chunk의 뜻

- arena가 관리하는 힙 영역을 구성하고 있는 사용되거나 해제되었거나, 미사용한 힙 영역.
- malloc()으로 할당 / 반환받게 되는 영역으로 32bit 환경에서는 8bytes의 배수로, 64bit 환경에서는 16bytes의 배수로 할당됨
- 각 chunk는 size가 얼마인지, 인접한 chunk의 위치는 어디인지를 나타내는 meta-data를 포함함.
- malloc()이 반환하는 주소는 chunk의 시작 지점이 아니라 data이기 때문에, 반환되는 위치부터 바로 포인터로 접근해서 쓸 수 있음.
- chunk 구조체

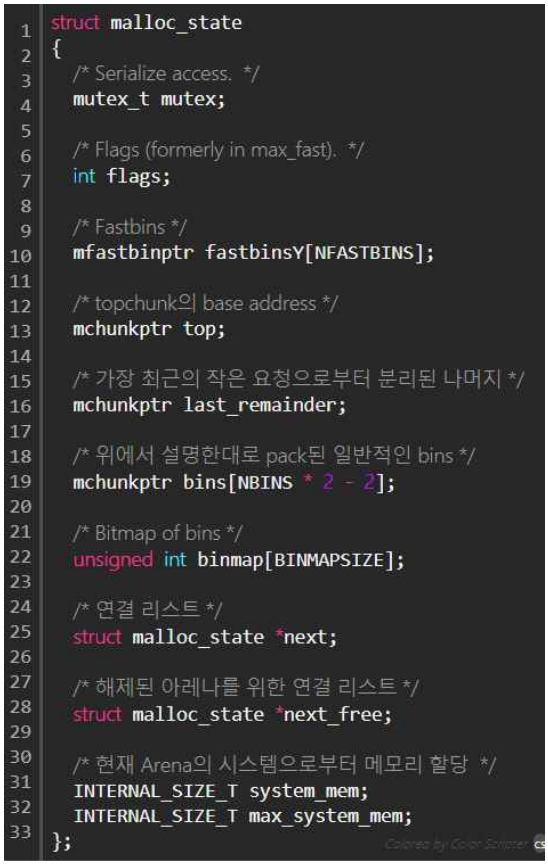
```c
struct malloc_chunk {

  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* large block에서만 사용하고 해당 bin list의 크기 순서를 나타냄 */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

Arena

- main을 포함한 thread에 대한 힙 영역.
- 힙 영역의 공간이 부족하면 새로운 영역에 추가로 할당받기 때문에 여러 개의 힙 영역을 가질 수 있음. (단, 모든 thread가 각자의 arena를 가지지는 못함.)
- > 그렇기 때문에 힙 영역을 어떤 arena가 관리하고 있는지, 힙 영역의 크기가 어느 정도인지, 이전에 사용하던 heap 영역의 정보가 어디에 있는지를 저장해야 함. => 이를 저장하기 위한 구조체가 malloc_info 구조체.
- malloc_info 구조체

```c
typedef struct _heap_info {
  mstate ar_ptr;     /* 현재 heap을 담당하고 있는 Arena */
  struct _heap_info *prev;  /* 이전 heap 영역 */
  size_t size;       /* 현재 size(bytes) */
  size_t mprotect_size; /* mprotected(PROT_READ|PROT_WRITE) size(bytes) */
  char pad[(-6*SIZE_SZ)&MALLOC_ALIGN_MASK]; /* 메모리 정렬 */
  /* sizeof(heap_info) + 2*SIZE_SZ는 MALLOC_ALIGN_MASK의 배수 */
} heap_info;
```

- 힙 영역에서도 어떤 부분을 사용하면 되는지에 대해 알고 있어야하기 때문에, malloc_state 구조체는 각 arena에 하나씩 주어지고, 해제된 bin, top chunk와 같은 arena에 대한 정보를 저장함.
- malloc_state 구조체

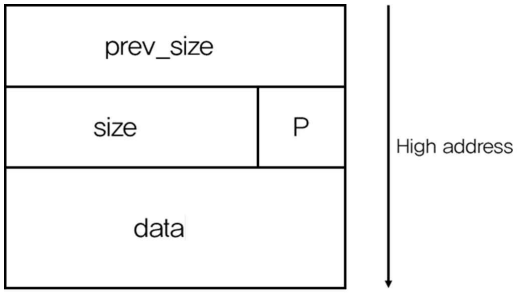


```
struct malloc_state
{
  /* Serialize access. */
  mutex_t mutex;

  /* Flags (formerly in max_fast). */
  int flags;

  /* Fastbins */
  mfastbinptr fastbinsY[NFASTBINS];

  /* topchunk의 base address */
  mchunkptr top;

  /* 가장 최근의 작은 요청으로부터 분리된 나머지 */
  mchunkptr last_remainder;

  /* 위에서 설명한대로 pack된 일반적인 bins */
  mchunkptr bins[NBINS * 2 - 2];

  /* Bitmap of bins */
  unsigned int binmap[BINMAPSIZE];

  /* 연결 리스트 */
  struct malloc_state *next;

  /* 해제된 아레나를 위한 연결 리스트 */
  struct malloc_state *next_free;

  /* 현재 Arena의 시스템으로부터 메모리 할당 */
  INTERNAL_SIZE_T system_mem;
  INTERNAL_SIZE_T max_system_mem;
};
```

- main_arena는 application's initial heap만 사용함.
- 다른 arena는 mmapped heap을 사용함.
- main_arena를 제외한 arena는 additional arena를 가리키는 next pointer를 가지고 있음.

* 32bit 환경에서의 구조 조사함

chunk의 구조

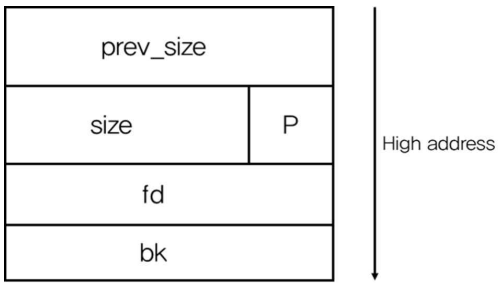


- prev_size 필드 : flag를 제외한 이전 chunk의 크기. 이전 chunk가 free되었을 때 설정됨
- > 이전 chunk와 merge하는 경우에 유용하게 사용

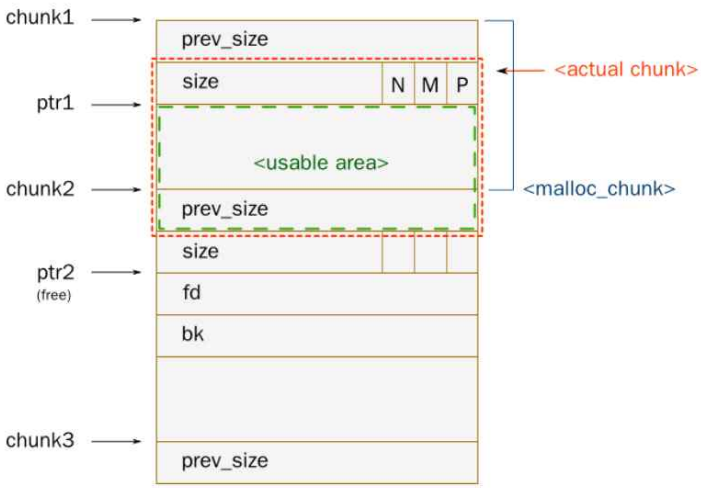
- size 필드 : 현재 chunk의 크기. malloc() 시에 설정되고, 하위 3비트는 특별한 용도(flag)로 사용됨.
- > prev_inuse 플래그 : 이전 chunk가 사용 중인지 아닌지를 판별하는 플래그로, 이전 chunk가 사용 중이거나 fastbins에 들어가 있는 chunk일 경우 1, free되어 있는 경우 0으로 설정됨. (bin에 대해서는 뒤에서 설명)
- > is_mmapped 플래그 : 해당 필드가 mmap() 시스템 콜을 통해 할당된 것인지를 나타내는 플래그로 chunk 자체가 단일 mmap() 호출로 할당된 영역 전체일 경우 1, 어떤 heap의 일부일 경우 0으로 설정됨. (이를 표시해주는 이유는 mmap()으로 할당된 chunk는 다른 방식으로 관리되기 때문임)
- > non_main_arena 플래그 : multi thread application에서 각 thread마다 다른 heap 영역을 사용하는 경우 현재 chunk가 main heap(arena)에 속하는지 여부를 나타내는 플래그로, 이 chunk가 mmapped memory에서 왔을 경우 1, main arena와 main heap에서 왔을 경우 0으로 설정됨.

- data : 입력한 값이 저장되는 영역

free된 chunk 구조



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



- malloc()을 통해 3개의 chunk를 할당받았고, 두 번째 chunk를 free했다고 했을 때.
- malloc()을 통해 할당된 chunk는 prev_size, size, data 필드로 구성되는데, 여기에선 다음 chunk (chunk 2)의 prev_size 필드가 현재 chunk (chunk 1)의 데이터 영역으로 사용됨
- > 개념적으로는 다음 chunk의 prev_size 필드도 현재 chunk에 속하며, free() 시에는 현재 chunk의 크기를 중복하여 저장하는 용도로 사용됨. (플래그 제외) => boundary tag 기법
- > free() 시에는 다음 chunk의 prev_inuse비트 (P 플래그)를 지워야하고(0으로 함), prev_size 필드는 오직 P 플래그가 지워진 경우에만 사용되어야 함.

- 할당된 chunk들을 이후에 free()하게 되면, bin이라는 구조를 통해 관리됨. (뒤에서 설명)

bin의 뜻

- arena에 있는 free된 chunk들을 빠르게 재할당할 수 있도록 free된 chunk를 종류별로 가리키는 포인터의 리스트임.
- 각 chunk의 fd, bk 필드로 연결된 doubly-linked list이고, chunk의 크기에 따라 총 126개로 분리됨. (첫 번째 bin은 특별한 용도로 사용됨.)

bin의 종류와 그 쓰임

① fast bin

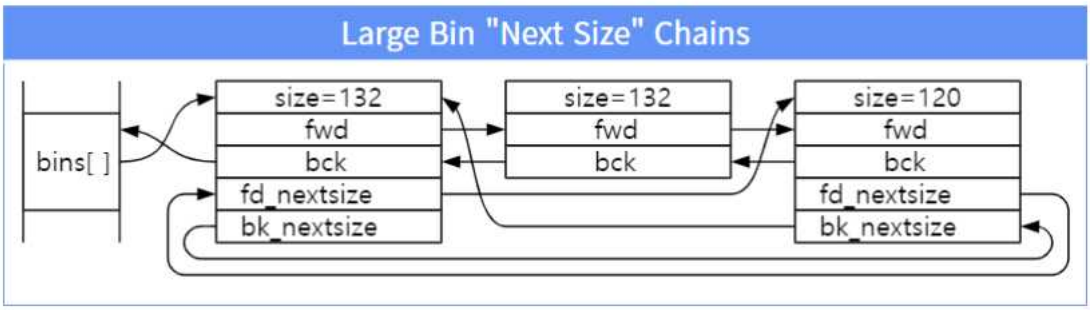
- chunk 크기 기준으로 80 바이트 미만의 크기를 가지는 chunk는 fast bin을 통해 관리함.
- malloc() 및 free() 시에 가장 먼저 조사하는 bin.
- 속도를 높이기 위해서 singly-linked list로 구성되며 LIFO와 같은 방식으로 동작함.
- fast bin 내의 chunk들은 병합이 일어나지 않는 한 계속 bin 내에 남아서 요청을 수행할 수 있음.

② small bin

- chunk 크기 기준으로 512 바이트 미만인 것들이 8바이트 단위로 구분되는데, 최소 크기인 16바이트부터, 504바이트까지 총 62개의 bin으로 구성됨.
- 즉, fast bin 포함!
- > chunk의 최소 크기 제한으로 인해 0, 8에 해당하는 bin이 존재하지 않아서 62개
- fast bin과의 차이1 : P flag가 없어서 chunk가 일단 small bin에 연결되면 인접 chunk와 합치려고 해서 인접 chunk가 없음.
- > 단, fast, unsorted, in-use chunk와 인접해있는 경우에는 결합하지 않기 때문에 이 경우에는 인접 chunk가 있을 수 있음.
- fast bin과의 차이2 : circular doubly linked list 구조임.

③ large bin

- 512 바이트 이상의 크기를 가진 chunk들을 위한 것인데, small bin처럼 동일한 크기의 chunk만을 포함하는 것이 아니라 해당 index가 나타내는 크기보다 작은 크기의 chunk들을 모두 포함한다.
- > ex) 4KB를 위한 bin이 있다고 할 때, 정확히 4096 바이트의 크기의 chunk만을 포함하는 것이 아니라, 4088, 3968 등의 크기를 가지는 chunk들도 포함함.
- 다만 할당의 효율성을 위해서 해당 bin 내에서는 크기 별로 정렬됨.
- > 이 때 fd_nextsize와 bk_nextsize 필드가 이용되며 (추가적인 circular doubly-linked list), 이들은 현재 bin 내에서 크기가 다른 첫 번째 chunk에 대한 포인터를 저장함.
- > 요청 사이즈를 만족하는 chunk가 여러 개 있을 경우, 보통 두 번째가 선택되는데, 첫 번째 chunk를 선택할 경우 **_nextsize 포인터를 다시 이어야하기 때문임.



④ unsorted bin

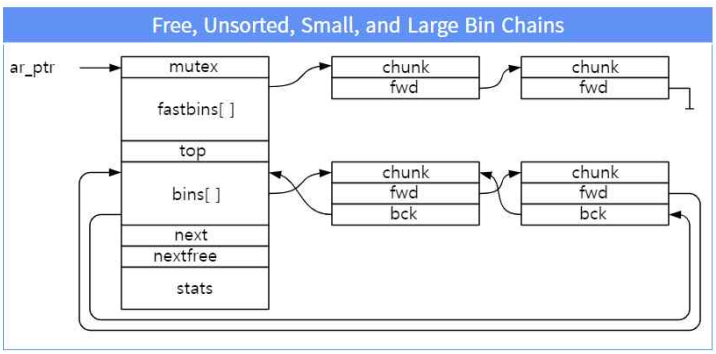
- 첫 번째 bin은 unsorted chunk의 list로 사용됨.
- free()된 chunk는 곧바로 해당 bin에 들어가지 않고 먼저 unsorted chunk list에 들어가며 이후의 메모리 할당 시 동일한 크기의 영역을 다시 요청하는 경우에는 이를 바로 재사용하도록 함.
- FIFO와 같은 방식으로 동작하며, 할당되거나 해당 bin으로 돌아가게 됨.
- 단 한 번의 재사용 기회가 주어짐
- ex) small bin chunk가 unsorted bin에 존재하는 상태에서 또 다른 small bin chunk가 free되는 경우 이를 계속 unsorted bin에 추가하다가 fastbin과 같은 다른 size의 chunk가 free될 때 small bin으로 옮김.
- > 다른 size여도 small bin으로 분류된다면, 그냥 unsorted bin에 추가함.

● 이러한 bin들은 bin 내에 이용 가능한 free chunk가 있는지 빨리 조사하기 위해서 별도의 bitmap을 유지하여 관리함. 해당 bin 내에 free chunk가 없다면 그보다 큰 bin 내의 가장 작은 chunk를 빨리 찾기 위해 이용할 수 있음.

⑤ 그 외

- 128KB 이상의 큰 메모리를 요청하는 경우에는 heap을 이용하지 않고, mmap() 시스템 콜을 통해 별도의 영역을 할당하여 chunk를 만들고, 이를 사용자에게 반환하며, 이러한 chunk들은 bin내에 속하지 않음. 이러한 chunk들은 is_mmapped 플래그로 쉽게 확인할 수 있기 때문에 free() 시에 단순히 munmap()을 호출하여 메모리 영역을 해지함.

● bin 리스트



- fast bins[]는 LIFO이기 때문에, 맨 왼쪽 chunk가 먼저 반환되고, bins[]는 FIFO라서 맨 오른쪽 chunk가 먼저 반환됨. bins[].bk의 chunk가 다음 반환 chunk.

⑥ top chunk
- arena의 가장 꼭대기에 있는 chunk.
- 어떤 bin에도 속하지 않음. heap 영역의 마지막에 위치.
- 사용자가 요청한 크기가 top chunk보다 작은 경우 top chunk는 user chunk(사용자가 요청한 크기)와 Remainder chunk(top chunk에서 사용자가 요청한 크기를 뺀 나머지 크기)로 분리됨. 여기에서 remainder chunk는 새로운 top chunk가 됨.
- 다른 free chunk들이 메모리 할당 요청을 만족하지 못하는 경우에만 top chunk를 분할하여 요청을 처리함.
- 사용자가 요청한 크기가 현재 top chunk보다 큰 경우에는 sbrk(main arena) 또는 mmap(thread arena) syscall을 사용하여 top chunk를 확장시킨 후 할당.

출처

https://tribal1012.tistory.com/141
https://umbum.tistory.com/386
http://egloos.zum.com/studyfoss/v/5206220
https://mintnlatte.tistory.com/357