

Double Free Bug

하계 Heap-찢이 1팀

구성

Heap 기초와 DFB 발생 포인트


Double Free Bug

Heap에 대해서

- 메모리가 동적 할당될 때 이곳에 적재된다.
- 동적 할당은 런타임에 정해지기 때문에 크기가 유동적이다.
- 메모리를 사용한 후에 free 함수로 해제 해준다.

Heap에 대해서

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     char* str;
7     str = malloc(20);
8     free(str);
9 }
```

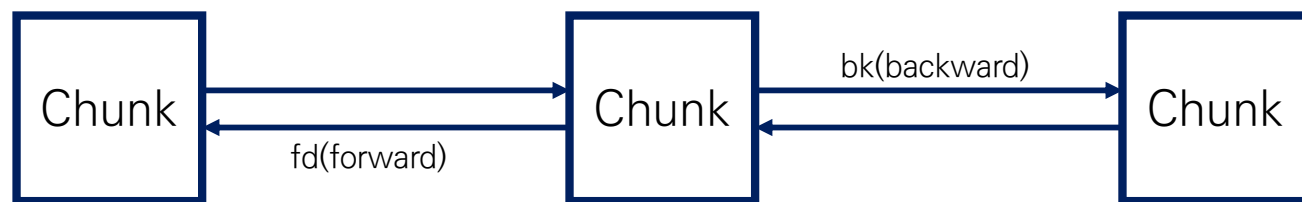


```
1 struct malloc_chunk {
2
3     INTERNAL_SIZE_T mchunk_prev_size; /* Size of previous chunk (if free). */
4     INTERNAL_SIZE_T mchunk_size;      /* Size in bytes, including overhead. */
5
6     struct malloc_chunk* fd;           /* double links -- used only if free. */
7     struct malloc_chunk* bk;
8
9     /* Only used for large blocks: pointer to next larger size. */
10    struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
11    struct malloc_chunk* bk_nextsize;
12 };
13 _
```

메모리를 동적으로 할당하면 chunk라는 자료구조로 만들어서 heap에 추가시킨다.

우측에 보이는 chunk의 구조체를 보면, size 이외에도 부수적인 metadata가 섞여서 할당 되는 것을 볼 수 있다.

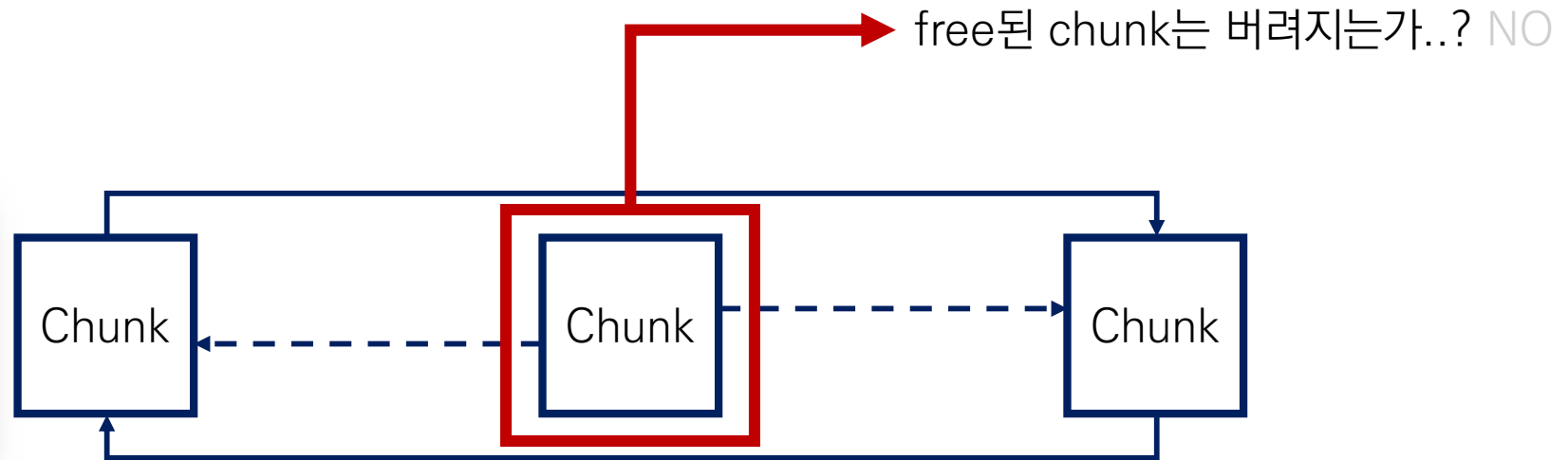
Heap에 대해서



이렇게 만들어진 chunk들은 본인의 앞, 뒤 chunk의 정보를 가진 노드들로 작용을 해서 doubly linked list를 이룬다.
(본래 heap이라는 자료구조 자체가 이진트리 형태를 띄고 있음)

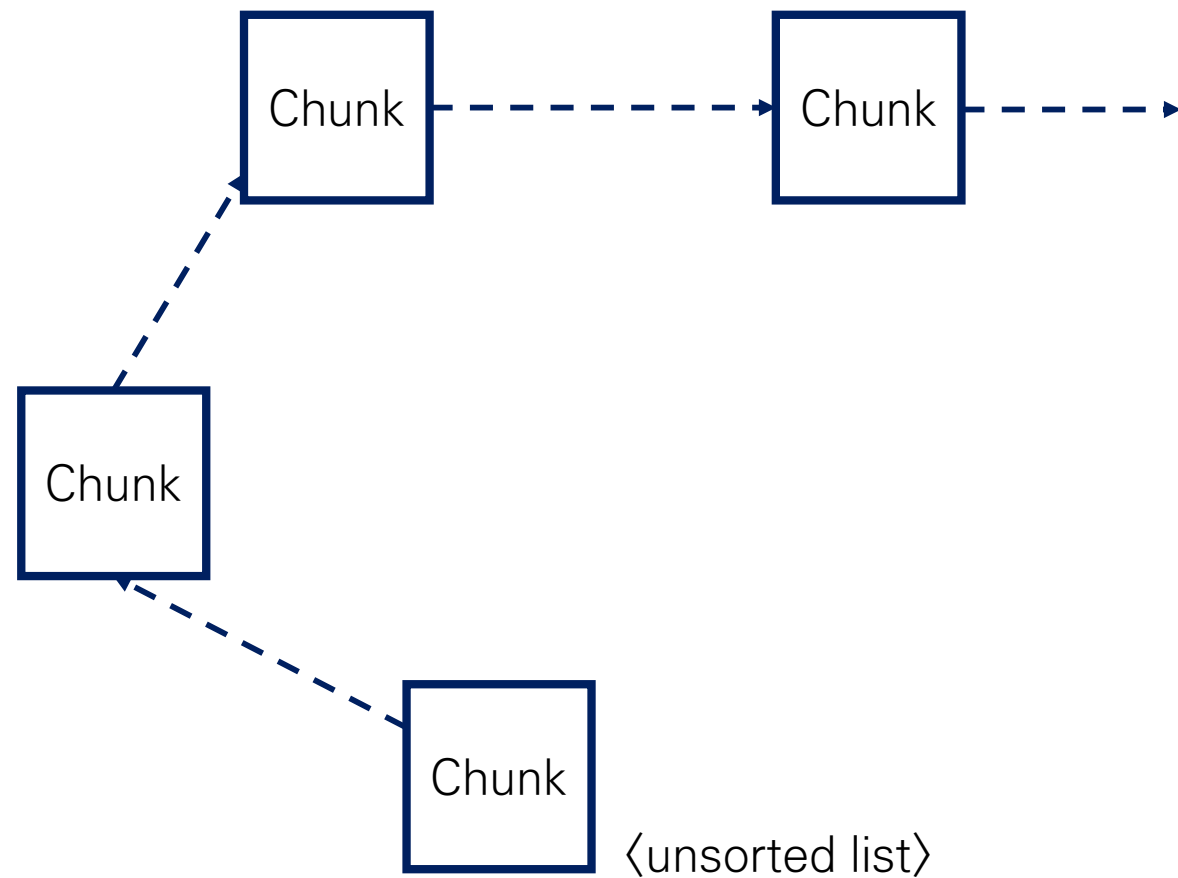
Heap에 대해서

```
1 #define unlink(P, BK, FD)
2 {
3     BK = P->bk;
4     FD = P->fd;
5     FD->bk = BK;
6     BK->fd = FD;
7 }
```



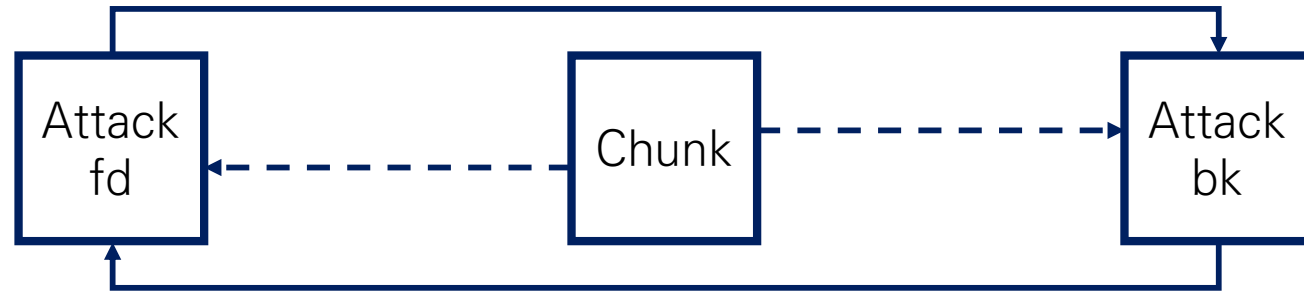
사용이 끝난 chunk에 대해서는 해당 chunk의 포인터를 free로 전달하여 양 옆의 링크를 끊는다. (unlink)
($fd+12=bk$, $bk+8=fd$)

Heap에 대해서



free된 애들은 안 버리고 bin이라는 자료구조에 차곡차곡 넣어 놔다가 똑같은 크기의 chunk가 필요할 때 재활용(환경 사랑..)

Double Free Bug



Heap overflow 등으로 fd와 bk를 조작한다면..?

원하는 주소에 값을 쓸 수 있을 것...

-> 프로그램의 흐름을 제어, 데이터의 손상 등등 가능
($bk = fd + 12$, $fd = bk + 8$)