实例分析第五部分

郭啸、洪行健、方敏学

pipe 类型的文件操作实现的是什么样的功能？

- 管道是一个小的内核缓冲区，以文件描述符对的形式提供，一个用于读一个用于写，是一种进程间的交互方式。

pipealloc 函数的两个参数 分别代表什么？

```
int pipealloc(struct file **f0, struct file **f1)
{
  struct pipe *pi;

  pi = 0;
  *f0 = *f1 = 0;                                    //初始化两文件描述符
  if((*f0 = filealloc()) == 0 || (*f1 = filealloc()) == 0)   //打开两个文件，如果返回为0，则证明创建不成功
    goto bad;
  if((pi = (struct pipe*)kalloc()) == 0)            //在内核态内存中找到空闲建立管道，返回0则不成功
    goto bad;
  pi->readopen = 1;
  pi->writeopen = 1;
  pi->nwrite = 0;
  pi->nread = 0;
  initlock(&pi->lock, "pipe");
  (*f0)->type = FD_PIPE;
  (*f0)->readable = 1;
  (*f0)->writable = 0;
  (*f0)->pipe = pi;
  (*f1)->type = FD_PIPE;
  (*f1)->readable = 0;
  (*f1)->writable = 1;
  (*f1)->pipe = pi;
  return 0;

 bad:
  if(pi)
    kfree((char*)pi);
  if(*f0)
    fileclose(*f0);
  if(*f1)
    fileclose(*f1);
  return -1;
}
```

- Pipealloc的两个参数是两个二级指针，代表了两个文件描述符地址所存放的地址，或者说指向文件描述符指针的指针。

piperead和pipewrite 函数的流程是什么样的？请介绍出报错（return -1） 的原因，以及核心语句。

```
int pipewrite(struct pipe *pi, uint64 addr, int n)
{
  int i;
  char ch;
  struct proc *pr = myproc();

  acquire(&pi->lock);                                    //获得锁
  for(i = 0; i < n; i++){                                //写n个字节
    while(pi->nwrite == pi->nread + PIPESIZE){           //已经写满
      if(pi->readopen == 0 || myproc()->killed){         //如果读程序没打开，或者进程结束，释放锁返回-1
        release(&pi->lock);
        return -1;
      }
      wakeup(&pi->nread);                                //唤醒读程序
      sleep(&pi->nwrite, &pi->lock);                     //相当于在写满PIPESIZE时睡眠进程
    }
    if(copyin(pr->pagetable, &ch, addr + i, 1) == -1)    //将需要写的内容拷贝，一次一个字节
      break;
    pi->data[pi->nwrite++ % PIPESIZE] = ch;              //写入pipedata
  }
  wakeup(&pi->nread);
  release(&pi->lock);                                     //释放锁
  return n;
}
```

核心语句就是写入data

pi->data [pi->nwrite++%PIPESIZE]=ch

piperead和pipewrite 函数的流程是什么样的？请介绍出报错（return -1） 的原因，以及核心语句。

```c
int piperead(struct pipe *pi, uint64 addr, int n)
{
  int i;
  struct proc *pr = myproc();
  char ch;

  acquire(&pi->lock);                                   //获得锁
  while(pi->nread == pi->nwrite && pi->writeopen){      //管道内为空，并且没有写程序处在打开状态
    if(myproc()->killed){                               //进程退出，则释放锁，返回-1
      release(&pi->lock);
      return -1;
    }
    sleep(&pi->nread, &pi->lock); //DOC: piperead-sleep //睡眠
  }
  for(i = 0; i < n; i++){  //DOC: piperead-copy
    if(pi->nread == pi->nwrite)
      break;
    ch = pi->data[pi->nread++ % PIPESIZE];              //读出，拷贝到指定地址
    if(copyout(pr->pagetable, addr + i, &ch, 1) == -1)
      break;
  }
  wakeup(&pi->nwrite);                                  //唤醒写程序
  release(&pi->lock);                                   //释放锁
  return i;
}
```

pipe 类型的文件读写和 inode 类型的读写有什么区别?

- Pipe是缓冲区读写，无需磁盘操作
- Pipe类型的文件读写可以写任意长度，inode类型的读写只能写固定长度。

pipe 类型操作的实现目的是什么，有什么优点？

- 实现两个进程间的通讯。
- 在xv6的shell中，有可以实现类似pipe效果的临时文件命令，
- 例如echo a | c; echo a >/tmp/m; c < /tmp/m
- 相比之下：

Pipe可以自动清扫，传输任意长度的数据，实现同步。而临时文件需要手动删除，不能同步。

（进阶题）linux 内核代码中 pipe 的实现和 xv6 有哪些不同？

- 带有pipe属性的inode
- 批量读写