Heap Chunk 구조 - 서연주

1. chunk란

실제로 할당/반환받게 되는 영역으로, 8 bytes의 배수로 할당됨

커다란 heap을 다양한 사이즈의 chunk로 나눠 할당

하나의 chunk는 하나의 heap 내부에 존재하며, 하나의 arena에 속함

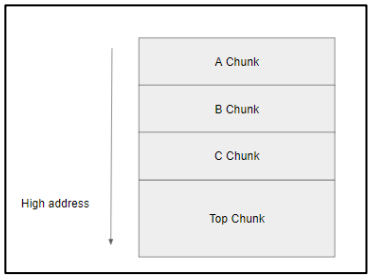
2. 할당된 chunk 구조 파악

a=malloc(8);

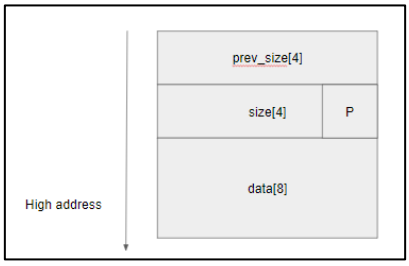
b=malloc(8);

c=malloc(8);

(물론 낮은 주소에서 높은 주소로 쌓임)

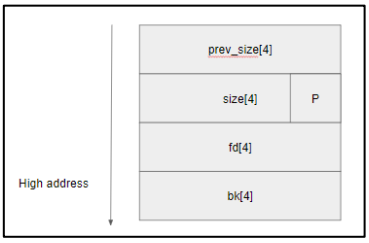


할당받은 각 chunk는 오른쪽 그림과 같은 구조



- prev_size : 이전 chunk의 크기를 나타냄,
  이전 chunk가 free되었을 때 설정
- size : 현재 chunk의 크기를 나타냄
  malloc() 시에 설정, 하위 3비트는 플래그 용도
  - prev_inuse 플래그 : 이전 chunk가 사용중인지 아닌지를 판별 (1, 0)
  - is_mmapped 플래그 : 해당 필드가 mmap() 시스템 콜을 통해 할당된 것인지를 나타냄
    (mmap()을 통해 할당된 chunk는 다른 방식으로 관리됨)
  - non_main_arena 플래그 : 각 thread마다 다른 heap 영역을 사용하는 경우
    현재 chunk가 main heap에 속하는지 여부를 판별
- data : 입력한 값이 저장되는 영역 (AAAAA가 들어가는 영역)

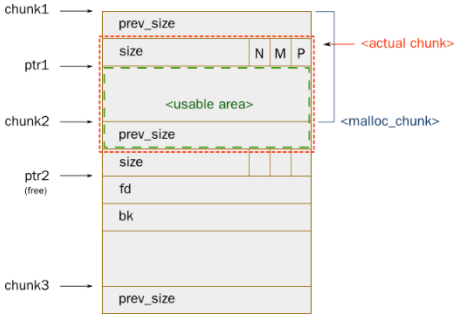
3. 해제된 chunk 구조 파악



free된 chunk의 구조는 오른쪽과 같음

할당되었을 때와 다른 점은 data부분

- fd(forward pointer) : 아직 사용되지 않은 다음 chunk의 주소
- bk(backward pointer) : 아직 사용되지 않은 이전 chunk의 주소



사실 다음 chunk의 prev_size 필드도 현재 chunk의 데이터 영역으로 사용

(플래그들은 제외)

또한 free시에는 다음 chunk의 prev_inuse플래그를 지워야하고, prev_size

필드는 오직 p플래그가 지워진 상태에서만 사용 (free됐을때만 사용)

4. Bins

: heap은 영역을 할당하고 해제할 때, 메모리를 좀 더 효율적으로 사용하기 위해 bin이라는 구조를 사용하여 해제된 chunk list를 관리 → bin 구조

-free된 chunk의 필드인 fd, bk를 이용하여 리스트를 연결하고 있음

-chunk의 크기를 기준으로 사용하는 bin이 달라짐 : fast bin, unsorted bin, small bin, large bin

(fastbinsY 배열 : fast bin 수용, bins 배열 : unsorted, small, large bin 수용)

/*5. Fast bin

chunk의 크기가 16~80 byte인 경우 → fast chunk → fast chunk를 수용한 bin == fast bin

-모든 bin들 가운데, fast bin은 메모리 할당과 해제가 빠름

-bin의 개수는 10개, 각 fast bin은 단일 연결리스트를 가짐

-chunk의 할당과 해제는 list의 앞에서 발생

ex) */

5. 코드 분석 (참고 : https://s0ngsari.tistory.com/entry/Heap-with-GDB)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


int main(int argc, char *argv[]){
    char* buf = (char*)malloc(256);
    char* buf1 = (char*)malloc(512);
    strcpy(buf, argv[1]);
    strcpy(buf1, argv[2]);
    printf("%s\n", buf);
    free(buf);
    free(buf1);
}
```

1. buf를 256, buf1은 512만큼
2. chunk에 strcpy
3. buf와 buf1를 차례로 free

1) gdb분석

```
Dump of assembler code for function main:
   0x080484ad <+0>:     push   %ebp
   0x080484ae <+1>:     mov    %esp,%ebp
   0x080484b0 <+3>:     and    $0xfffffff0,%esp
   0x080484b3 <+6>:     sub    $0x20,%esp
   0x080484b6 <+9>:     movl   $0x100,(%esp)
   0x080484bd <+16>:    call   0x8048370 <malloc@plt>
   0x080484c2 <+21>:    mov    %eax,0x18(%esp)
   0x080484c6 <+25>:    movl   $0x200,(%esp)
   0x080484cd <+32>:    call   0x8048370 <malloc@plt>
   0x080484d2 <+37>:    mov    %eax,0x1c(%esp)
   0x080484d6 <+41>:    mov    0xc(%ebp),%eax
   0x080484d9 <+44>:    add    $0x4,%eax
   0x080484dc <+47>:    mov    (%eax),%eax
   0x080484de <+49>:    mov    %eax,0x4(%esp)
   0x080484e2 <+53>:    mov    0x18(%esp),%eax
   0x080484e6 <+57>:    mov    %eax,(%esp)
   0x080484e9 <+60>:    call   0x8048360 <strcpy@plt>
   0x080484ee <+65>:    mov    0xc(%ebp),%eax
   0x080484f1 <+68>:    add    $0x8,%eax
   0x080484f4 <+71>:    mov    (%eax),%eax
   0x080484f6 <+73>:    mov    %eax,0x4(%esp)
   0x080484fa <+77>:    mov    0x1c(%esp),%eax
```

main+15 : 0x100 = 256만큼 잘 할당

main+25 : 0x200 = 512만큼 잘 할당

2) 첫번째 브포

```
(gdb) b *0x080484c2
Breakpoint 1 at 0x80484c2
```

두번째 malloc 전까지 브포를 걸어서 확인

3) 첫번째 chunk구조

```
(gdb) x/100x $eax-8
0x804b000:      0x00000000      0x00000109      0x00000000      0x00000000
0x804b010:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b020:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b070:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b080:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b090:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0a0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b100:      0x00000000      0x00000000      0x00000000      0x00020ef9
```

0x804b000 : prev_size

0x804b004 : size of chunk (0x109 = 265 (prev_size[4] + size[4]+ prev_inuse[1] + data[256] = 265))

0x804b008 : user data

0x804b10c : top chunk모든 chunk의 마지막엔 top chunk가 위치

*

모든 chunk의 마지막엔 top chunk가 위치

어떠한 bin에도 속하지 않으며 항상 heap영역 마지막에 위치

다음 malloc할 때 새로운 chunk size를 top chunk에서 할당하고 나머지는 다시 top chunk가 됨

4) 두번째 브포

main+37에 브포 (두번째 malloc 할당 완료)

5) 두번째 chunk 구조

```
0x804b000:      0x00000000      0x00000109      0x00000000      0x00000000
0x804b010:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b020:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b070:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b080:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b090:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0a0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b100:      0x00000000      0x00000000      0x00000000      0x00000209
```

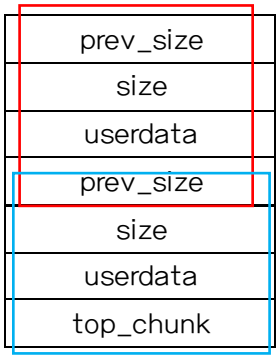
원래 top chunk였던 부분인 0x804b10c이 size of chunk가 됨

0x209 = 521 (prev_size[4] + size[4]+ prev_inuse[1] + data[512] = 521)

prev_size의 부분인 0x804b108은 앞의 chunk에서 data영역으로 사용

top chunk는 0x804b000 + 268 + 520 (아직 이해 X) = 0x804b314

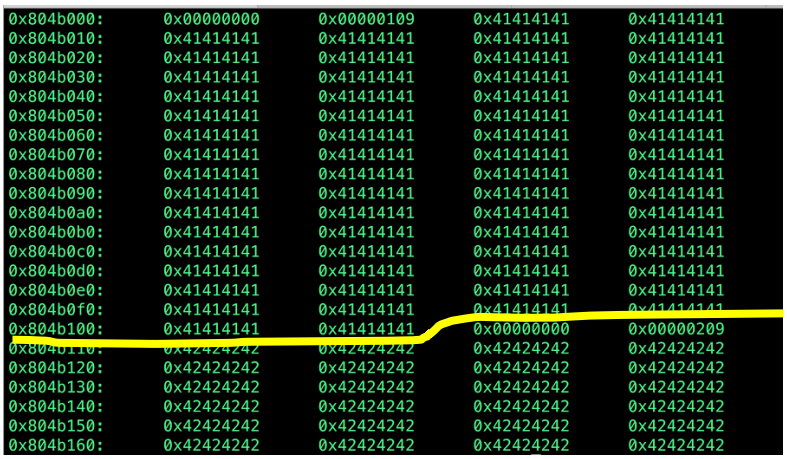
6) 지금까지의 구조



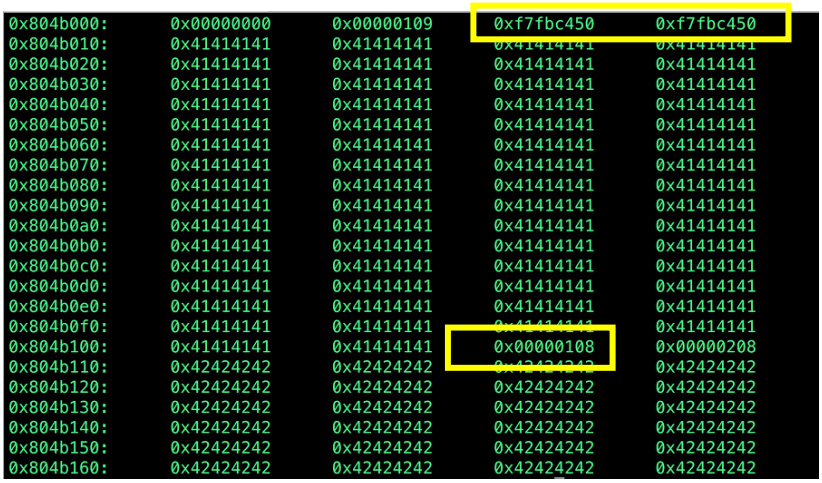
빨간색 테두리 → buf의 chunk

파란색 테두리 → buf1의 chunk

7) data를 strcpy 이후 구조



8) 첫번째 free 후 구조



0x804b008에 이상한 값 : free될 때의 fd(forward pointer)와 bk(backward pointer)

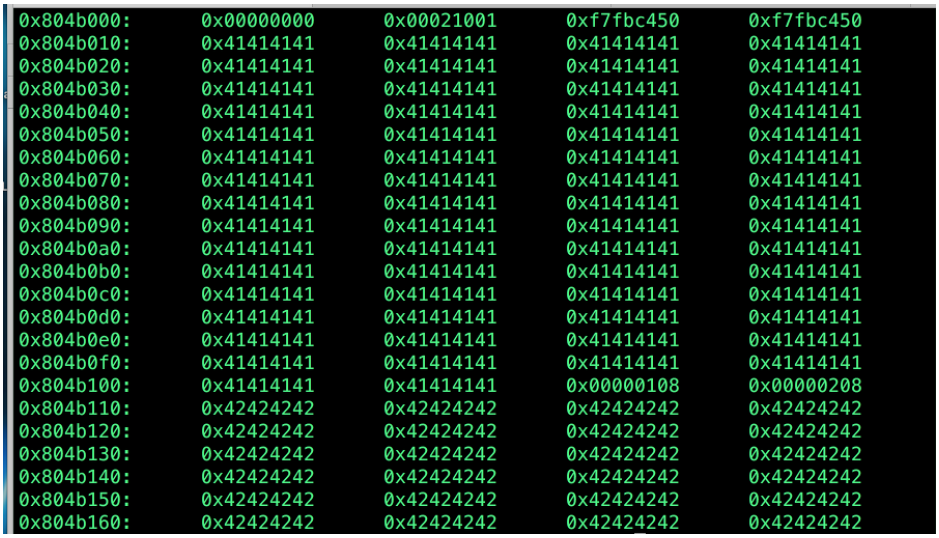
fd → next chunk, bk→prev chunk를 가리킴

free 한 번 후 topchunk 바로 앞을 가리킴

```
(gdb) x/x *0xf7fbc450+4
0x804b314: 0x00020cf1
```

첫번째 chunk를 해제하니 두번째 chunk의 prev_size가 바뀜

9) 두번째 free 후 구조 (main+125에 브포)



```
0x804b000:      0x00000000      0x00021001      0xf7fbc450      0xf7fbc450
0x804b010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b020:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b030:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b040:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b050:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b060:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b070:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b080:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b090:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b0a0:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b0b0:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b0c0:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b0d0:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b0e0:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b0f0:      0x41414141      0x41414141      0x41414141      0x41414141
0x804b100:      0x41414141      0x41414141      0x00000108      0x00000208
0x804b110:      0x42424242      0x42424242      0x42424242      0x42424242
0x804b120:      0x42424242      0x42424242      0x42424242      0x42424242
0x804b130:      0x42424242      0x42424242      0x42424242      0x42424242
0x804b140:      0x42424242      0x42424242      0x42424242      0x42424242
0x804b150:      0x42424242      0x42424242      0x42424242      0x42424242
0x804b160:      0x42424242      0x42424242      0x42424242      0x42424242
```

* 사이즈 128이하로 할당하면 fastbin에 속하게 됨, fastbin에 속하게 되면 free했을때 fd와 bk가 생성되지 않음 (이건 더 이해 필요…)