Designdokument

Beskrivning av datastrukturer:
I imalloc.h finns de datastrukturer som hade definierats.

Memory är en pekare till en union av managed- och manual-strukturerna. I managed och manual finns det funktionspekare till de olika funktionerna som ska kunna användas när iMalloc anropas med managed eller manual som indata. I managed ligger också strukturerna Refcount och GC som bara initialiseras om de angivits i anropet av iMalloc.

Strukturerna Refcount och GC är definierade som anonyma eftersom de bara ska kunna nås från managed-strukturen. I dessa finns funktionspekare till de funktioner som ska kunna användas genom anrop till respektive struktur.

I priv_imalloc.h introducerar vi ytterligare två datastrukturer som heter chunk och list.

Strukturen chunk innehåller all metadata om det utrymme som användaren begär av funktionen alloc enligt följade:

Refcount: Representerar hur många variabler som refererar chunk-datat.

Data: Pekar på det utrymme som användaren får tillbaka från anropet av alloc(och har rätt att använda hur som helst).

Size: Representerar storleken av data utrymmet, d.v.s. hur mycket utrymme som begärdes av alloc.

Next: Pekar på nästa chunk i listan.

Prev: Pekar på föregående chunk i listan.

Mark: Talar om i fall minnesutrymmet chunken beskriver är upptaged(1) eller ledigt(0).

Strukturen list representerar en lista av chunks.

First: Pekar på första chunken i listan.

Current: Är en pekare som kan iterera över chunksen i listan.

L*ast*: Pekar på sista chunken i listan.

Sort: Beskriver hur listan är sorterad(1=ascending, 2=descending, 4=address)

tSize: Representerar totala utrymmet användaren allokerat med iMalloc.

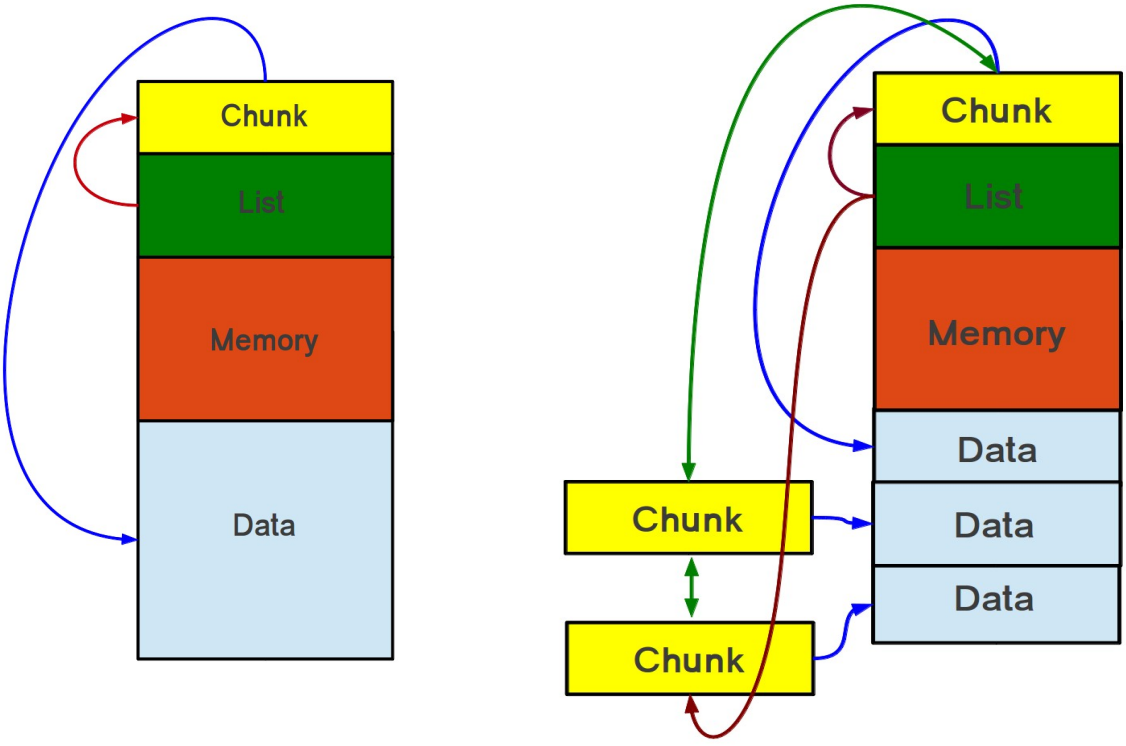
Övergripande design:
När iMalloc körs skapar vi en chunk med pekare till datat av storleken som angivits av indata till iMalloc. Det betyder att minne allokeras för både storleken av strukturen chunk samt storleken användaren skickade som indata.

Chunk-strukturen måste också tas i åtanke när man allokerar eftersom metadatat om minnet inte ska göra att användaren av iMalloc får mindre minne att arbeta med än vad han begärde. Utöver chunk-strukturen så allokeras också minne för list-strukturen och manual- eller managed-strukturen(beroende på indata).

När iMalloc körs skapas en virtuell heap samt utrymme och informationen om både heapen och vilka funktioner som användaren kan utföra. Med virtuell heap menas att vi allokerar minne från den riktiga heapen(med malloc) och ger användaren illusionen att han allokerar minne från en iMalloc-heap med funktionen alloc, även fast allt data ligger på den riktiga heapen.

Chunksen lagras som en länkad lista som kan nås från strukturen list. Listan sorteras beroende på indatat till iMalloc och både fria och upptagna chunks ligger i samma lista. En fri eller upptagen chunk känns igen på om dens mark bit är 0 eller 1.

Pekaren iMalloc returnerar är en pekare till de funktioner som är tillgängliga på den heap som skapades.



Bilderna ovan är en grafiskrepresentation av hur datat och strukturerna allokeras på heapen. Den första chunken skapas ”överst”, därefter kommer list, sedan memory (managed eller manual) dvs det som returneras till användaren och ”sist” kommer datat ligga (data av storleken som användaren fråga efter).
När nya chunks skapas så placeras de på heapen med malloc och deras data pekare kommer peka på korrekt del avdet redan allokerade datat.

iMallocFree
Vi insåg att om man kör free av det som iMalloc returnerar så kommer det bli klar en massa alokeringar på stacken, detta löste vi med en funktion iMallocFree som frigör allt allokerat utrymme.

Bugg rapport
Vi har hittat en bugg som vi ej har hunnit lösa. När vi traverserar stacken så hittar vi gamla pekare som ej används längre och dessa kan av slump peka in på det data som användaren allokerat.
Detta kan resultera i att chunks som collect skall frigöra ej frigörs.
Vi har ett test som skapar ett antal väldigt stora chunks (så sannorlikheten att det blir fel är större), om detta test passerar så är buggen närvarande.

Ful-lösning
Vi använde oss av en global variabel ”globalpList” som är en pekare till listan av chunks. Detta gjorde vi för att i funktionerna retain och count i refcount endast tar en void-pekare som indata. Vi kom inte på något sätt att hitta den chunken som har metadatat om void-pekaren med bara void-pekaren som utgångspunkt.

Det blev dock trivialt när vi hade tillgång till listan eftersom då kunde vi jämföra varje chunks void-pekaren med den som skickades in till retain- eller count-funktionen för att hitta vilken chunks refcount vi ska arbeta med.

Denna variabel används bara i retain och count. En lösning vore att sätta in en Memory som argument i de två funktionerna, detta var dock ej fallet i imalloc.h.