BELLEK

Size anlattığım bu makine düzeni hydra labaratuvarındaki makinelerdir.Bu derste anlatılan programları farklı makineler üzerinde çalıştırırsanız (kenner, hydra makineleri) farklı sonuçlar almanız muhtemeldir. Bununla beraber, nasıl bir makine düzeni üzerinde çalışıyorsanız çalışın bunu çözebilmelisiniz. (*However, you should be able to figure out how whatever the machine you are on is laid out.)*

Kabuğu yüklemelisiniz, böylece bu dersi uygularken çekirdek dosyalarını oluşturmazsınız. Örneğin, bu iş eğer sizin .cshrc dosyanızda yapılmamışsa, şunu yazın:

UNIX> limit coredumpsize 0

Bu ders Unix'te hafızaya giriş dersidir.

Daha öncede söylenildiği gibi hafıza büyük bir dizi (0xffffffff diyelim) gibidir. C deki bir işaretçi bu diziyi indeksler.C işaretçisi 0xefffe034 değerini gösterdiğinde hafıza dizisinde 0xefffe035 numaralı hücre gösterilir.(hafıza 0 dan başlayarak indexlenir.)

Maalesef belleğin her yerine erişemezsiniz. Bu durum için çokça gördüğümüz bir örnek 0. adrestir. Eğer bir işaretçiyi dereferans edip 0 değerine atamaya çalışırsanız segmentasyon ihlali hatası alacaksınız. Bu, Unix'in o konum için hafıza kullanımının illegal olduğunu söyleme şeklidir.

Mesela, aşağıdaki kod segmentasyon ihlali oluşturur:

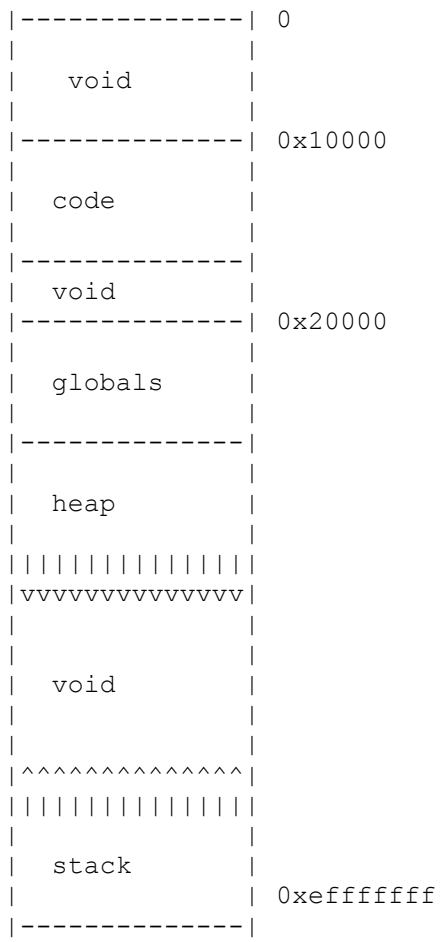
```
main()
{
char *s;
char c;

s = (char *) 0;
c = *s;
```

Anlaşıldı ki bellek 4 legal bölgeden oluşuyor.Bunlar;

1.Kod: Program komutlarının bulunduğu bellek bölgesidir.
2.Küreseller: Bu bölgede küresel değiskenler bulunur. (init data and bss)
3.Yığıt: **malloc()** fonksiyonu ile aldığınız bellek bölgesidir.
4.Yığın: Stack ise yerel değişkenleri ve fonksiyon parametrelerini içerir.

Eğer belleği büyük bir dizi olarak görüntülersek, bölgeler(veya segmentler)aşağıdaki gibi olur:



Şunu not edin, **malloc()** çağrısı yaptıkça yığıt aşağı doğru, yığın ise iç içe geçmiş yordam çağrıları yapıldıkça yukarı doğru gider.

SAYFALAMA

Ne tür sistemler kullandığınıza bağlı olarak, işletim sistemleri her prosese 0x0'dan başlayan ve 0xffffffff veya 0x8fffffff'a kadar giden bellek adres alanları tahsis eder. Bunların hepsi sanal bellek adreslerdir. Aklınızda kalmasına yardımcı olması için telefon numaralarının evinize atanmasını söyleyebiliriz. Evinizin bulunduğu caddenin adresinin değiştirilememesine rağmen telefon numaraları mantıksaldır ve kolayca değiştirilebilir. İşletim sistemi bu sanal adres alanını fiziksel adres alanına

göstermek durumundadır. Bu işletim sistemi tarafından bellek yönetimi dahilinde yapılan işlerin bir kısmıdır. Bellek yönetimi sınırlı fiziksel adres alanını çok sayıda proseslerin ihtiyaçları dahilinde en iyi şekilde kullanımını sağlar. Bellek yönetimini gerçekleştiren fazlaca çözüm var, iyi ki UNIX sistemleri sayfalama denilen standartı kullanır.

Bellek hydra makinelerinde 8192 baytlık bölümlere ayrılmıştır. Bunlara sayfalar denir. Bazı makinalarda sayfaların boyutu 4096 battır. Bu donanım tarafından belirlenen birşeydir.Çoğunlukla aynı büyüklüktetir.

Bellek şu şekilde çalışır: İşletim sistemi belleğin belirli sayfalarını sizin için ayırır. Ne zaman ayrılan adresten okuma ya da yazma yapmak istendiğinde işletim sistemi donanımdan gelen talimatla o sayfanın sizin için ayrılıp ayrılmadığını kontrol eder. Eğer sizin için ayrılmış ise o zaman okuma veya yazma için işletim sistemi izin verir. Aksi takdirde segment ihlali (segmentation violation) hatası alırsınız. (Segment ihlalinin birden çok sebebi olabilir, bahsettiğim sadece bunlardan biridir.)

Olan şey şudur:

```
s=(char *) 0;
c = *s;
```

"c= *s" dediğinizde, donanım sizin belleğin 0 konumundaki adresini okumak istediğinizi anlar. İşletim sistemi yardımıyla kontrol eder ve size "Ben 0 konumlu adresi sizin için ayırmadım" der. Bunun sonucu segment ihlalidir. Bu sayfa ayrılabilir, fakat fiziksel bellekte mümkün değildir. Bu sayfa hatası (page fault) kavramıyla alakalıdır.

Sayfa hatası, bir prosesin sanal bellek adresinde bir alana ulaşmaya çalışırken, bu sayfanın fiziksel bellekte olmadığı zaman işletim sistemi tarafından oluşur. Bunun sonucu olarak, İşletim sistemi prosesi taleb edilen sayfa okunana kadar durdurur. Sayfa hatası, çoğu zaman bir hata değildir. Buna karşılık, segment hatası neredeyse her zaman hata verir.

Sayfalamanın asıl mantığı işletim sistemleri dersinde verilir. Daha fazla bundan bahsetmeyeceğim.

Hydra makinamızdaki ilk 8 sayfanın boş olduğu ortaya çıkıyor. Bu da 0'dan 0xffff adresine yazmaya ya da okumaya çalışmak segment ihlali hatası verir.

Sonraki sayfada (0x10000 adresinden başlayan) kod segmenti başlar. Bahsettiğim bu segment &etext değeriyle biter. Küresel segment 0xeffffffff adresi ile biter. Başlangıç adresi sizin yaptığınız farklı prosedür çağrıları(procedure calls) ile değişir. İleride bu konuyu biraz daha açacağız. Yığıtın(heap) sonundaki ve yığının(stack) başındaki her sayfa boştur ve erişilme sırasında segment ihlali verir.

&etext, &edata, &end.

Bu değişkenler hakkında daha fazla bilgi için**man etext** yada **man edata** yazın. Bu globaller unix 'e özeldir.
Bu üç harici değişken aşağıda gösterildiği gibi tanımlıdır:

```
extern etext;
extern edata;
extern end;
```

Bunlar görüldüğü gibi tipsiz değişkenlerdir. Genellikle "etext" ve "end" olarak kullanmazsınız çünkü bu değişkenler kodunuzun içerisinde bir kopyası tanımlanmadan harici olarak link edildiği sürece bu üç değişken (.etext, .edata, .end gibi) Id tarafından rezervelidir. Bunun yerine onların adreslerini kullanın.
Bu noktalar sırasıyla;
text sonu, globallerin initialize edilmiş data segmenti sonunda ve global initialize edilmemiş datası sonunda

Şimdi testaddr1.c programına bakın.
etext, edata ve end adreslerinin dışı yazdırılmıştır. 6 dış değer yazdırılmıştır.

- **main**, main() prosedürünün ilk komutu için bir göstericidir. Bu sadece kod segmenti içinde bir konumdur. Assembler derslerinden bunu hatırlamanız gerekir.
- **I** bir global değişkendir. Bu nedenle **&I** globals segmentinden bir adress olmalıdır.
- **i** bir lokal değişkendir. Bu nedenle **&i** yığında bir adres olmalıdır.
- **argc**, main() için bir argümandır. Bu nedenle **&argc** yığında bir adres olmalıdır.
- **ii** bir diğer lokal değişkendir. Bu nedenle **&ii** yığında bir adres olmalıdır. Ancak **ii** **malloc** ile ayrılmış, bellek için bir göstericidir. Bu nedenle **ii** heap bölgesinde bir adres olmalıdır.

testaddr1 'i çalıştırdığımızda aşağıdakine benzer bir çıktı alırız.

```
UNIX> testaddr1
&etext = 0x108b8
&edata = 0x20a34
&end   = 0x20a54

main   = 0x10688
&I     = 0x20a4c
&i     = 0xffbef82c
&argc  = 0xffbef884
&ii    = 0xffbef828
ii     = 0x20a68
UNIX>
```

Peki burada kod segmentinin 0x10000 adresinden 0x108b8 adresine gitmesi ne anlama geliyor?
Global segmenti 0x20000 adresinden 0x20a54 adresine gidiyor.
I initialize edilmemiş, bu nedenle **bbs**(block started by symbols) 'de görünür.
Heap 0x20a54 adresinden 0x20a68'den büyük bazı adreslere gider. (**ii** 0x20a68 den itibaren 4 bayt olarak ayrılmıştır)
Yığın, 0xffbef82 adresinden daha küçük adreslerden 0xffffffff adresine kadar gider. testaddr1 tarafından yazılan tüm adresler mantıklıdır.

Şimdi testaddr2.c dosyasına bakalım.
Yani, sıfır yasadışı bir bellek değeri ile bunu deneyelim:

```
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x0
Reading 0x0:  Segmentation Fault
UNIX>
```

Sıfır bellek adresinden okumaya çalıştığımızda, segmentation fault hatası alırız. Çünkü sıfır bellek adresi geçersiz. Donanım bunu işletim sistemine sorar ve parçalama hatası oluşturur. 0x0 dan 0xffff 'e bellek adresleri geçersiz. Eğer bu aralıkta bir adresi denersek parçalama hatası alırız.

```
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0xffff
Reading 0xffff:  Segmentation Fault
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x4abc
Reading 0x4abc:  Segmentation Fault
UNIX>
```

0x10000 adresi kod segmenti içerisinde. Bu legal bir adres olmalıdır.

```
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x10000
Reading 0x10000:  127
Writing 127 back to  0x10000:  Segmentation Fault
UNIX>
```

0x10000 adresinden okumaya başladığımıza dikkat edin. Bize 127 bayt verdi. Ancak 0x10000 yazdığımızda parçalama hatası alırız. Bu tasarımdandır. Kod segmenti yalnız okunabilir. Okuyabilirsizi fakat yazamazsınız. Bu mantıklı olandır çünkü programınız çalışırken onu değiştiremezsiniz. Bunun yerine tekrar derlemeyi denemelisiniz.
Şimdi eğer 0x11fff adresini denersek neolur? Bu &etext 'in yukarısında, kod segmentinin dışında olmalı.

```
UNIX> testaddr2
&etext = 0x1191b
```

```
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x11fff
Reading 0x11fff:  -48
Writing -48 back to  0x11fff:  Segmentation Fault
UNIX>
```

0x11fff adresi kod segmenti dışında bir adres olmasına rağmen okuyabildik.
Şimdi sayfa 9 dan 15 e tekrar okunamaz.

```
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x12000
Reading 0x12000:  Segmentation Fault
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x1f000
Reading 0x1f000:  Segmentation Fault
UNIX>
```

Globaller 0x20000 adresinden başlamakta ve böylece 16. sayfanın okunabilir ve yazılabilir olduğunu gördük.

```
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x20000
Reading 0x20000:  127
Writing 127 back to  0x20000:  ok
UNIX>
```

Bu sayfa içerisinde bir konuma okuyabilir ve yazabiliriz (0x20000 - 0x21fff). Sonraki sayfaya (0x22000 den başlıyor) erişilemiyor.

```
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x21dff
Reading 0x21dff:  0
Writing 0 back to  0x21dff:  ok
UNIX> testaddr2
&etext = 0x1191b
&end   = 0x21d90

Enter memory location in hex (start with 0x): 0x22000
Reading 0x22000:  Segmentation Fault
UNIX>
```

Sbrk(0)

sbrk () bir kaç derste kullanacağımız bir sistem çağrısıdır. Bu fonksiyon heap in sonundaki güncel kullanıcıyı geri döndürür. Malloc() fonksiyonunu çağırdığımız andan itibaren sbrk (0) zamanla değişebilir. testaddr3.c sbrk (0) nın aldığı değerleri gösterir. --Not: fonksiyon sayfa 16'dadır (0x02000-0x21fff)-- Donanımların 8192 byte aralığında kontrolleri gerçekleştirmelerinden itibaren, sbrk (0) fonksiyonu 0x20c78 değerini gönderse bile herhangi bir byte'ı sayfa 16'dan çağırabiliriz :

UNIX> testaddr3

&etext = 0x11993

&end = 0x21e18

sbrk(0)= 0x21e18

&c = 0xffbee103

Hafıza bölgesine erişim 16 lık sistemde(0x ile başlayarak) ifade edilir: 0x21fff

0x21fff adresini okuyor: 0

0 yazmak programı 0x21fff adresine yönlendirecektir: tamam

UNIX>

& end ve sbrk (0) fonksiyonlarının aynı değeri göndermesinin sebebi malloc () fonksiyonunu testaddr3.c içerisinde çağıramamamızdır. testaddr3a.c içindeyken malloc() fonksıyonu programın başlangıcında çağırılabilir. Böylece &end ve sbrk(0) farklı değerler döndürecektir:

UNIX> testaddr3a

&etext = 0x119a3

&end = 0x21e28

sbrk(0)= 0x23e28

&c = 0xffbee103

Hafıza bölgesine erişim 16 lık sistemde(0x ile başlayarak) ifade edilir: 0x23fff

0x23fff adresini okuyor: 0

0 yazmak programı 0x23fff adresine yönlendirecektir: tamam

UNIX> testaddr3a

&etext = 0x119a3

&end = 0x21e28

sbrk(0)= 0x23e28

&c = 0xffbee103

Hafıza bölgesine erişim 16 lık sistemde(0x ile başlayarak) ifade edilir: 0x24000

0x24000 adresini okuyor: Kesme Hatası (Segmentation Fault)

UNIX>

YIĞIN(STACK)

Yığının başlangıcı nerede?Eğer yukardaki 0xffbee103 adresi testaddr3 içinde denersek çalışı,Onların çoğunun uygun olduğunu görürüz:

UNIX> **testaddr3**
&etext = 0x11993
&end = 0x21e18
sbrk(0)= 0x21e18
&c = 0xffbee103

hafıza konumunu girin hex(0x ile başlar): **0xffb00000**

Okuma 0xffb00000: 0
Yazma 0 geri için 0xffb00000: ok
UNIX> **testaddr3**
&etext = 0x11993
&end = 0x21e18
sbrk(0)= 0x21e18
&c = 0xffbee103

hafıza konumunu girin hex(0x ile başlar): **0xff3f0000**
Okuma 0xff3f0000: 0
Yazma 0 geri için 0xff3f0000: ok
UNIX> **testaddr3**
&etext = 0x11993
&end = 0x21e18
sbrk(0)= 0x21e18
&c = 0xffbee103

hafıza konumunu girin hex(0x ile başlar): **0xff3effff**
Okuma 0xff3effff: Segmentasyon Hata

UNIX>

Sonuç ne?Sonuç şöyle çıkar:İşletim Sistemi 0xff3f0000 dan gelen tüm sayfaları yığının altında ayırır.Yığının altı nerede?Derine inelim:

UNIX> **testaddr3**
&etext = 0x11993
&end = 0x21e18
sbrk(0)= 0x21e18
&c = 0xffbee103

hafıza konumunu girin hex(0x ile başlar): **0xffbeffff**
Okuma 0xffbeffff: 0
Yazma 0 geri için 0xffbeffff: ok
UNIX> **testaddr3**
&etext = 0x11993
&end = 0x21e18
sbrk(0)= 0x21e18
&c = 0xffbee103

hafıza konumunu girin hex(0x ile başlar):**0xffbf0000**
Okuma 0xffbf0000: Segmentasyon Hata
UNIX>

Böylece yığın 0xff3f0000 dan gelir 0xffbeffff a gider.Kabaca 8 Megabayt dir.
Varsayılan yığın boyutunu yazdırmak ve limit komutunu kullanarak yığın boyutunu değiştirebilirsiniz(anasayfa sayfasını okuyun).

UNIX> **limit**
...
stacksize 8192 kbytes
...

Bir yordam çağırıldığında,yığın yordamdaki yerel değişkenlere ve argümanlara(ek olarak birkaç başka şeyler) ayırır. Bir yordama dönmek istediğimiz zaman,bu değişkenler yığından atılır.Bu yüzden , testaddr4.c göz atın.Main() de yinelemeli argümanlar olduğu için birçok kez kendisini çağırır.Her özyenileme çağrıldığında görürsünüz,argc ve argv adresleri ve yerel değişkenler i küçük adreslerdedir—Çünkü her seferinde yordam çağrılır,Yığının argümanlarin ve yerel değiskenlerin tahsisleri yığında yukarı doğru büyür.Bu daha önce Assembler dersinde görmüstük.
UNIX> **testaddr4**
argc = 1. &argc = 0xffbee15c, &argv = 0xffbee160, &i = 0xffbee104
argc = 0. &argc = 0xffbee0e4, &argv = 0xffbee0e8, &i = 0xffbee08c
UNIX> **testaddr4 v**
argc = 2. &argc = 0xffbee154, &argv = 0xffbee158, &i = 0xffbee0fc
argc = 1. &argc = 0xffbee0dc, &argv = 0xffbee0e0, &i = 0xffbee084
argc = 0. &argc = 0xffbee064, &argv = 0xffbee068, &i = 0xffbee00c
UNIX> **testaddr4 v o l s**
argc = 5. &argc = 0xffbee144, &argv = 0xffbee148, &i = 0xffbee0ec
argc = 4. &argc = 0xffbee0cc, &argv = 0xffbee0d0, &i = 0xffbee074
argc = 3. &argc = 0xffbee054, &argv = 0xffbee058, &i = 0xffbedffc
argc = 2. &argc = 0xffbedfdc, &argv = 0xffbedfe0, &i = 0xffbedf84
argc = 1. &argc = 0xffbedf64, &argv = 0xffbedf68, &i = 0xffbedf0c
argc = 0. &argc = 0xffbedeec, &argv = 0xffbedef0, &i = 0xffbede94
UNIX>

Şimdi,yığın ayıralım.Böyle bir programla yığın cok fazla bellek ayırması yapabilir. breakstack1.c. bu tür bir pogramdır.Bu program sonsuz öz yenileme gerçekleştirir,ve iptr değişkenin içinde her özyinelemesinde yığın bellekten 10000 byat yer ayrılır.Bunu çalıştırdığınızda, özyineleme arama yapıldığında ve yığın 0x0xff3f0000 dallandığında segmantasyon ihlali aldığınızı görürsünüz:

```
UNIX> breakstack1
...
&c    = 0xff3fa347, iptr = 0xff3f7c30  ...  ok
&c    = 0xff3f7bbf, iptr = 0xff3f54a8  ...  ok
&c    = 0xff3f5437, iptr = 0xff3f2d20  ...  ok
Segmentasyon Hata
UNIX>
```

Çoğu zaman sonsuz özyineleme ve yığında taşma olabilir,Uygunsuz talimat yerine Segmantasyon hatası alırız.Fikir edinmek için,talimatların yığın parçasıyla ilgili olduğunu düşünülebilir..

Bellek ayırmanın ikinci bir yolu da çok fazla yerel bellek tahsisi yapmaktır.Örnek olarak breakstack2.c. bakın.Çalıştığında bellekten 10M yer tahsisi eder. 0xff3f0000 dan daha küçük bellek adreslerini referans etmesi ve çalışmasından dolayı çalışma sırasında iç parçalanma olur.Tam olarak nerede segmantasyon hatası olur? Düşününün…cevap aşağıda.

Printf ‘i çağırmak için yığından iptr ile kod çağrılırsa segmentasyon hatası olur.Bunun nedeni yığın işaretçisi(stack pointer) boşluğu göstermesidir. yığın işaretçisine referans gösterilmezse,program çalışmış olabilirdi.Örnegin, breakstack3.c. deneyin.

```
UNIX> breakstack3
a Çağrısı .  i = 1
a Yapıldıktan sonra.  i = 5
UNIX>
```

Siz anlayabilmelısınız,ve bu olguyu açıklayabilmelisiniz.