

任意地址任意写

首先申请1个堆块p0，并伪造一个与p0相邻的空闲堆块fake，
在fake->presize填入偶数，fake->size填入-4，
fake->fd填入A-12，fake->bk填入B，
当释放p0时触发unlink(fake)过程，就能实现在任意地址A写入任意值B

chunk 0

free(p0)

由chunk0 + 256找到chunk0下一个堆块fake，通过查看fake状态判断是否进行合并

fake

256 + 1

...

256

-4

A-12

B

un_size + 1

unallocated

p0

break



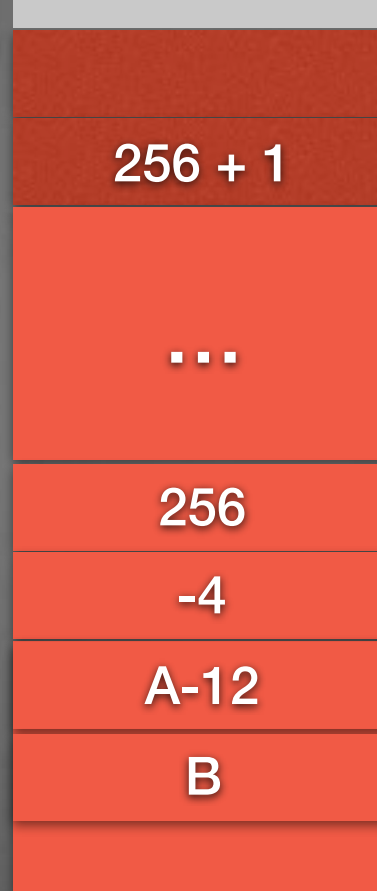
chunk 0

free(p0)

由于fake的size为-4,堆管理器将fake-4视为fake的下一个相邻堆块, 检查发现其size的标记位为0, 于是认为fake为空闲堆块, 触发空闲堆块合并

fake

fake-4



p0

break

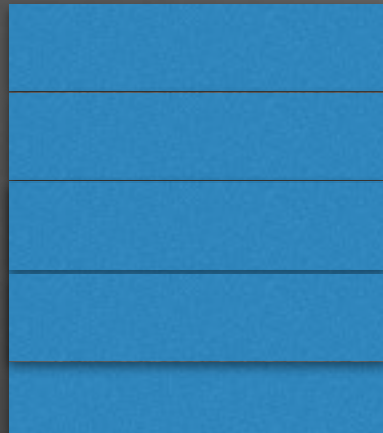
un_size + 1

unallocated

chunk 0

unlink(fake)

B



fake

256 + 1

...

256

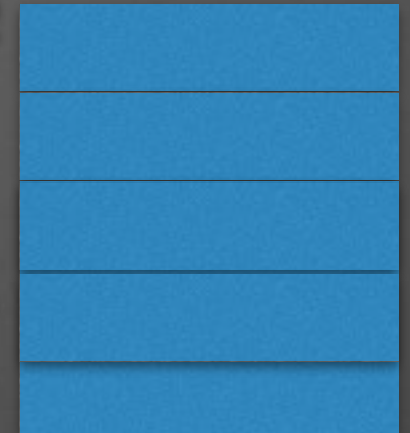
-4

A-12

B

A-12

A



p0

break

un_size + 1

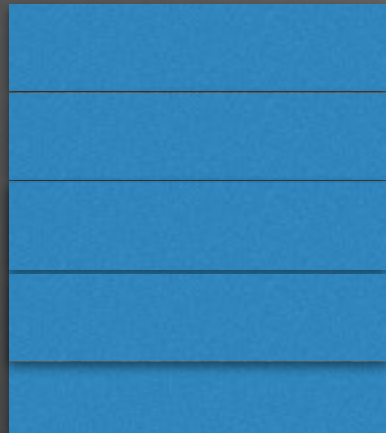
unallocated

```
#define unlink(P) {  
    FD = P->fd;  
    BK = P->bk;  
    FD->bk = BK;  
    BK->fd = FD;  
    ...  
}
```

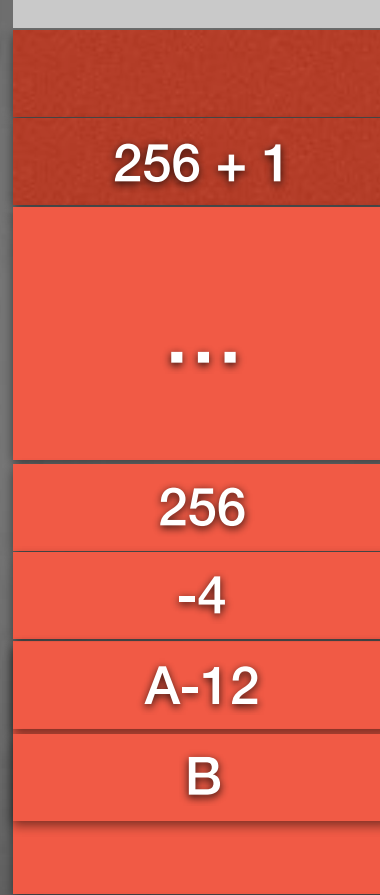
chunk 0

unlink(fake)

BK
(B)



P
(fake)



256 + 1

...

256

-4

A-12

B

un_size + 1

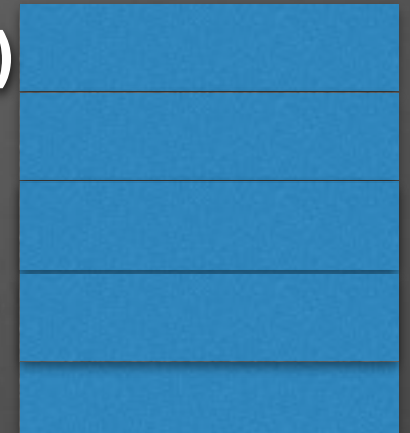
unallocated

p0

break

FD
(A-12)

A

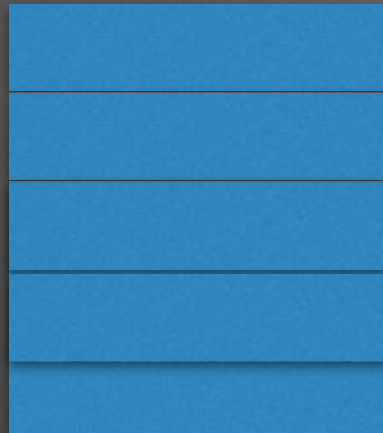


```
#define unlink(P) {  
    FD = P->fd;  
    BK = P->bk;  
    FD->bk = BK;  
    BK->fd = FD;  
    ...  
}
```

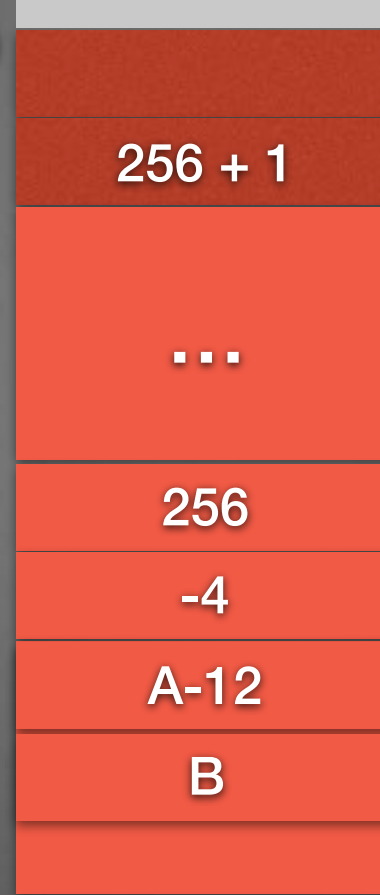
chunk 0

unlink(fake)

BK
(B)

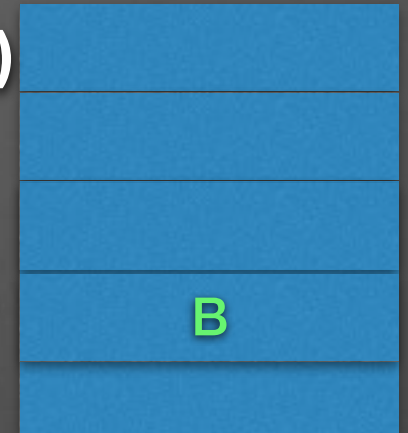


P
(fake)



FD
(A-12)

A



```
#define unlink(P) {  
    FD = P->fd;  
    BK = P->bk;  
    FD->bk = BK;  
    BK->fd = FD;  
    ...  
}
```

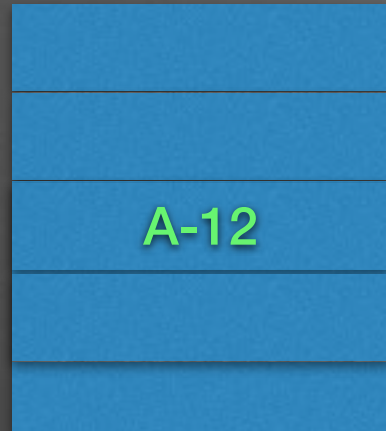
un_size + 1

unallocated

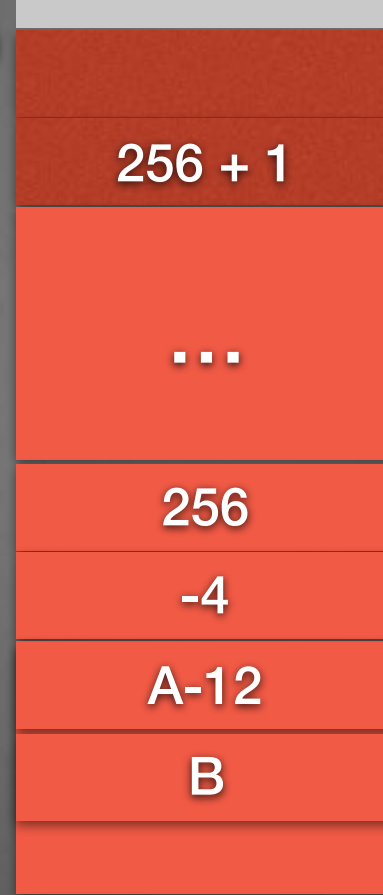
chunk 0

unlink(fake)

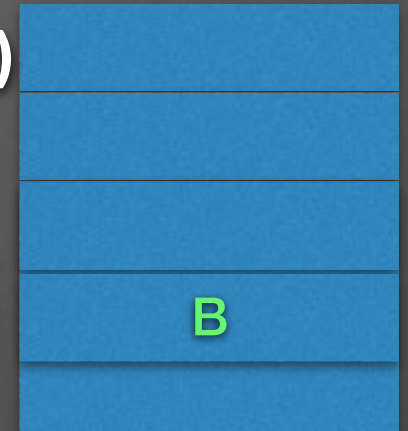
BK
(B)



P
(fake)



FD
(A-12)



un_size + 1

unallocated

```
#define unlink(P) {  
    FD = P->fd;  
    BK = P->bk;  
    FD->bk = BK;  
    BK->fd = FD;  
    ...  
}
```

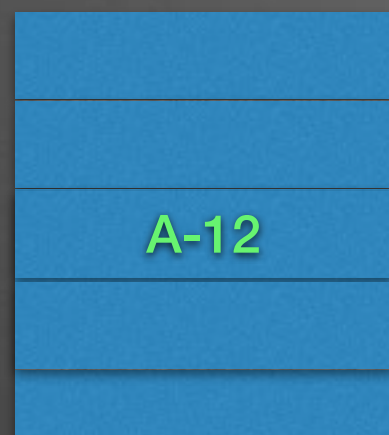
p0

break

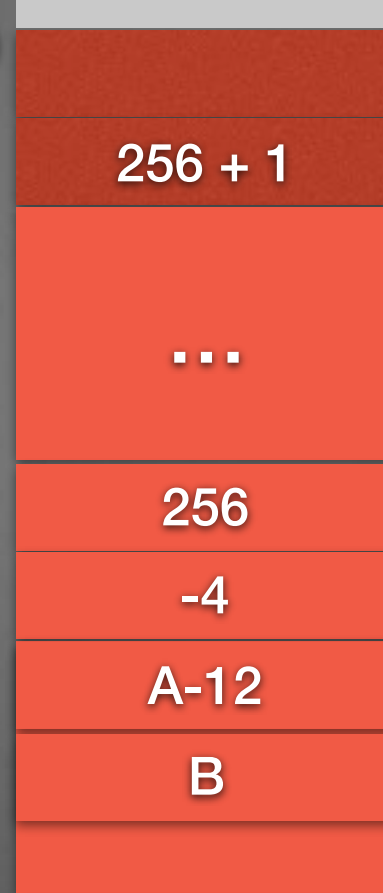
chunk 0

unlink(fake)

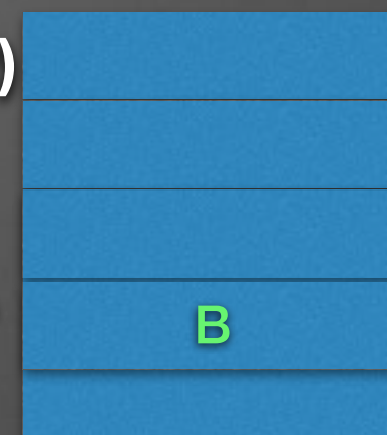
BK
(B)



P
(fake)



FD
(A-12)



p0

break

un_size + 1

unallocated

实现了在地址A写入B，
由于A和B可以为任意
值，即可以实现任意地
址任意写。
副作用是会在B+8位置
写入A-12。