fd);

}

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

{

if(argc == 1)

{

fprintf(2, "Usage: find filepath, filename\n");

exit(1);

}

if(argc == 2)

{

find(".", argv[1]);

exit(0);

}

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

exit(0);

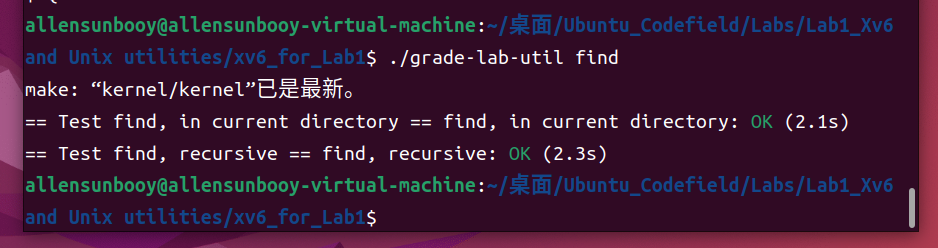
}

参考了ls.c的递归的思路，内容也基本类似，在fmtname()函数添加了buf[strlen(p)] = '\0';手动截断buf。find()函数与ls()函数结构基本类似。

在Makefile里添加find到UPROGS。

在终端进行测试，可以看到，能正确运行find。





# xargs

在user文件夹里新建xargs.c文件，主要的代码如下。

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

{

int i;

char \*args[MAXARG];

char buf[MAXARG \* 32];

char block[MAXARG \* 32];

int num;

for(i = 1, num = 0; i < argc; i++, num++)

args[num] = argv[i];

char \*p = buf;

int len;

int pose = 0;

while((len = read(0, block, sizeof(block))) > 0)

{

for(i = 0; i < len; i++)

{

if(block[i] == '\n')

{

buf[pose] = '\0';

args[num] = p;

num++;

args[num] = 0;

pose = 0;

p = buf;

num = argc - 1;

if(fork() == 0)

exec(argv[1], args);

wait(0);

}

else

{

buf[pose] = block[i];

pose++;

}

}

}

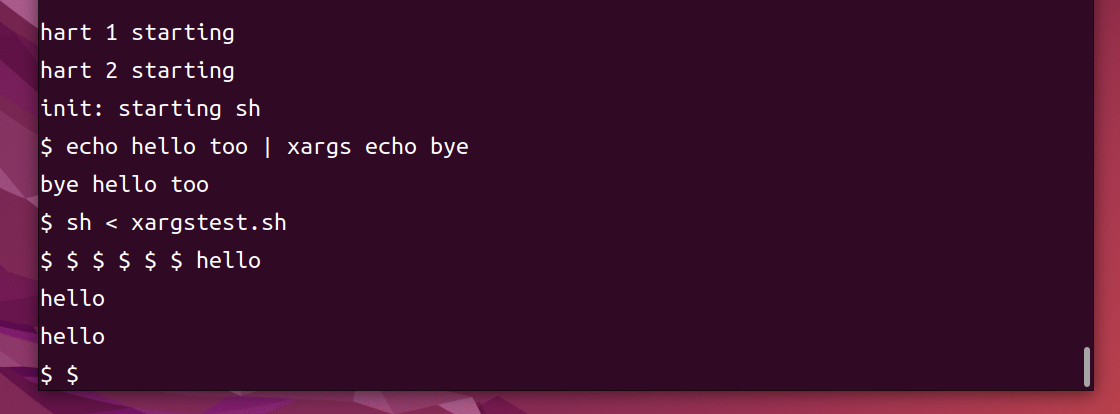
exit(0);

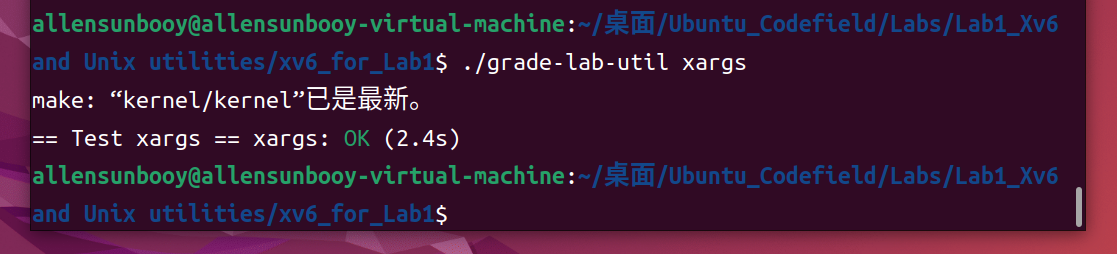
}

为了逐行处理命令，读取的时候，如果遇到\n，则将其截断，原命令行也在此截断，否则就照常读取。

在Makefile里添加xargs到UPROGS。

在终端进行测试，可以看到，能正确运行xargs。





# 备注

提交的xv6\_for\_Lab1文件夹，已经过make clean处理。

