Öğrencinin;

ADI: Abdulrahman

SOYADI: Alabrash

NO: 1421221032

BÖLÜM: Bilgisayar Mühendisliği

Projenin;

KONUSU: Linux Shell Yapımı

Dersin;

ADI: İşletim Sistemleri

EĞİTMEN: Prof. Dr. A. Yılmaz ÇAMURCU

Ar. Gör. Samet KAYA

Ar.Gör. Okan KARA



İçindekiler

[1- Özet 2](#_Toc466638943)

[2- Proje Konusu 3](#_Toc466638944)

[3- Proje Çıktıları ve Başarı Ölçütleri 3](#_Toc466638945)

[4- Proje Süresince Yapılanlar 3](#_Toc466638946)

[5- Ek Açıklamalar 3](#_Toc466638947)

[6- Kaynakça 3](#_Toc466638948)

# Özet

Komut satırı (veya komut satırı arayüzü), bir bilgisayar kullanıcısının, belli metinleri (komutları) girerek, bilgisayarla iletişime geçmesini sağlar. Komut satırı arayüzleri; konsol, kabuk, terminal veya uçbirim diye de adlandırılır. Komut satırı, kullanıcıya herhangi bir grafiksel arayüz sunmaz bunan yerine kullanıcının metin kipinde bilgisayarla iletişime geçmesini sağlar.

**“exit”** komutu girilene kadar çıkış olamaz

Eğer shellde olmayan bir komut ya da hatalı bir komut girilmişse “**Yanlış bir komut girdiniz**” demelidir.

**“topla"** komutu myshell’den iki parametre alarak çalışmalıdır ve çıktıları yazmalıdır. “alabrashshell>> topla 2 4” çıktı: “2+4=6”. Bu programların çağırılması fork ile yapılması ve programlar bitmediği sürece shell devam etmemelidir.

**“palindrome”** myshellden bir adet parametre almalıdır.

“alabrashshell>> palindrome yatay” çıktı: “yatay palindromedir”. Şeklinde çıktı verir.

Komutlar arasına “&” işareti kullanılarak komutlar sırayla işleyebilmelidir. Örneğin “alabrashshell>>

topla 1 2 & asal 5” iki programda.

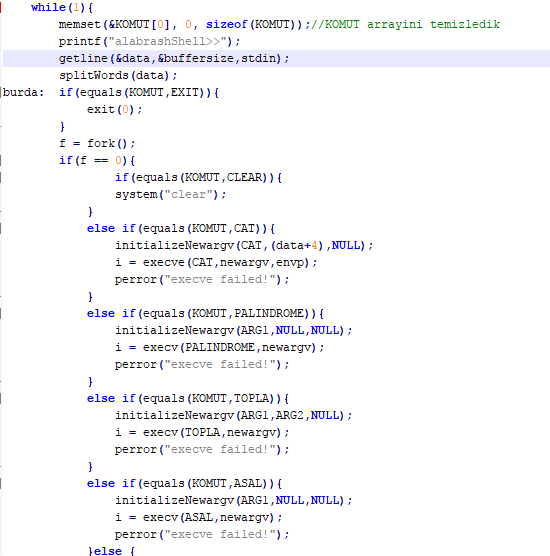
**“cat yazi...”** çıktı: “cat:yazi...”

**“clear”** ekran temizler.

# Proje Konusu

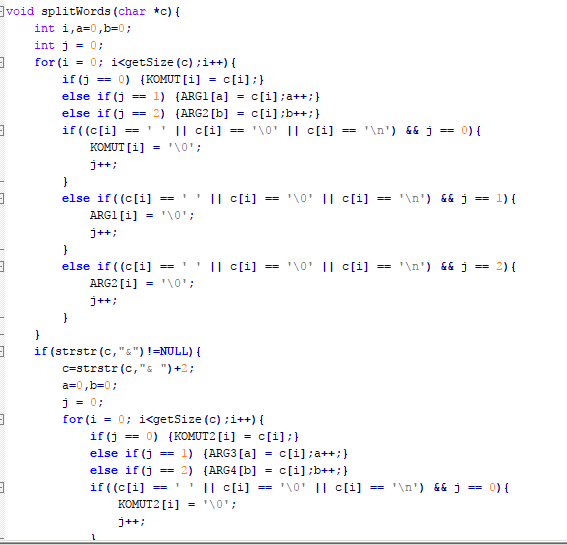
Bu projede kendi linux shell programımızı yazdık. Bu kapsamda bu shell programının kendine özgü komutları, yanıtları, hata çıktıları olmaktadır. Linux işletim sistemindeki shell bir c programıdır yani sizden istenen sistem programlama kapsamında bir C programı yazıldı.

# Proje Çıktıları ve Başarı Ölçütleri

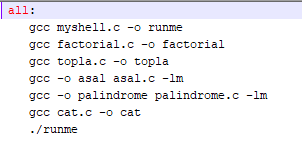


Do-while döngüsü, durum değişkenini denetlemek için daha uygundur çünkü değeri denetlemeden önce bir kez yürütülür. Döngü içinde bir istemi basarız, bir satırı okumak için bir işlev çağırır, satırı args'a bölmek için bir işlev çağırır ve args'ı çalıştırırız.

Örneğin “topla 4 5” komutunun ne anlama geldiğini programa anlatmak için yazılan ilk kelimeyi ayırıp functions.h header ınde bulunan KOMUT isimli bir değişkene atama yapılıyor ve bu değişkene ana programdan erişiliyor. Bu işlem için komut satırına girilen bütünleşik veri boşluklara göre yukarıda anlattığım splitWord fonksiyonu sayesinde gerçekleştiriliyor. 4 ise yine header da bulunan ARG1 değişkenine atanıyor. Ve 5 ARG2 ye atanıyor.



Gelen satır & işareti varsa , String headerinde bulunan strstr(char \*c,char des) substring işlemi yapmakta . substring aldıktan sonra KOMUT2 , ARG3 ve ARG4 argümanlara atma yapılmakta .

Makefile ile başlayalım bir programda bulunan ve programın çalışması için her birisinin derlenmesi gereken dosyaların compile edilme bilgisini içerisinde barındırır. Her birini teker teker derlemek yerine sadece bu dosyayı çalıştırarak hepsini aynı anda derlemek mümkündür. Bu dosyanın içeriği şu şekildedir  


Functions headered bulan methodlar ,

* int equals(char\*,char\*) 🡪 parametre olarak verilen iki string i karşılaştırır int değer döndürür.(1,0).
* int getSize(char\*) 🡪 parametre olarak verilen string in boyutunu döndürür.( Sizeof fonksiyonu bazı hatalar verebiliyor bu yüzden yazdım. )
* int stringToInt(char []) 🡪 parametre olarak verilen string i integere parse eder
* void splitWords (char \*c ) 🡪 parametre olarak verilen string i boşluklara göre ayırır ve reference düzeyinde ilgili değişkenlere atama yapar

myshell sınıfı ise ;

Bu sınıfın main metodunda, daha önce değer attığımız argümanların (KOMUT,KOMUT2,ARG1… ARG4). Daha sonra fork() işlemi harekete geçirir ve dönüş değerini kaydeder. Fork () geri döndüğünde aslında aynı anda çalışan iki süreç var. Alt süreç ilk if koşulunu alacaktır (burada pid == 0).

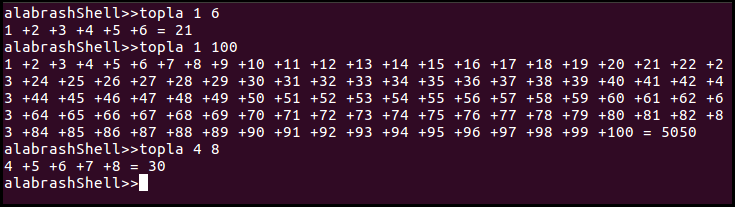
Alt süreçte, kullanıcı tarafından verilen komutu çalıştırmak istiyoruz. Bu nedenle, exec sistem çağrısı execvp'nin birçok türevinden birini kullanıyoruz. Bazıları değişken sayıda dize bağımsız değişkeni alır. Diğerleri dizelerin bir listesini alır. Yine de bazıları, işlemin yürüdüğü ortamı belirlemenize izin verir. Bu özel değişken, bir program adı ve dize argümanlarının bir dizisi (vektör olarak da bilinir, dolayısıyla 'v' olarak adlandırılır) beklemektedir (ilki birincisi program adı olmalıdır). 'p', programın çalıştırılacağı tam dosya yolunu sağlamak yerine adını vereceğiz ve işletim sisteminin yoldaki programı aramasına izin vereceğiz anlamına geliyor.

Exec komutu -1 döndürürse (veya aslında geri dönerse) bir hata olduğunu biliyoruz. Daha sonra kabuğun çalışmaya devam edebilmesi için çıkıyoruz.

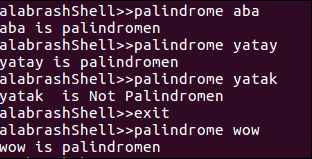
Fork () yaptıktan sonra çocuğun süreci yürüteceğini biliyoruz, bu nedenle ebeveynin komutun çalışmasını bitirmesini beklemesi gerekiyor. Waitpid () işlevi, işlemin durumunun değişmesini beklemek için kullanılır. waitpid () ile sağlanan makroları, süreçler çıkıncaya veya öldürülene kadar beklemek için kullanırız. Ardından işlev, sonuçta tekrar girilmesini isteyecek olan çağrı işlevine bir sinyal olarak 1 döndürür.

Çocuk süreçlere yazdığımız fonksiyonları da ekliyoruz.

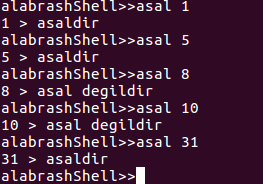
* Topla



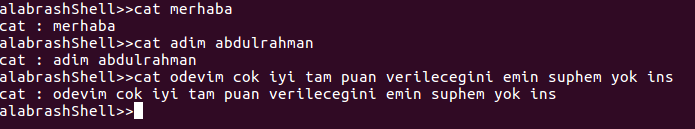
* Palindrome

+

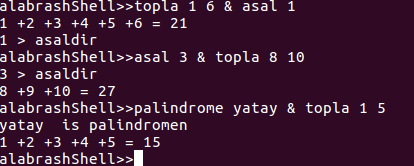
* Asal



* Cat



* & ile



* Yanlış komut girildiğinde



# Proje Süresince Yapılanlar

Öncelikle shell nedir, shell kullanımı hakkında bilgi almak için lab notlarına baktım sonar shell yazımı hakkında internetteki yazıları ve videoları seyrettim. Son olarakta arka planda çalışan kodları birleştirmeye çalıştım. Genel olarak hata aldım ve son olarak <https://brennan.io/2015/01/16/write-a-shell-in-c/> kaynağından yararlanarak tamamlamaya çalıştım.

# Ek Açıklamalar

Functions.h header dosyasının içerisindeki fonksiyonlara ek olarak myshell.c dosyasında ayrı bir fonksiyon bulunmaktadır ve bu kısaca her bir komut için gerekli argümanları newargv string array ine atar. (void initializeNewargv(char\*,char\*,char\*))

Yine programın akışı için gerekli, shell yapısı gereği sonsuz bir döngü kullanılması gerekiyor. Bu yapıyı sonsuz bir while döngü ile sağladım.

# Kaynakça

* <http://ceyhuntopcuoglu.com/terminal-ucbirim-konsol-komut-satiri-nedir.html>
* https://brennan.io/2015/01/16/write-a-shell-in-c/
* https://www.cs.cornell.edu/Courses/cs414/2004su/homework/shell/shell.html