Bir kabul yazma zamanı.Senin işin **jsh** yazmak—sahip olacağın kabuk çok ilkel olacak.**Jsh csh** a benzer.

Bu komutları çalıştırmaya izin veren ve giriş çıkış yönlendirmelerini sağlayan bir komut satırı yorumlayıcısıdır. Bourne/c kabuğundaki aynı sözdizimini(syntax) ve operatörleri <,>,>>,& kullanacağız.Bizden tırnak işaretleri,ortam değişkeni uzantıları,desen eşleştirme,iş denetimi ve komut geçmişleri ile başa çıkılmamız beklenmiyor.Üç kısımdan oluşuyor:

BİRİNCİ KISIM

Birinci kısımda jsh1, basit bir uygulaması olacaktır.Jsh1 kendi istemi olan, bir isteğe bağlı komut satırı argümanı alır. Eğer belirtilemezse, istemi "jsh1" olmalıdır. Jsh1 standart giriş çıkışları okur ve eğer boş değilse onları çalıştırmayı dener. Kabuk gibi, jsh1 göreceli yol adları olarak belirtilen yürütülebilir dosyaları bulak için PATH değişkeni aramak gerekir(diğer bir deyişle jsh1 sadece execve yerine execvp ı çağırmaya ihtiyaç duyar). Bir komut satırı eğer & işareti ile bitiyorsa, **jsh1** istemi döndürmeden önce bitirme komutu beklemek zorunda değil. Aksi taktirde bitirmek için komut beklemesi gerekiyor.

Kullanıcı tipi CNTL-D veya exit olduğunda Jsh1 exit yapmalı.

CNTL-C ve **CNTL-Z** hakkında endişe etmeye gerek yok. Ancak, çıkış yaptığımızda, etrafa bak ve boşta gezen processler yok edilir jsh tarafından debugg yapıldığında.

Bir sonraki bölümde dosya yönlendirici olduğundan bunu dert etmeye gerek yok.Sadece fork(), wait() ve execvp() ları doğru yazın.

Belki bakmak isteyeceksiniz forkcat1.c ve forkcat2.c a aynı örneklerde kullanılan frok(), wait() ve execvp().

İkinci kısma geçmeden önce aşağıda ki zombi processini okuyun!

İKİNCİ KISIM

Sıradaki, jsh2. Burada giriş ve çıkışı, yönlendirilen dosyaya ekleyin—i.e. <,> ve >>. Bu aynı zamanda DUP teorisiyle daha kolay yapılabilir. Bu simgelerin <,> ve >> diğer argümanlarla beyaz boşluk ile ayrıldığını tahmin edebilirsiniz.

ÜÇÜNCÜ KISIM

Şimdi uygula pipes jsh3 te. Diğer bir deyişle, <,>,>> ve & ların herhangi bir kombinasyonu çalışmalı. Aynı pipe ile bağlanmış iki farklı processin aynı pointer için **headsort.c** a bak.

Devam etmeden önce,beklerken pipe içindeki tüm processlerin tamamlandığına dikkat edin.

Hatırlamak adına, herhangi bir processed piped numarasına sahip olabilir. Örnek olarak, aşağıdakinin çalışması gerekiyor.(burada f1 rezerve edilip f2 nin içine koyulacak)

```
jsh3: cat -n f1 | sort -nr | sed s/...../// | cat > f2
```

Zombies

Kapanacak zombi proseslerin numaralarını miniminize etmeye çalışacağız. Bu demek değildir ki bir süreliğine var olacaklar, ama sonsuza kadar değil. Wait() i bir kabuk komutu için çağırdığımızda, zombie processin numarasını döndürebilir. Bu iyi—sadece onunla başa çıkmak zorundayız.

```
jsh3: cat f1 > /dev/null &
jsh3: vi lab3.c
```

vi komutunu sonlandırılmasında bekleme için wait() içağıracağız. Ama cat'dan zombie işlemının durumuyla geri dönecektir. Sadece bu şeylerin olabileceğini ve v inin tamamlanması için wait() i

tekrar çağırmamızın farkına varmamız gerekir. Zombie işlemlerin temizlenmesin de non-bloking waits i kullansak bile wait3 ü kullanmalıyız. waidpid kullanılmazına izin verilmez.

Open Files

execvp i çağırdığınızda sadece üç tane dosyanın(0,1,2) açık olduğundan emin olmalıyız.Eğer başka açık dosya varsa ,kabuğumuzda hataya sahip olacagız

Ayrıca komut verildiğinde istemimiz kabukta yazılmalıdır, sonra üç dosya açılmalıdır(0,1,2), diğer taraftan unutulan bir veya iki açık dosya tanımlayıcıları varsa kodumuzda hata olusur.bunu kontrol ediniz. Benım jsh3 asla beşten yuksek bır dosya tanımlayıcısı kullanılamaz.

Waiting in jsh3

Jsh3 de & işareti belirtilmemişse, kabuğumuz devam etmeyecek ta ki tüm processler tamamlanana kadar. Bunun için red-black tree ye ihtiyacımız olacak.

Errors

Kodumuz hatalara karşı çalışabilmeli. Örneğin Örneğin, çok-aşamalı bir borunun sonunda kötü bir çıktı dosyası belirtirseniz, o zaman hata unutulmamalıdır, ve kabuk çalışmaya devam edecektir. Aklınıza gelebilecek tüm hata durumlarını kontrol ettiğinizden emin olun.