Minishell

Bir deniz kabuğu kadar güzel

Özet:
Bu proje basit bir kabuk oluşturmakla ilgilidir.
Evet, kendi küçük Bash'in.
Süreçler ve dosya tanımlayıcıları hakkında kapsamlı bilgi edineceksiniz.

Sürüm: 8.3

İçindekiler

Bölüm I Giriş

Kabuklar, BT'nin en başından beri var olmuştur.

O zamanlar, tüm geliştiriciler bilgisayarla hizalı 1/0 anahtarlar üzerinden iletişim kurmanın son derece sinir bozucu olduğu konusunda hemfikirdi.

İnsan diline yakın bir dilde etkileşimli komut satırları kullanarak bir bilgisayarla iletişim kurmak için yazılım oluşturma fikrini ortaya atmaları çok mantıklıydı.

Minishell ile zamanda geriye yolculuk yapacak ve *Windows* var olmadan önce geliştiricilerin karşılaştığı zorlukları deneyimleyeceksiniz.

Bölüm II

Ortak Talimatlar

- Projeniz C dilinde yazılmış olmalıdır.
- Projeniz Norm'a uygun olarak yazılmış olmalıdır. Eğer bonus dosyalarınız/fonksiyonlarınız varsa bunlar norm kontrolüne dahil edilir ve norm hatası varsa 0 alırsınız.
- Fonksiyonlarınız, tanımlanmamış davranışlar dışında beklenmedik bir şekilde (segmentasyon hatası, veri yolu hatası, dou- ble free, vb.) çıkmamalıdır. Bu gerçekleşirse, projeniz işlevsel olmayan olarak kabul edilecek ve değerlendirme sırasında 0 alacaktır.
- Tüm heap-ayrılmış bellek gerektiğinde uygun şekilde serbest bırakılmalıdır. Bellek sızıntılarına tolerans gösterilmeyecektir.
- Konu gerektiriyorsa, kaynak dosyalarınızı cc kullanarak -Wall, -Wextra ve -Werror bayrakları ile gerekli çıktıya derleyen bir Makefile göndermelisiniz. Ayrıca, Makefile dosyanız gereksiz yeniden bağlantılar gerçekleştirmemelidir.
- Makefile dosyanız en azından $(NAME), all, clean, fclean ve re.
- Projenize bonuslar göndermek için Makefile dosyanıza, projenin ana bölümünde izin verilmeyen tüm çeşitli başlıkları, kütüphaneleri veya işlevleri ekleyecek bir bonus kuralı . Bonuslar, konu aksini belirtmediği sürece _bonus.{c/h} dosyalarına yerleştirilmelidir. Zorunlu ve bonus kısımların değerlendirilmesi ayrı ayrı yapılır.
- Projeniz libft kullanmanıza izin veriyorsa, kaynaklarını ve ilişkili Makefile'ını bir libft klasörüne kopyalamanız gerekir. Projenizin Makefile'ı kütüphaneyi Makefile'ını kullanarak derlemeli, ardından projeyi derlemelidir.
- Bu çalışmanın **teslim edilmesi gerekmese ve not verilmeyecek** olsa da, projeniz için test programları oluşturmanızı teşvik ediyoruz. Bu size kendi çalışmanızı ve meslektaşlarınızın çalışmalarını kolayca test etme fırsatı verecektir. Bu testleri özellikle savunmanız sırasında faydalı bulacaksınız. Nitekim, savunma sırasında kendi testlerinizi ve/veya değerlendirdiğiniz akranınızın testlerini kullanmakta özgürsünüz.
- Çalışmanızı atanan Git deposuna gönderin. Yalnızca Git reposundaki çalışmalara not verilecektir. Çalışmanıza not vermek için Deepthought atanırsa, bu gerçekleşecektir

akran değerlendirmelerinizden sonra. Deepthought'un notlandırması sırasında çalışmanızın herhangi bir bölümünde bir hata meydana gelirse, değerlendirme duracaktır.

Bölüm III

Zorunlu kısım

Program adı	minishell
Dosyaları teslim edin	Makefile, *.h, *.c
Makefile	NAME, all, clean, fclean, re
Argümanlar	
Harici fonksiyonlar.	readline, rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay, add_history, printf, malloc, free, write, access, open, read, close, fork, wait, waitpid, wait3, wait4, signal, sigaction, sigemptyset, sigaddset, kill, exit, getcwd, chdir, stat, lstat, fstat, unlink, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, perror, isatty, ttyname, ttyslot, ioctl, getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs
Libft yetkili	Evet
Açıklama	Bir kabuk yazın

Kabuğunuz:

- Yeni bir komut beklerken bir komut **istemi** görüntüleyin.
- Bir çalışma **geçmişiniz** olsun.
- Doğru yürütülebilir dosyayı arayın ve başlatın (PATH değişkenine göre veya göreli ya da mutlak bir yol kullanarak).
- Alınan bir sinyali belirtmek için en fazla **bir global değişken** kullanın. Sonuçlarını düşünün: bu yaklaşım sinyal işleyicinizin ana veri yapılarınıza erişmemesini sağlar.

Dikkatli olun. Bu global değişken sadece sinyal numarasını saklamalı ve herhangi bir ek bilgi veya veriye erişim sağlamamalıdır. Bu nedenle, global kapsamda "norm" tipi yapıların kullanılması yasaktır.

- Kapalı olmayan tırnak işaretlerini veya \ (ters eğik çizgi) veya ; (noktalı virgül) gibi konu tarafından gerekli görülmeyen özel karakterleri yorumlamaz.
- Kabuğun tırnaklı dizideki meta karakterleri yorumlamasını engelleyecek ' (tek tırnak) işleci.
- Kabuğun $ (dolar işareti) hariç tırnak içindeki meta karakterleri yorumlamasını engelleyecek " (çift tırnak) işlemini gerçekleştirin.
- Aşağıdaki **yönlendirmeleri** uygulayın:
  - < girişi yeniden yönlendirmelidir.
  - > çıktıyı yeniden yönlendirmelidir.
  - << bir sınırlayıcı verilmeli, ardından sınırlayıcıyı içeren bir satır görülene kadar girdiyi okumalıdır. Ancak, geçmişi güncellemek zorunda değildir!
  - >> çıktıyı ekleme modunda yeniden yönlendirmelidir.
- **Boruları (**| karakteri) uygulayın. Boru hattındaki her komutun çıktısı bir boru aracılığıyla bir sonraki komutun girişine bağlanır.
- Değerlerine genişlemesi gereken **ortam değişkenlerini** ($ ve ardından bir dizi karakter) .
- En son çalıştırılan ön plan ardışık düzeninin çıkış durumuna genişlemesi gereken $?
- bash'teki gibi davranması gereken ctrl-C, ctrl-D ve ctrl-\'yi işleyin.
- Etkileşimli modda:
  - ctrl-C yeni bir satırda yeni bir istem görüntüler.
  - ctrl-D kabuktan çıkar.
  - ctrl-\ hiçbir şey yapmaz.
- Kabuğunuz aşağıdaki **yerleşik** komutları uygulamalıdır:
  - echo -n seçeneği ile
  - yalnızca göreli veya mutlak yol ile cd
  - seçeneksiz pwd
  - seçeneksiz dışa aktarma
  - seçenek olmadan ayarlanmamış
  - seçenek veya argüman olmadan env
  - seçeneksiz çıkış

readline() işlevi bellek sızıntılarına neden olabilir, ancak bunları düzeltmeniz gerekmez. Ancak bu, **kendi kodunuzda, evet sizin yazdığınız kodda bellek sızıntısı olabileceği anlamına gelmez**.

Kendinizi konu açıklamasıyla sınırlandırmalısınız. Sorulmayan herhangi bir şey gerekli değildir.
Bir gereksinim hakkında herhangi bir şüpheniz varsa, bash'i referans olarak alın.

Bölüm IV Bonus kısmı

Programınız uygulamalıdır:

- && ve öncelikler için parantezli|| .
- Joker karakterler * geçerli çalışma dizini için çalışmalıdır.

Bonus bölümü yalnızca zorunlu bölüm mükemmel bir şekilde tamamlandığında değerlendirilecektir. Mükemmel, zorunlu bölümün tamamen uygulandığı ve herhangi bir sorun olmadan çalıştığı anlamına gelir. TÜM zorunlu gereklilikleri geçmediyseniz, bonus bölümünüz hiç değerlendirilmeyecektir.

Bölüm V

Sunum ve akran değerlendirmesi

Ödevinizi her zamanki gibi Git deponuza gönderin. Savunma sırasında yalnızca deponuzdaki çalışmalar değerlendirilecektir. Doğru olduklarından emin olmak için dosyalarınızın adlarını iki kez kontrol etmekten çekinmeyin.

M L Q * d z O k v P
:)? e Q 6 u nk] R *
f *q lV - O , ?ZP
7 c 1V H
e 9 () 8
U
> D s / w
; E ' > M
q < >

z b H (3
l X 0g E 6