Galatasaray Üniversitesi Bilgisayar Mühendisliği Bölümü
INF333 İşletim Sistemleri: **Ödev 1**

1 `fork()` (20 Puan)

Bu soruda aşağıdaki isterleri yerine getiren bir C programı yazmanız istenmektedir:

- Ebeveyn süreç bir çocuk süreç yaratsın ve çocuk sürecin sonlanmasını `wait()` veya `waitpid()` sistem çağrısıyla beklesin,
- Çocuk süreç:
  - `srand()` ve `rand()` fonksiyonlarının yardımıyla **0-9** arasında üreteceği rasgele bir sayıyı ekrana yazdırsın:
    Örnek çıktı: "`Çocuk süreç 5 sayısını üretti ve 5 saniye uyuyacak`"
  - `sleep()` fonksiyonunu kullanarak ürettiği sayının değeri kadar saniye uykuya dalsın,
  - Uykudan uyandıktan sonra, ürettiği sayı dönüş değeri olacak şekilde kendini sonlandırsın (`exit()`).
- Çocuk süreç sonlandığında debloke olacak ebeveyn süreç, `wait()`/`waitpid()` çağrısından edindiği `status` değişkenindeki bilgiyi, TP'de gördüğümüz `WIFEXITED()` ve `WIFSIGNALED()` makrolarıyla işleyerek ekrana çocuk sürecin `PID`'sini ve hangi değerle sonlandığını yazdırsın.
  Örnek çıktı: "`Çocuk süreç (PID: 23943) 5 dönüş değeriyle sonlandı.`"

2 İlkel Kabuk

Bu soruda ilk TP'de gördüğümüz kavramları kullanarak çok basit bir kabuk programı yazmanız istenmektedir.

2.1 Temel döngü (40 Puan)

Ebeveyn süreç sonsuz bir `while` döngüsü içerisinde ekrana "`>`" karakterini basarak kullanıcıdan komut beklesin:

- Kullanıcının girdisini `fgets()` ile yeteri kadar büyüklükte (örneğin `char line[2000]`) bir karakter dizisine okuyun,
- `fgets()` fonksiyonu, satır sonu karakteri olan `\n` baytını da okumaktadır. `line[]` içerisindeki bu karakteri `\0` ile ezerek silin. Eğer kullanıcı hiçbir metin yazmadan sadece enter'a bastıysa bunu anlayıp döngünün başına dönün,
- Bu adıma geldiyseniz elinizde boş olmayan bir metin var demektir:

```
> ls (Enter)
Bu aşamada line[] içerisinde "ls" yazmaktadır.  Ebeveyn artık switch/fork/exec
mantığını işletip çocuk sürecin ls programını çalıştırmasını sağlamalıdır.
```

  - Ebeveyn süreç `wait()/waitpid()` sistem çağrılarından birini kullanarak çocuk süreci beklemeli,
  - Çocuk süreç `line[]` içerisinde yazan programı `execlp()/execvp()` sistem çağrılarından biriyle çalıştırmalı,
  - Çocuk sürecin çalıştırdığı program sonlandığında, ilk sorudaki gibi debloke olacak ebeveyn süreç `WIFEXITED()` ve `WIFSIGNALED()` makrolarıyla ekrana çocuk düzgün sonlandıysa dönüş değerini, sinyalle sonlandıysa sinyalin numarasını yazmalıdır.

Bu kısmı doğru yapıp yapmadığınızı aşağıdaki gibi bir etkileşim senaryosuyla test edebilirsiniz:

```
$ gcc kabuk.c -o kabuk
$ ./kabuk
> (Kabuğunuz sizden komut bekliyor.  Enter'a basın)
> (Hiçbir şey yazmadan Enter'a basınca kabuk sürekli yeni bir > karakteri basmalı)
> (Enter)
> (Enter)
> date
Cum Mar 25 11:08:06 EET 2016
Child terminated with:  0
(date komutu çalıştı ve başarıyla sonlandı.  Dönüş değeri 0)
> shutdown
Must be root.
Child terminated with:  1
(shutdown programı root yetkisi istediğinden program 1 değeriyle sonlandı)
```

2.2 execvp() dönüş değeri (20 Puan)

TP'de de bahsedildiği gibi, başarılı olan bir exec* ailesi çağrısı, süreci tamamen ele geçirip yeni programı çalıştıracaktır. Bu durumda yazdığınız kod **asla** exec* ailesi çağrısının bir alt satırına geçmeyecektir çünkü süreç artık bambaşka bir kimliğe bürünmüş, çalıştırılacak programa dönüşmüştür.

exec* ailesi çağrılarının başarısızlıkla geri dönmesine yol açan yaklaşık 20 adet sebep vardır. Bunların tamamına man 2 execve komutuyla man sayfasından erişebilirsiniz.

Bilgi notu: Linux sistem seviyesinde hata takibi
Sistem çağrıları ve bazı standart kitaplık fonksiyonları başarısızlıkla sonlandıklarında, errno adlı global bir C değişkenine hata numarasını yazarlar. Bu değişkene erişmek için programınızda errno.h başlık dosyasını include etmeniz gerekmektedir. Dikkat edilmesi gereken nokta, bu değişkenin **daima** son hataya dair bilgi saklamasıdır.

Hata numaraları okunabilir veya anlaşılabilir olmadığı için sistem seviyesinde bu sayılara denk gelen ve **E** harfiyle başlayan sembolik hata isimleri mevcuttur. Bu hataların isimlerine ve açıklamalarına "man 3 errno" komutuyla erişebilirsiniz.

Ödevin bu bölümünde ilgileneceğimiz durum, çalıştırılmak istenen programın sistemde bulunamadığı durumdur. Bu durumda kullandığımız bash kabuğu nasıl davranmaktadır önce onu inceleyelim:

```
ozan@kivanc:~/tmp/inf333 $ olmayankomut
olmayankomut:command not found
```

- Yukarıdaki davranışı aynen tekrarlamak için exec* ailesi çağrısından sonra errno değişkeninin değerinin ENOENT olup olmadığını kontrol etmeniz ve buna göre ekrana yukarıdaki mesajı yazdırmanız gerekmektedir. ENOENT hatası verilen program bulunamadığı zaman oluşmaktadır.
- Linux kabuklarında programı çalıştırmakla yükümlü alt sürecin exec* ailesi çağrısı komutu bulamadığı için başarısız olursa, 127 dönüş değeriyle sonlanır. Bunu da bash ile gözlemleyebiliriz:

  ```
  ozan@kivanc:~/tmp/inf333 $ olmayankomut
  olmayankomut:command not found
  ozan@kivanc:~/tmp/inf333 $ echo $?  (Son komutun dönüş değeri ?  adlı bash değişkeninde
  tutulur)
  127
  ```

  Yine siz de yazdığınız kabuğun bu durumda 127 ile sonlanmasını sağlayın.

2.3 Gömülü `exit` komutu (10 Puan)

Linux kabuklarında bazı komutların kabuğun koduna gömülü olduklarını söylemiştik. Genellikle `cd, exit` gibi çok temel komutlar gömülü olarak tasarlanırlar.

Bu komutları çalıştırmak için `exec*` ailesi çağrılarına ihtiyaç yoktur. Hatta ve hatta bir çocuk süreç yaratılmasına da gerek yoktur. Ebeveyn süreç desteklediği gömülü komutlardan birinin adını tespit ettiği anda o komutun işlevini gerçekleştiren fonksiyonu çalıştırır ve tekrar > işareti basan döngünün başına döner.

Ödevin bu kısmında `exit` gömülü komutunu tasarlamanız istenmektedir. Eğer komut satırından girilen komut `exit` ise (basit bir string karşılaştırma işlemi), ebeveyn süreç `exit(EXIT_SUCCESS)` çağrısıyla sonlanmalıdır. Bu gömülü komut eksikken ilkel kabuğumuzu düzgünce sonlandırmanın bir yolu yok oysa `bash` kabuğunda `exit` yazdığımızda kabuk düzgünce sonlanabilmektedir:

```
$ bash (yeni bir kabuk açalım)
$ (yeni kabuktayız)
$ exit
$ (eski kabuğa döndük)
```

Bu kısmı yaptıktan sonra kabuğunuz aşağıdaki gibi davranmalıdır:

```
$ ./kabuk
> exit
Bye!
$ (bash'e döndük)
```

2.4 Çoklu komut (10 Puan)

Ödevin 40 puanlık temel döngü kısmını yaptıysanız, ekstra argüman alan komutların çalıştırılamadığını farketmiş olabilirsiniz. Bunun sebebi `fgets()`'in okuduğu satırı boşluklardan ayırmadan doğrudan `exec*` ailesine vermemiz. Oysa normalde kullanıcının girdiği satır boşluklardan ayrılmalı ve elde kalan *string*'ler `exec*` ailesine argüman olarak geçirilmelidir. Bu anlatılanı yapan `run_program(char *str)` fonksiyonunu yazın. Bu fonksiyon *str* içerisinde ne olursa olsun düzgünce boşluklardan bölüp ilgili programı çalıştırmayı başarsın:

```
$ ./kabuk
> ls
a.out kabuk kabuk.c
> ls /tmp (program adıyla beraber 2 argüman oldu)
evince-34039 skype-2304 tmp_file
```

Bu fonksiyonda argümanları dizi içerisinde alan `execvp()`'yi kullanmak işinizi kolaylaştıracaktır. Karakter dizisini bölerken `strtok()` standart fonksiyonundan faydalanabilir veya kendi bölme algoritmanızı kodlayabilirsiniz.