深圳大学
Shenzhen University

操作系统

xv6实验题目解答-1

计算机与软件学院

Lab2实验题目解答

- 题目：仿照echo，写一个命令echo_reversal，实现以下功能：把输入的每个参数中的字符次序颠倒输出。
  - echo_reversal.c

```c
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  int i,j;
  int length;
  char ch;
  for(i = 1; i < argc; i++){
    length=strlen(argv[i]);
    for(j=0;j<length/2;j++){
      ch=argv[i][length-1-j];
      argv[i][length-1-j]=argv[i][j];
      argv[i][j]=ch;
    }
    printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");
  }
  exit();
}
```

Lab2实验题目解答

- Makefile

```
UPROGS=\
	_cat\
	_echo\
	_echo_reversal\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\

fs.img: mkfs README $(UPROGS)
	./mkfs fs.img README $(UPROGS)
```

```
EXTRA=\
	mkfs.c ulib.c user.h cat.c echo.c echo_reversal.c forktest.c grep.c kill.c\
	ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
	printf.c umalloc.c\
	README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
	.gdbinit.tmpl gdbutil\
```

Lab2实验题目解答

- Xv6中并发进程有几种状态，在源码中分别以什么常量代表，试解释每种状态的意义。
  - （proc.h ）
  - UNUSED/未使用的,EMBRYO/萌芽态,
  - SLEEPING/阻塞态,RUNNABLE/就绪态，
  - ZOMBIE/僵死状态

```
enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
```

Lab2实验题目解答

- Xv6中PCB是以什么方式存放的，链表还是数组？系统最多允许同时运行多少个进程？
- 数组形式，最多允许同时运行个进程。
- (proc.c)

```
struct {
  struct spinlock lock;
  struct proc proc[NPROC];
} ptable;
```

- (param.h)

```
#define NPROC        64  // maximum number of processes
```

Lab2实验题目解答

- Xv6是否支持多核cpu? 如果支持的话，是通过哪个数据结构支持的？
- (proc.h)

```
extern struct cpu cpus[NCPU];
```

- (param.h)

```
#define NCPU          8  // maximum number of CPUs
```

Lab2实验题目解答

- 系统启动的第一个进程，其入口函数在哪个文件第几行？它主要实现什么功能？ （提示：阅读《xv6中文文档》第1章“第一个进程”）
- 在 main 初始化了一些设备和子系统后，它通过调用 userinit （1239） 建立了第一个进程。
  - （读userinit函数）
  - 注意，它执行的是initcode.S
  - initcode.S 干的第一件事是触发 exec 系统调用。它用 /init 的二进制代码代替 initcode 的代码，开启shell。
  - （读init.c的代码）

大作业Part I-1（代码理解问题 p9）

- sh.c (:8024-:8026)
  1. if(fork1() == 0)

     为什么fork1()返回值为0时才进入if语句内部？
- 进入fork1函数查看代码，里面调用了fork函数。
  - 文档p8页，fork 函数在父进程、 子进程中都返回（一次调用两次返回）。 对于父进程它返回子进程的 pid， 对于子进程它返回 0。
  - 因此，这里进入了子进程，执行用户输入的命令。

大作业Part I-1（代码理解问题 p9）

1. runcmd(parsecmd(buf)); (sh.c :8025)
   - 阅读runcmd的代码，其中：

(parsecmd -> parseline -> parsepipe)

   - $echo README对应的cmd->type是哪个？
   - EXEC;
   - 相应的，$ls; echo “hello world“ 对应的cmd->type是哪个？
   - LIST;
   - 而ls | wc 对应的cmd->type是哪个？
   - PIPE; (看parsepipe)

大作业Part I-1（代码理解问题 p10）

- 问题：阅读:7930对应的switch分支及相关代码，请说明如何确保rcmd->fd为标准输入的呢？（sh.c/L82）
- 关闭rcmd->fd，而open函数打开的文件就对应着刚才所关闭的文件描述符。
- （看代码，parsecmd -> parseline -> parsepipe -> parseexec -> parseredirs）

```
case REDIR:
  rcmd = (struct redircmd*)cmd;
  close(rcmd->fd);
  if(open(rcmd->file, rcmd->mode) < 0){
    printf(2, "open %s failed\n", rcmd->file);
    exit();
  }
```

大作业Part I-1（代码理解问题 p10）

- 看看parseredirs及redircmd：

```
switch(tok){
case '<':
  cmd = redircmd(cmd, q, eq, O_RDONLY, 0);
  break;
case '>':
  cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
  break;
case '+':  // >>
  cmd = redircmd(cmd, q, eq, O_WRONLY|O_CREATE, 1);
  break;
}
```

大作业Part I-1（代码理解问题 p11）

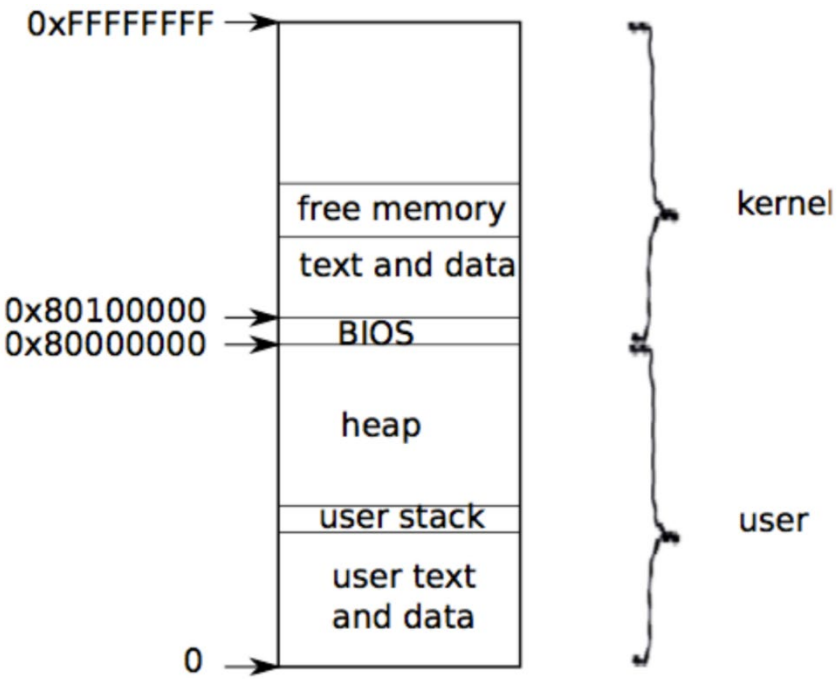
- 第二、三个if语句中，管道的读端口和写端口都通过close语句关闭了，请问还怎么保证pcmd->left的输出进入管道的写端口，而pcmd->right的输入进入管道的读端口？
- 注意：dup 复制一个已有的文件描述符， 返回一个指向同一个输入/输出对象的新描述符。close 会释放一个文件描述符， 使得它未来可以被重用。
- 因为pipe在父进程被调用，在 fork 之后， 父进程和子进程都有了指向管道的文件描述符。
- 如果数据没有准备好， 那么对管道执行的 read 会一直等待， 直到有数据了或者其他绑定在这个管道写端口的描述符都已经关闭了。

大作业Part I-1（代码理解问题 p11）

- 所以在第二个if语句中的子进程，通过dup把标准输出1指向了管道的输出。然后释放掉多余的p[0]和p[1]两个文件描述符。它现在只剩1个标准输出，指向管道输出，另外一个是标准输入。
- 同理，第三个if语句中的子进程，标准输入0指向管道的输入，另有一个标准输出。
- 这样，管道就搭建好了，而且管道的输入不会被同一子进程的管道输出所阻塞。

大作业Part I-2（代码理解问题）

- “xv6 的地址空间结构有一个缺点， 即无法使用超过 2GB 的物理 RAM”——请给出解释。为什么xv6的内存空间只能有2GB?
- 0x80000000=2 x 2^{30}=2GB
  - 2^{10}=1KB; 2^{20}=1MB; 2^{30}=1GB



大作业Part I-2（代码理解问题）

- （p15）“这个映射就限制内核的指令+代码必须在 4mb 以内。”——请给出解释。为什么？

```
pde_t entrypgdir[NPDENTRIES] = {
  // Map VA's [0, 4MB) to PA's [0, 4MB)
  [0] = (0) | PTE_P | PTE_W | PTE_PS,
  // Map VA's [KERNBASE, KERNBASE+4MB) to PA's [0, 4MB)
  [KERNBASE>>PDXSHIFT] = (0) | PTE_P | PTE_W | PTE_PS,
};
```

大作业Part I-2（代码理解问题）

- KERNBASE>>PDXSHIFT=512。entrypgdir实际上是二级页表。NPDENTRIES=1024。这个二级页表有1024项。
- Entrypgdir只有两项，其余为空。

```
// Page directory and page table constants.
#define NPDENTRIES      1024    // # directory entries per page directory
#define NPTENTRIES      1024    // # PTEs per page table
#define PGSIZE          4096    // bytes mapped by a page
```

- 一个二级页表项能记录1024个页号，每个页面有4K。因此1K x 4K=4M。
- 实际上，xv6通过设置PTE_PS标记，利用了x86 CPU的**Page Size Extension (PSE)**特性，直接生成了一个4MB大小的连续页面。

大作业Part I-2（代码理解问题）

- 请问initcode.S 所触发 exec 系统调用执行了哪个程序，而那个程序又是实现什么功能的呢？
- （p18）它会用从文件系统中获取的 /init 的二进制代码代替 initcode 的代码。 现在initcode 已经执行完了， 进程将要运行 /init 。 init （7810行） 会在需要的情况下创建一个新的控制台设备文件， 然后把它作为描述符0， 1，2打开。 接下来它将不断循环， 开启控制台 shell， 处理没有父进程的僵尸进程， 直到 shell 退出， 然后再反复。 系统就这样建立起来了。