Malloc () ve Free() fonksiyonları

Malloc() sınıfı büyük bir buffer (yığın) bellek tutar. Kullanıcı ne kadar yer tahsisinde bulunursa malloc() o kadar yer açmaktadır sistemde.Eger buffer bellekte kullanıcının istediği kadar yer yoksa, sbrk() fonksiyonu çağrılır. Yeterince heap depolama alanı ayrılır.

Malloc() fonksiyonu aslında farklı bir şekilde çalışır.Aslında hafızadaki boş alanların listesini tutar.malloc() fonksiyonu çağrıldıgında ise istenen alana göre hemen o yeri bulur ve kullancıya o istediği yeri tahsis eder ve o yer kullanıldığı için artık onu kendi listesinden siler.

Malloc() fonksiyonuyla aslında dinamik bellek kullanımı yapıyoruz ve sistemden istediğimiz kadar yer yanihafıza talebinde bulunuyoruz .Böylece gereksiz yere hafızayı kullanmış olmuyoruz .Ne kadar ihtiyacımız varsa o kadar alıyoruz.malloc() fonksiyonu da bize istediğimiz kadar yer veriyor al kardeşim kullan diyor.

Ve şimdi gelelim Free() fonksiyonuna .Bakalım o ne işe yarıyor. Bildiğimiz gibi malloc() serbest listeden bize bellek alır . Free() ise aldığımız o yerle eğer işimiz bittiyse tekrar geri vermemizi sağlar. Yani aldığımız bellek alanını serbest bırakır.

Başlangıçta serbest olan liste olan liste boştur. Malloc() çağrıldıgında o listeden alan tahsisi yapılır.Ve o alan kullanıldıktan sonra free() ile tekrar serbest bırakılır.

Not: **sbrk()fonksiyonu , aynı sistem çağrısı giriş noktası(system call entry point)'ni kullanır ve işlemin veri bölgesi(data region)'nın en yüksek adres değerini, belirten artma miktarına uygun olarak arttırır.**

Bir örneğe geçmeden önce ben bunun serbest olan listeğe nasıl uygulanacağı hakkında bir şeyler söylemek istiyorum.

Serbest liste başı olan malloc head alanı global bir değişkendir.Başlangıçta malloc head değeri null olacaktır. Malloc() fonksiyonu ilk çağrıldığında sbrk() fonksiyonu ile boş listeye bazı bellek değerleri alınır. Şimdi serbet liste öğesi haline gelen bellek alanını kullanmak için birkaç byte liste yapısı(lis structure) kullanılır.

Başka bir deyişle eger serbest listede bir öbek alanı var ise ve öbek alanı atıyorum 12 byte ise siz bu ilk 12 bytelık alanı bellekteki bu liste yapısından kullanabilirsiniz. Örneğin:

bytes 0-3: Integer -- >bu öbek boyutu

bytes 4-7: serbest listedeki sonraki öbek göstericisi

bytes 8-11: serbest listedeki önceki öbek göstericisi

Peki bunu nasıl kullanacağız. Aşağıdaki gibi bir typedef yapı kurarak:

```
typedef struct flist {
   int size;
   struct flist *flink;
   struct flist *blink;
} *Flist;
```

Serbest listenin öğesi olan ve yeri s ile başlayan (s (**char*),(void*) veya (caddr_t**) olabilir) öbeğe hafızanın başlangıç yeri olarak mudahale etmek istediğniz zaman :

```
Flist f;

  f = (Flist) s;
```

yapısını kullanabilirsiniz.

Yani aşagıdajki gibi bir başlangıç yapılabilir. **malloc_head == NULL**. Örnek. Serbest liste boş.Şimdi kullanıcının malloc(16) yı çağırdığını varsayalım. Sizin malloc programı serbest listede bellekte hiç yer olmadığını gördü. Ve bu yüzden sbrk(8192) fonksiyonunu çağırdı. Bu foksiyon veri, saklamak için 8K gibi bir bellek tahsis etti. Sbrk(8192) nin 0x6100 gibi bir adres döndürüdğünü varsayalım.

Ve sizin belleğiniz aşagıdaki gibi bir görünüm alır:

```
malloc_head == NULL

         |---------------|
         |               | 0x6100 (heap başlangıcı)
         |               | 0x6104
         |               | 0x6108
         |               | 0x610c
         |               | 0x6120
         |               |
              .....      |
         |               |
         |---------------| 0x8100 (heap sonu -- sbrk(0))
```

Serbest listedeki belleğe parça koymak için listede üzerine bağlamak gerekmektedir. Bellek aşagıdaki gibi görünür.

```
malloc_head == 0x6100

         |---------------|
         |    8192       | 0x6100 (start of heap)
         |    NULL       | 0x6104
         |    NULL       | 0x6108
         |               | 0x610c
         |               | 0x6110
         |               |
              .....      |
         |               |
         |---------------| 0x8100 (end of heap -- sbrk(0))
```

Şimdi sizin kullanıcın ihtiyacı olduğu 16 bytelık alanı karşılmanız gerekir. Siz bu parçağı bellekte iki parçaya ayırdınız.

Bu 24 bytedan(16 byte kullanıcı için 8 byte defter için) ve kalan 819-24=8168 byte olur.

Bellek üzerine ikinci bir öbek koydun ve kullanıcı için ayrılan 16 byte pointer ile döndürdün.

Aşagıdaki gibi bir yapı oluşur.

```
malloc_head == 0x6118

         |---------------|
         |     24        | 0x6100 (start of heap)
         |               | 0x6104
         |               | 0x6108  <-- beginning of 16 bytes for the user
         |               | 0x610c
         |               | 0x6110
         |               | 0x6114
         |     8168      | 0x6118
         |     NULL      | 0x611c
         |     NULL      | 0x6120
              .....      |
         |               |
         |---------------| 0x8100 (end of heap -- sbrk(0))

       0x6108 is returned to the user.
```

Şimdi kullanıcının malloc(8)i çağırdığını varsayalım. Aslında aynı şey . Bellekteki 16 bytelık öbekleri dilimlre ayırdın ve pointer ile bunları 8 byte olarak kullnıcıya tahsisi ettin. Diğer 8 bytelar ise defter(bookkeeping) için kullanılmaktadır.
Kalan 8152 bayt serbest listeye(free list) geri konur. Diğer deyişle malloc(8) den sonra yığın görünümü bu şekilde olur:

```
malloc_head == 0x6128
         |---------------|
         |     24        | 0x6100 (start of heap)
         |               | 0x6104
         |               | 0x6108
         |               | 0x610c
         |               | 0x6110
         |               | 0x6114
         |     16        | 0x6118
         |               | 0x611c
         |               | 0x6120
         |               | 0x6124
         |     8152      | 0x6128
         |     NULL      | 0x612c
         |     NULL      | 0x6130
              .....      |
         |               |
         |---------------| 0x8100 (end of heap -- sbrk(0))

0x6120 is returned to the user.
```

Şimdi kullanıcının **free(0x60108)** gibi bir fonksiyon çalıştırdığını düşünelim.

Yani kullanıcının malloc() ile hafızadan aldığı bellek alanının serbest bırakmaktadır. Yani serbest liste tarafından dilimlenen 24 byte tekrar serbest listeye geri döndürülmektedir.

Free () fonksiyonu serbest listedeki düğümlere bu byte geri çevirip serbest listedeki ilk düğümden itibaren yerşleştirecektir.
Free() fonksiyonun işi bittiğinde bellek şu şekilde görünür.

```
malloc_head == 0x6100
         |---------------|
         |     24        | 0x6100 (start of heap)
         |   0x6128      | 0x6104
         |   NULL        | 0x6108
         |               | 0x610c
         |               | 0x6110
         |               | 0x6114
         |     16        | 0x6118
         |               | 0x611c
         |               | 0x6120
         |               | 0x6124
         |     8152      | 0x6128
         |     NULL      | 0x612c
         |    0x6100     | 0x6130
              .....      |
         |               |
         |---------------| 0x8100 (end of heap -- sbrk(0))
```

Diğer bir değişle serbest liste şimdi iki düüme sahiptir.

0x6128 ile başlayan ve 24 byte içeren
0x6100 ile başlayan ve 8152 byte içeren.

Örneklere devam ediyoruz. Sizinde anladığınız gibi hepsi aynı mantıkla ilerlemekte . şimdi kullanıcı malloc(24) ü çalıştırmış olsun .
Bakalım ne olacak.
Malloc() free list de 32 byte sahip bir öbek parça bulmaya çalışacaktır.
Bunun 24 byte kullanıcı için , kalan 8 byte ise bookkepping için.
Böylece serbest bölgedeki ilk öbek (yığın) rededilir. Çünkü sadece 24 byte alan vardır.Bunun yerine 32 byte lik dilimler açılacaktır. Ve bellek aşagıdaki gibi görünür.

```
malloc_head == 0x6100
         |---------------|
         |     24        | 0x6100 (start of heap)
         |   0x6148      | 0x6104
         |   NULL        | 0x6108
         |               | 0x610c
         |               | 0x6110
         |               | 0x6114
         |     16        | 0x6118
         |               | 0x611c
         |               | 0x6120
         |               | 0x6124
         |     32        | 0x6128
         |               | 0x612c
         |               | 0x6130
         |               | 0x6134
         |               | 0x6138
         |               | 0x613c
         |               | 0x6140
         |               | 0x6144
         |     8130      | 0x6148
         |     NULL      | 0x614c
         |     0x6100    | 0x6150
              .....      |
         |               |
         |---------------| 0x8100 (end of heap -- sbrk(0))
```

Ve 0x6130 değeri kullanıcıya döndürüldü. Bu kullanıcı için ayrılan bellek 24 bayttır.Ve son olarak kullanıcının malloc(8) i çağırdığını varsayalım. Biz free list te 16 byte lık öbekler bulmak istiyoruz.(kullnıcı için 8 byte bookkepping için 8 byte)

İlk olarak 24 byte gibi bir parça karşımıza çıkacaktır. Ancak bi soru var. Biz bundan 16 byte nasıl koparacağız veya 24 byte ın hepsini mi kullanacağız.,Cevap 24 byte ın hepsini kullanacağız. Bunun 16 byte ını kullanıcıya vereceğiz(ancak 8 byte istemişti) geri kalan 8 byte bookkepping için olacaktır.Peki neden? Çünkü 16 bytelık dilim alırsak sadece 8 byte artan olacak. Ve free list te pointers ile deplamak için yeterli değil. Bu yüzden bu 24 bytelık dilimi kullanıcı ve free list için alacağız.

```
malloc_head == 0x6148
          |---------------|
          |      24       | 0x6100 (start of heap)
          |               | 0x6104
          |               | 0x6108
          |               | 0x610c
          |               | 0x6110
          |               | 0x6114
          |      16       | 0x6118
          |               | 0x611c
          |               | 0x6120
          |               | 0x6124
          |      32       | 0x6128
          |               | 0x612c
          |               | 0x6130
          |               | 0x6134
          |               | 0x6138
          |               | 0x613c
          |               | 0x6140
          |               | 0x6144
          |      8130     | 0x6148
          |      NULL     | 0x614c
          |      NULL     | 0x6150
               .....      |
          |               |
          |---------------| 0x8100 (end of heap -- sbrk(0))

0x6108 is returned to the user.
```

Not: malloc head ve 0x6148 den başlayan öbek pointerlar free list den çıkaraılcak öbeklere yansıtılacak şekilde değiştirilir.

Free () fonksiyonu hakkında daha fazla bilgi

Free fonksiyonu verimli ve dinamik bellek kullanmayı sağlayan en önemli fonksiyonlardandır. Bu fonksiyon ile malloc ya da calloc ile daha önceden ayırdığınız bellek bölgelerini işletim sistemine **geri iade** edebilirsiniz. Parametre olarak içine daha önce ayırdığınız bellek bölgesinin başlangıç adresini göndermelisiniz.

Yani daha önce malloc ile aldıgımız alanı free ile serbest bırakıyoruz.malloc dan dondürülmüş olan bellekleri free ile kullanılırken adresleri ile çağrılmalıdır. Bu açık bir şekilde olmalıdır.peki neden ?

Çünkü free adresin arkasında bulunan 8 byte lık hafızanın serbest bırakılmasını beklemektedir. Eğer malloc ile tahsis edilmeyen bir adresi free() ile serbest bırakmaya çalışırsanız garip şeylerle karşılaşabilirsiniz.

Örneğin **badfree.c**. programcıgına bakın:

```
main()

{ int i, j; char *s;

 for (i = 0; i < 1000; i++)

 { s = (char *) malloc(10);

 printf("0x%x\n", s);

 for (j = 0; j < 10; j++)

s[j] = 'a' + j % 26;

free(s+8);

 }

 }
```

Free() bir dizi ortasında adres ile bunu çağırır. Ve bir süre sonra malloc() içerğinin boşalmasına neden olur. Malloc ilk olarak bufferdan bir bellek tahsis ediyor görünüyor. Bu belleği bir süre alır kullanır. Sonra zaten serbest kalacaktır. Yani aslında sahte free'd blok kullanıyor. Sınıf içinde kullanılan malloc tarafından ayrılan alanların asılında free tarafından takip edilidiği malloc ca bilinmektedir. Ve free () bunların gecerli bir arguman olup olmadıgını anlamak için arguman kontrolü yapmaktadır. Bu aslında iyi bir öneridir. Ancak malloc geniş bir alanda bu adresleri daha hızlı arayabilmek için hash table veya red-black tree structure yapısını kullanacaktır. Aksi takdirde bunu free() ile yapmak çok uzun sürer. Ancak rb.o uygulamaları red-black tree yapısını kullanmaz ,buna dikkat edilmesi gerekmektedir. Çünkü rb rutinleri malloc ve free ile aranmaktadır. Free() için hata denetimi yapabilmek için başka bir yol da bookkeeping byte kullanmaktır. Ve böylece malloc() tarafından döndürülen adrsten önce 4 byte kullanılmadığını fark edceksiniz. Malloc() kullandığınız zaman ne yapmanız gerektiğinin sağlaması için bu sözcükle ayarlama yapılır. Sonra free() çağırdıgın zaman istenilen sağlamanın olup olmadığını görmek için bu kelime kontrol edilecektir.

Free Coalescing blokları (coalesce→bir araya gelmek ,kaynaşmak)

Free() fonksiyonu çagrıldığı zaman free list hafıza alanına bir yığın yerleştirecektir. Bellekten hemen önce veya sonra serbest bir yığın zamanı olabilir. Eğer öyleyse daha çok free liste 3 olasılıkla free öbekler olmasındansa bunları birleştirmek daha mantıklı olcaktır. İşte buna free coalescing blokları denmektedir. Bu gibi durumlarda coalescing uygulayabiceğiniz bir yoldur.

Not:Bunu kendi laboratuarınızda uygulamak zorunda değilsiniz. Ancak isterseniz bu hack için iyi bir egzersiz sayılabilir.

Bunun yerine bookkepping için tahsis edilen 8 byten kullanıcı için ayrılan bellek öncesi 4 sonrasında da 4 byte şeklinde tutabilirsiniz. Bir belleğe bir yığın tahsis edildiğinde diğer bir deyişle yığınların yani öbeklerin aynı boyutta olması ayarlanır. Free liste free öbekler arasında farklı bir düzen vardır. Bu öbeklerin boyutu en az 16 byte olmalıdır. İlki yukarda (epey yukarda sanırım ilk sayfada) açıklandığı gibi 12 byte olmalıdır. Sadece bouytlu yerine boyutu ilk 4 byte tutulmalıdır. Sonraki 4 bytelık öbekleri hold-size tutmak gerekir. Bu nasıl yapılır aşağıya bak:

Bir yığını serbest bıraktığın zaman free() den pointer arkasında 8 bayte gectiğni görürsünüz.

Bunların her biri olmalıdır bakınız:

*Bellekten verilen bir yığın boyutu önce parçalar halinde ayrılır.

*eğer bu parçalar serbest ise bellekte verilen parçaların boyutu ayarlanır.

Başka bir deyişle eğer 8 byte word verilen pointer pozitifse serbest bırakılması gereken alan tahsis edilmiştir.

Mevcut parçadan sonra word negatifse daha sonra takip eden yığın serbesttir ve coalescing edileblir. Aksi takdirde öbek tahsis edilmiştir.

Açıkçası, emin olmak için yığının başında ve sonunda dikkat çekmek gerekir ki çekleriniz tüm işlerinizdir. Bu zor değildir.