lab6挑战性任务报告

1.不带.b后缀指令

在runcmd中，对argv[0]进行特判即可

```
if (strstr(argv[0], ".b") == NULL) {
    strcat(p, ".b");
}
```

2.实现指令条件执行

修改_gettoken，使用2*(*s)来代表||和&&：

```
if (strchr(SYMBOLS, *s)) {
    int t = *s;
    *p1 = s;
    *s++ = 0;
    if (strchr(SYMBOLS, *s)) {
        t+=*s;
        if (t==124) {
            t=126;
        }
        *s++ = 0;
    }
    *p2 = s;
    return t;
}
```

随后修改parsecmd，使得在检测到||或者&&时执行fork，令子进程执行已经解析出来的指令并打上通信标记，父进程等待子进程执行完、传递回返回值后，判断下个指令是否执行。

然后修改libos.c中的libmain，在执行exit前向父进程发出信号，传递执行指令的返回值。在runcmd中执行接受，接受子进程的返回值。然后根据是否需要通信标记决定是否要向父进程传递返回值。

3.更多指令

新增`touch.c,mkdir.c,rm.c`三个文件，具体的实现较为简单，只需要为每个文件编写单独的系统调用即可

4.实现反引号

在识别到奇数反引号时，开启一个管道，新建一个进程，一直执行到下一个反引号即可

5.实现历史指令

在sh.c的main中，新建一个文件/.mosh_histroy，并在每次解析指令后把指令写进去即可。

```
int fd = open("/.mosh_history",O_RDWR);
int n;
read(fd, buf2, (long)sizeof buf2);
write(fd,buf,strlen(buf));
write(fd,"\n",1);
close(fd);
```

针对history指令，在runcmd中特判执行即可。

```
if (strstr(argv[0], "history")!=NULL) {
    argc = 2;
    argv[0] = "cat.b";
    argv[1] = "/.mosh_history";
}
```

6.实现一行多指令

该题比较简单，读取到";"即创建新的进程

```
case ';':
    son = fork();
    if(son==0) {
        return argc;
    }
    else {
        if(*rightpipe == 0){
            dup(1, 0);
        } else if(*rightpipe == 1) {
            dup(0, 1);
        }
        wait(son);
        return parsecmd(argv, rightpipe);
    }
    break;
```

7.实现追加重定向

模仿>，修改文件偏移量即可。

```
case 126://>>
    if (gettoken(0, &t) != 'w') {
        debugf("syntax error: >> not followed by word\n");
        exit();
    }

    fd = open(t, O_WRONLY | O_CREAT);
```

```
        int n;
        while ((n = read(fd, buf, (long)sizeof buf)) > 0);
        struct Fd *f;
        f=(struct Fd *)INDEX2FD(fd);
        struct Filefd *filefd=(struct Filefd *)f;
        f->fd_offset=filefd->f_file.f_size;
        r=dup(fd, 1);
        close(fd);
        if (r < 0) {
            user_panic(">> redirection not implemented");
        }
        break;
```

8.实现引号支持

读取到奇数引号后，把后面的内容直到下一个引号都当作同一个word即可

9.实现前后台任务管理

读取到单个&后，创建子进程，执行已经解析的内容，父进程不再等待子进程。

同时，把管理后台进程的数据放在内核态，用系统调用访问，这样一来，如果想知道后台进程状况，只需要进行系统调用就可以。