**Parçalanma**

Parçalanma birçok yerde kullanılan bir kavramdır. Bir kullanıcı programı tarafından bellek tahsisi yapıldığında parçalanma oluşur ama bellek kullanılmaz. Malloc ders 2 deki örnek yığında da gösterildiği gibi, malloc fonksiyonu bellekte 8130 bayt yer tutar. Bu bellek program için işletim sistemi tarafından tahsis edilen bellektir ama kullanılmaz.

Sınırsız belleğe sahip makine olsaydı, parçalanma bir sorun olmazdı. Ama makinelerin çoğunda 16 ilâ 100 megabayt bellek kullanılır ve çoklu programlamayı desteklemeleri istenir. Diğer bir deyişle makine üzerinde aynı anda çalışan birçok program parçacığı vardır ve her biri belleği kullanır. Bu program parçacıkları bellekten yer alabilmek için rekabet etmek durumunda kalır ve bu durum bellek kaybına yok açar. Bu da parçalanmanın bizim için kötü olduğu anlamına gelir. Bu nedenle genel olarak parçalanmayı en aza indirmek isteriz.

Parçalanmada genellikle boşa ayrılan bellek yüzde olarak rapor edilir. Örneğin bir programda 4K kod, 1K global, 8K yığın ve 1K yığıt segmenti olduğunu varsayalım. Ayrıca, malloc serbest listesinde 6K olduğunu varsayalım. Bu durumda programdaki parçalanma yüzdesi 6 / (4 +1 +8 +1) = 43% olarak hesaplanır. Bu programın toplam bellek kullanımı 14K olduğundan bu kayıp çok da kötü değildir. Ama bunun gibi 1000 program makinede aynı anda çalıştığında veya program boyutu 14M olduğunda parçalanma oldukça fazla israfa yol açacaktır.

İç ve dış parçalanma olmak üzere 2 çeşit parçalanma vardır. Dış parçalanma serbest listenin düğümlerinde kullanılmayan bellek nedeniyle olan parçalanmaya denir. Yukarıdaki örnekte kullanılmayan 6K bellek dış parçalanmadır.

İç parçalanma biraz daha zekice yapılmış .İlgili hafıza altyapı için ayrılır kullanıcı için ayrılmaz.Örneğin malloc() un genelde muhasebe için ayırdığı 8 byte ı iç parçalanmaya katkıda bulunur.Bunun neden böyle olduğunu aşağıdaki kod parçasında görelim.

```
char *array[100000];

main()
{
  int i;

  for (i = 0; i < 100000; i++) array[i] = (char *) malloc(8);
}
```

Programcı bu kodun 8 bytelık miktarlarda 800K lar ayırdığını umar.Ancak muhasebe için tahsis edilen extra 8 byte tan bu yana 1.6 M yer tahsis edilir. Bu iç parçalanmanın %50 si heap altyapısıdır ve belkide programcının gözünde atık olarak kabul edilir.

İç parçalanma için başka bir örnek te, yukardaki örnekte kullanıcının malloc(8)i çağırır ve bizde 16 byte lık pointer döndürdük çünkü bizim 24 byt lık yığınımız 2 bytlık parçalara ayrılamayan free list in içinde .Bu bir parçalanmadır çünkü hafıza gerçekten boşa harcanmıştır.

İç ve dış parçalanma arasındaki fark çok incedir ama bir ayrım olduğunu görürsünüz.

## İÇ PARÇALANMAYLA MÜCADELE

İç parçalanmayla mücadele etmenin bir yolu birden çok free list i tutmaktır. Örneğin 4 bytlık free listin olsun. Bir tane 8 bytelık bir tane 16 bytelık ve 16 dan daha büyük bir tane olsun. 4 8 yada 16 bytlık free list tahsis ettiğinde free() için yığının kapladığı alanı depolamana gerek kalmaz ve böylece iç parçalanman olmaz. Geniş malloc() un iç parçalanması o kadar da sorun değildir. Hikayenin tamamı bu değil(bu konuda biraz düşünürseniz,free() nin çalışmasıyla ilgili sorularınız olabilir )fakat bunu kabataslak şekilde bırakacağım.Bazı sistemler sadece bu şekilde hafıza tahsis ederler.Bizim Unix versiyonumuz bu şekilde tahsis etmiyor.

### Freelist Ayırma Algoritması

Sebest listenizin şu şekilde göründüğünü varsayalım:

Heat ----> Chunk 1 ----> Chunk 2 ----> Chunk 3 ----> Chunk 4 ----> Chunk 5

40 bytes 24 bytes 72 bytes 16 bytes 8K bytes

Eğer siz malloc(8) isteği verirseniz bu yığınlardan gerekli 16 bytelık alan ayırabilirsiniz. (8 kullanıcı için, 8 de kayıt için). Peki hangisini seçebiliriz ? Bunu yapmak için birkaç standart algoritmalar mevcuttur. Bunlar :

1. ++-- bu yığınlardan Closest fit i vereni bul ve bunu seç
2. Worst fit -- daima en büyük yığından yer ayırır.
3. First fit -- baştan başla, ve yeterince büyük olan ilk yığından yer ayırır.
4. Next fit – ilk ayrılan yığından sonra başlar ve ilk yeterince büyük yeri seçer.(freelist dairesel gibi davranır.)
   Bu nedenle eğer malloc(60), malloc(32), malloc(8) gibi 3 istek verirsek yığınların ayrılması şu şekilde olmalıdır:

   1. Best fit: malloc(60): chunk 3, malloc32 : chunk 1,malloc8 : chunk 4
   2. Worst fit: malloc(60):chunk 5,malloc(32) : chunk 5(arta kalan),malloc(8) : chunk 5(arta kalan)
   3. First fit: malloc(60) : chunk 3,malloc(32) : chunk 1,malloc(8) : chunk 2
   4. Next fit: malloc(60) : chunk 3,malloc(32) : chunk 5,malloc(8) : chunk 5(arta kalan),yada chunk 1, bu sizin kalanı nasıl ele aldığınıza bağlıdır.

   First fit /Next fit üretmek en iyi performansı göstermeye yöneliktir—Best fit birçok iç ve dış parçalama hatalarıyla sonuçlanmaktadır ve Worst fit daha kötüdür. Tüm metotlar her malloc() ile tüm freelist i sorgulamayı gerektirir. (yada,en az derecede O(log n) liste eylemlerinde eğer freelisti yığın içerisinde tutarsanız) ki bu First ve Next fit e göre kesinlikle kötü bir performans verecektir.

   Siz bu ayırma algoritmalarından malloc() la istediğinizi uygulayabilirsiniz.