MALLOC()-1

Bu konu sbrk() ve malloc() ile ilgilidir.

Geçen ders bazı bellek elemanları üzerinde durduk- - kod segmentinin içindeki son adresin &etext ve global segmentindeki son adresinde &end olduğunu öğrendik.Program yürütüldüğünde malloc() kullanarak heapden(yığın) bellek tahsis edilir ve böylece heap büyür.Heapin boyutunu anlamak için brk() veya sbrk() kullanılır.Her iki de sistem çagrısıdır ve onların man sayfaları okunabilir.Sadece sbrk() üzerinde duracağız.

```
caddr_t sbrk(int incr);
```

'caddr_t' bir c adresi göstericidir, (char *) veya (void *) ile aynıdır.

İncr heap alanına çok sayıda bayt vermek için işletim sisteminde belirtilmiştir.sbrk() çağrılmadan önce yığındaki son değeri geri döndürür. Çağrıldıktan sonra heapin sonuna yeni adres eklenir.

```
sbrk(incr) + incr;

Eğer sbrk(0) çağrılırsa heapin sonundaki döndürülür.
Şimdi malloc() bütün sbrk() çağrılarını bellekteki heap alanından tahsis
eder.sbrk() sadece rütinler için çağrılır.malloc() ve sbrk() bellekten alan
alma yoludur fakat malloc() daha etkilidir.

fb2.c

#include < stdio.h >
#include < sys/types.h >

main()
{
  int *i1, *i2;

  printf("sbrk(0) before malloc(4): 0x%x\n", sbrk(0));
  i1 = (int *) malloc(4);
  printf("sbrk(0) after `i1 = (int *) malloc(4)': 0x%x\n", sbrk(0));
  i2 = (int *) malloc(4);
  printf("sbrk(0) after `i2 = (int *) malloc(4)': 0x%x\n", sbrk(0));
  printf("i1 = 0x%x, i2 = 0x%x\n", i1, i2);
}
```

Sbrk(0)'dan önce veya sonra bazı malloc() çağrıları yazılır.Burada çalışan sonucu hydra3a üzerindedir.

```
UNIX> fb2
sbrk(0) before malloc(4): 0x21ab0
sbrk(0) after `i1 = (int *) malloc(4)': 0x23ab0
sbrk(0) after `i2 = (int *) malloc(4)': 0x23ab0
i1 = 0x21ac0, i2 = 0x21ad0
UNIX>

Malloc() çağrılarından sonra sbrk() değişmez.Neden?çünki malloc()
tamponlanır.sbrk() çagrısı büyük bir numara ile çağrılır- -Bu bufferdan 12k
ve 8k bellek dışarı verir.i1 ve i2yi tahsis ettikten sonra yinede birsürü
bellek alanı var- -malloc() çagrısı 0x0x21ad0 den 0x0x23ab0 ye yine
sbrk()çağrısından önce kullanılabilir.Kabaca 8160 bayta denk gelir.Böylece
fb2ac de malloc(8164) yaptığımızda i1 ve i2 sonra tahsis edilir,sbrk()yi
daha fazla bellek alanı almak için çağırdık ve bu durumda:
```

```
UNIX> fb2a
sbrk(0) before malloc(4): 0x21b68
sbrk(0) after `i1 = (int *) malloc(4)': 0x23b68
sbrk(0) after `i2 = (int *) malloc(4)': 0x23b68
i1 = 0x21b78, i2 = 0x21b88, sbrk(0)-i2 = 8160
sbrk(0) after `i3 = (int *) malloc(8164)': 0x25b68
i3 = 0x21f78
UNIX>
```

fb3.c.

```
#include < stdio.h >

main()
{
  int j, *buf;

  for (j = 0; j < 10; j++) {
    buf = (int *) malloc(4);
    printf("malloc(4) returned 0x%x\n", buf);
  }
}
UNIX> fb3
malloc(4) returned 0x219d0
malloc(4) returned 0x219e0
malloc(4) returned 0x219f0
malloc(4) returned 0x21a00
malloc(4) returned 0x21a10
malloc(4) returned 0x21a20
malloc(4) returned 0x21a30
malloc(4) returned 0x21a40
malloc(4) returned 0x21a50
malloc(4) returned 0x21a60
UNIX>
```

Maloc() dan dönüş değerinin 16 bayt daha büyük olduğuna dikkat edeceğiz.Sadece 4 bayt ayrıldıgından beri 4 bayt daha fazla ayrılması bekleniyor.malloc() her çağrıldığında ekstra bayt ayırır ve bunun muhasebesini kendi yapar.free() çağrımına ekstra baytlar yardımcıdır.Bu ekstra baytlar döndürülen bellekten önce sıksık tahsis edilir.

Nasıl tahsis yapar: Bellek bölgelerinde malloc() kullanarakbaşlangıç adreslerini yazdırır, ve degerleri ve de 2 kelime olan başlangış adreslerini bir yere alır.fb4.c bakınız:

Fb4.c

/* fb4.c

Jim Plank

CS360 -- Systems Programming

Malloc lecture #1

```
  October, 1996 */


#include <stdio.h>
#include <sys/types.h>


main()
{
  int *buf;
  int i, sz;


  i = 1000;
  printf("sbrk(0) = 0x%x\n", sbrk(0));
  for (sz = 4; sz < 32; sz += 4) {
    buf = (int *) malloc(sz);
    buf[0] = i;
    i++;
  printf("Allocated %d bytes.  buf = 0x%x, buf[-1] = %d, buf[-2] = %d, buf[0] = %d\n",
          sz, buf, buf[-1], buf[-2], buf[0]);
  }


  sz = 100;
  buf = (int *) malloc(sz);
  buf[0] = i;
  i++;
  printf("Allocated %d bytes.  buf = 0x%x, buf[-1] = %d, buf[-2] = %d, buf[0] = %d\n",
          sz, buf, buf[-1], buf[-2], buf[0]);
  printf("sbrk(0) = 0x%x\n", sbrk(0));
```

}

```
UNIX> fb4
sbrk(0) = 0x70f8
Allocated 4 bytes.  buf = 0x61a8, buf[-1] = 0, buf[-2] = 16, buf[0] = 1000
Allocated 8 bytes.  buf = 0x61b8, buf[-1] = 0, buf[-2] = 16, buf[0] = 1001
Allocated 12 bytes.  buf = 0x61c8, buf[-1] = 0, buf[-2] = 24, buf[0] = 1002
Allocated 16 bytes.  buf = 0x61e0, buf[-1] = 0, buf[-2] = 24, buf[0] = 1003
Allocated 20 bytes.  buf = 0x61f8, buf[-1] = 0, buf[-2] = 32, buf[0] = 1004
Allocated 24 bytes.  buf = 0x6218, buf[-1] = 0, buf[-2] = 32, buf[0] = 1005
Allocated 28 bytes.  buf = 0x6238, buf[-1] = 0, buf[-2] = 40, buf[0] = 1006
Allocated 100 bytes.  buf = 0x6260, buf[-1] = 0, buf[-2] = 112, buf[0] =
1007
sbrk(0) = 0x70f8
UNIX>
```

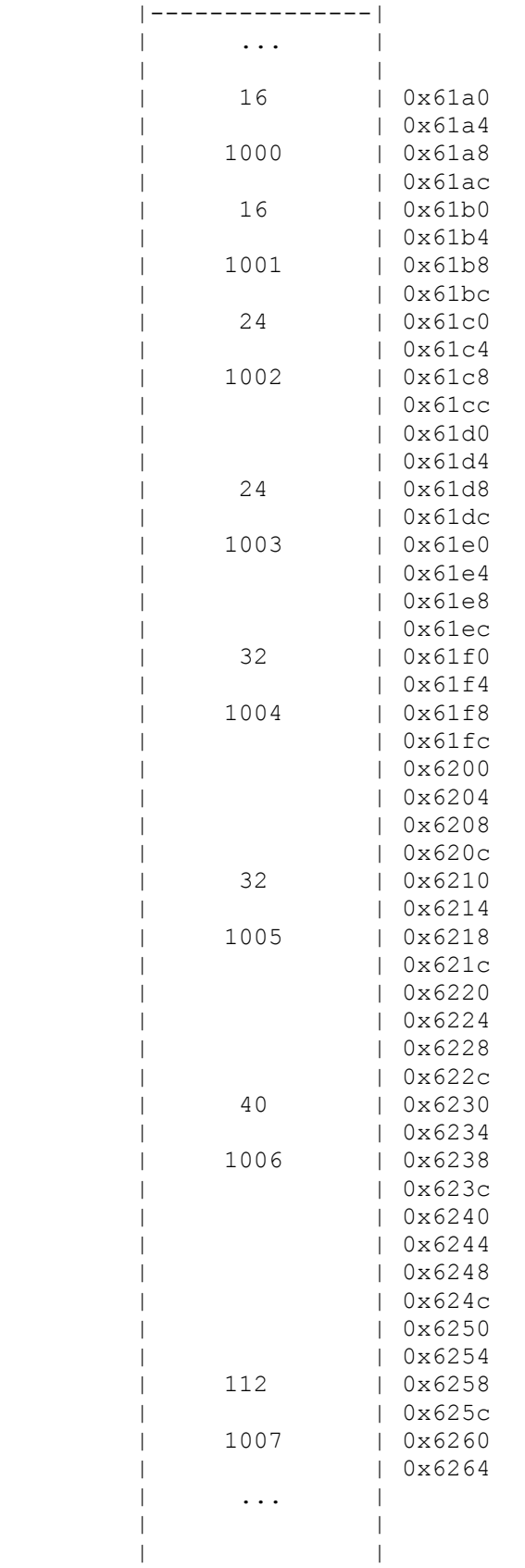
Yani malloc() ilk çağrıldığında yığına bakılır ve buf[0] i=1000 yapılır.

```
|---------------|
|      ...      |
|               |
|      16       | 0x61a0
|               | 0x61a4
|     1000      | 0x61a8  <--------- return value
|               | 0x61ac
|               | 0x61b0
|               | 0x61b4
|      ...      |
|               |
|               |
|               |
|---------------| 0x70f8 (sbrk(0));
```

Şimdi, malloc () ikinci kez çağrıldığında (buf = malloc (8)), malloc () 0x61b8 döndürür. Sonra buf [0] i = 1001 ayarlanır, yığın aşağıdaki gibi görünüyor:

```
|---------------|
|      ...      |
|               |
|      16       | 0x61a0
|               | 0x61a4
|     1000      | 0x61a8
|               | 0x61ac
|      16       | 0x61b0
|               | 0x61b4
|     1001      | 0x61b8  <--------- return value
|               | 0x61bc
|               | 0x61c0
|               | 0x61c4
|      ...      |
|               |
|               |
|---------------| 0x70f8 (sbrk(0));
```

sonsbrk (0) çağrıldığında, yığın şöyle olacaktır:



```
          |---------------| 0x70f8 (sbrk(0));
```

Malloc(); tampona işletim sisteminden bellek almak için sbrk() yı çağırır, ve bu çağırıldığında buffer dan bellek alınır. Tampon bölge alanının dışında çalıştığıda , daha fazla alan için sbrk() çağırır.

Neden malloc(4) ve malloc(8) 16 byte yer kaplarken, malloc(16) ve malloc(16) 24 byte yer kaplar ? Çünkü malloc() 8 byte ın katlarını tahsis eder. Böylece malloc(4) ve malloc(8) kullanıcı için 8 bayt ayırır artı muhasebe için 8 byte ayrılır. Malloc(12) ve malloc(16) kullanıcı için 16 byte ayırır , artı muhasebe için 8 byte ayırır.Malloc(100) kullanıcılar için 104 byte ayırır, ekstra muhasebe için 8 byte ayırır. Neden bu dolguyu malloc() yapmaz ? Öyle ki döndürülen adresler 8 in katları olur ve böylece her türde pointer için geçerli olacaktır. Malloc bunu yapmadığını varsayarsak, bunun yerine herhangi bir işaretçi dönebilir. Eğer aşağıdaki gibi olursa:

```
int *i;

  i = (int *) malloc(4);
  *i = 4;
```

Bu bus error oluşturabilir, çünkü Malloc() 4 ün katı olmayan değer döndüremeyebilir. Bu halde, malloc() 8 in katı döndürür, öyleki çift ve uzun tamsayı pointerları bus error oluşturmaz. Malloc() belleği nasıl dağıtacağını nasıl bilir? Global değişken kullanılır, örneğin aşağıdaki gibi global değişkeler olabilir:

```
char *malloc_begin = NULL;
char *malloc_end = NULL;
```

malloc içağrıldığında ilk olarak malloc_begin = NULL denetlenir. Öyleyse , bir buffer almak için sbrk() çağırılır. Bu buffer ın başlangıcını ve sonunu belirtmek için, malloc_begin ve malloc_end 'i kullanır. Malloc() çağırıldığında buffer başından belleği oluşturur ve malloc_begin buna göre güncellenir. Eğer tamponda yeterice yer yoksa , o zaman daha fazla bellek almak için sbrk() çağırılır ve malloc_end genişlemiş tamponu göstermek için artırılır.

Şimdi bu hiç free() kullanmadan , nasıl malloc yazacağımızı tanımlar. Free() çağırıldığında, malloc() belleğin yeniden kullanılabilir olması gerekmektedir. Bu da bazı şeyleri malloc() ile daha sofistike yapmanız gerektiğini getirir . Bunu ders 3 de konuşucaz . O zamana kadar bunu düşünün…

Malloc hakkında thisnote 'u okumadıysanız, hemen okumalısınız ..

EK :Thisnote

He sistem kedi yolunda malloc() uygular .fb4 aşğıdaki çıktıyı verir:

```
UNIX> fb4
sbrk(0) = 0x70f8
Allocated 4 bytes.  buf = 0x61a8, buf[-1] = 0, buf[-2] = 16, buf[0] = 1000
Allocated 8 bytes.  buf = 0x61b8, buf[-1] = 0, buf[-2] = 16, buf[0] = 1001
Allocated 12 bytes.  buf = 0x61c8, buf[-1] = 0, buf[-2] = 24, buf[0] = 1002
Allocated 16 bytes.  buf = 0x61e0, buf[-1] = 0, buf[-2] = 24, buf[0] = 1003
Allocated 20 bytes.  buf = 0x61f8, buf[-1] = 0, buf[-2] = 32, buf[0] = 1004
Allocated 24 bytes.  buf = 0x6218, buf[-1] = 0, buf[-2] = 32, buf[0] = 1005
Allocated 28 bytes.  buf = 0x6238, buf[-1] = 0, buf[-2] = 40, buf[0] = 1006
```

```
Allocated 100 bytes.  buf = 0x6260, buf[-1] = 0, buf[-2] = 112, buf[0] =
1007
sbrk(0) = 0x70f8
```

Bu, 1996 yılında Cetus makinelerindeki çıktı idi. 1999 yılında, cetus1a daki çıktı:

```
UNIX> fb4
sbrk(0) = 0x20b08
Allocated 4 bytes.  buf = 0x20b18, buf[-1] = 0, buf[-2] = 9, buf[0] = 1000
Allocated 8 bytes.  buf = 0x20b28, buf[-1] = 0, buf[-2] = 9, buf[0] = 1001
Allocated 12 bytes.  buf = 0x20f20, buf[-1] = 0, buf[-2] = 17, buf[0] =
1002
Allocated 16 bytes.  buf = 0x20f38, buf[-1] = 0, buf[-2] = 17, buf[0] =
1003
Allocated 20 bytes.  buf = 0x21528, buf[-1] = 0, buf[-2] = 25, buf[0] =
1004
Allocated 24 bytes.  buf = 0x21548, buf[-1] = 0, buf[-2] = 25, buf[0] =
1005
Allocated 28 bytes.  buf = 0x21d30, buf[-1] = 0, buf[-2] = 33, buf[0] =
1006
Allocated 100 bytes.  buf = 0x22730, buf[-1] = 0, buf[-2] = 105, buf[0] =
1007
sbrk(0) = 0x22b08
```

Bu yukarıdaki örneğin çıktısı farklı, ama gördüğünüz gibi, işlevsel olarak eşdeğer. döndürülen işaretçi ayrılan bayt sayısı yediden az iki kelimeden önce bir
sözcük değeri.Neden? Bilmiyorum - ama yapmanız gereken tüm almak için yedi ekleyin.
Masamın üzerinde Pentium kutusu Linux 2.2.10 çalışıyor ve şu çıkışı vardır:

```
UNIX> fb4
sbrk(0) = 0x8049778
Allocated 4 bytes.  buf = 0x8049780, buf[-1] = 17, buf[-2] = 0, buf[0] =
1000
Allocated 8 bytes.  buf = 0x8049790, buf[-1] = 17, buf[-2] = 0, buf[0] =
1001
Allocated 12 bytes.  buf = 0x80497a0, buf[-1] = 17, buf[-2] = 0, buf[0] =
1002
Allocated 16 bytes.  buf = 0x80497b0, buf[-1] = 25, buf[-2] = 0, buf[0] =
1003
Allocated 20 bytes.  buf = 0x80497c8, buf[-1] = 25, buf[-2] = 0, buf[0] =
1004
Allocated 24 bytes.  buf = 0x80497e0, buf[-1] = 33, buf[-2] = 0, buf[0] =
1005
Allocated 28 bytes.  buf = 0x8049800, buf[-1] = 33, buf[-2] = 0, buf[0] =
1006
Allocated 100 bytes.  buf = 0x8049820, buf[-1] = 105, buf[-2] = 0, buf[0] =
1007
sbrk(0) = 0x804a000
```

Şimdi yedi eksi boyutu gösterici önce ilk kelimenin içindedir.
İşte Ultrix'li çalışan bir DEC istasyonuna çıktısı:

```
UNIX> fb4
sbrk(0) = 0x10001100
Allocated 4 bytes.  buf = 0x10005000, buf[-1] = 268435695, buf[-2] = 0,
buf[0] = 1000
Allocated 8 bytes.  buf = 0x10006000, buf[-1] = 268435951, buf[-2] = 0,
buf[0] = 1001
Allocated 12 bytes.  buf = 0x10006010, buf[-1] = 268435951, buf[-2] = 0,
buf[0] = 1002
```

```
Allocated 16 bytes.  buf = 0x10007000, buf[-1] = 268436207, buf[-2] = 0,
buf[0] = 1003
Allocated 20 bytes.  buf = 0x10007020, buf[-1] = 268436207, buf[-2] = 0,
buf[0] = 1004
Allocated 24 bytes.  buf = 0x10007040, buf[-1] = 268436207, buf[-2] = 0,
buf[0] = 1005
Allocated 28 bytes.  buf = 0x10007060, buf[-1] = 268436207, buf[-2] = 0,
buf[0] = 1006
Allocated 100 bytes.  buf = 0x10008000, buf[-1] = 268436719, buf[-2] = 0,
buf[0] = 1007
sbrk(0) = 0x10008ffc
UNIX>
```

Shot a int döndürdüğümde aldığım çıktı :

```
UNIX> fb5
sbrk(0) = 0x10001100
Allocated 4 bytes.  buf = 0x10005000, buf[-1] = 4096, buf[-2] = 239, buf[0]
= 1000
Allocated 8 bytes.  buf = 0x10006000, buf[-1] = 4096, buf[-2] = 495, buf[0]
= 1001
Allocated 12 bytes.  buf = 0x10006010, buf[-1] = 4096, buf[-2] = 495,
buf[0] = 1002
Allocated 16 bytes.  buf = 0x10007000, buf[-1] = 4096, buf[-2] = 751,
buf[0] = 1003
Allocated 20 bytes.  buf = 0x10007020, buf[-1] = 4096, buf[-2] = 751,
buf[0] = 1004
Allocated 24 bytes.  buf = 0x10007040, buf[-1] = 4096, buf[-2] = 751,
buf[0] = 1005
Allocated 28 bytes.  buf = 0x10007060, buf[-1] = 4096, buf[-2] = 751,
buf[0] = 1006
Allocated 100 bytes.  buf = 0x10008000, buf[-1] = 4096, buf[-2] = 1263,
buf[0] = 1007
sbrk(0) = 0x10008ffc
UNIX>
```

Gerçekten tuhaf. Bir açıklama da tahmin edebilir misiniz? Ben yapabileceğimdüşünüyorum, ama tarama işlemi biraz sürecektir. Bizi izlemeye devam edin ☺