节，并且创建初始的空闲块。此刻，分配器已初始化了，并且准备好接受来自应用的分配和释放请求。

code/vm/malloc/mm.c

```
1  int mm_init(void)
2  {
3      /* Create the initial empty heap */
4      if ((heap_listp = mem_sbrk(4*WSIZE)) == (void *)-1)
5          return -1;
6      PUT(heap_listp, 0);                          /* Alignment padding */
7      PUT(heap_listp + (1*WSIZE), PACK(DSIZE, 1)); /* Prologue header */
8      PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 1)); /* Prologue footer */
9      PUT(heap_listp + (3*WSIZE), PACK(0, 1));     /* Epilogue header */
10     heap_listp += (2*WSIZE);
11
12     /* Extend the empty heap with a free block of CHUNKSIZE bytes */
13     if (extend_heap(CHUNKSIZE/WSIZE) == NULL)
14         return -1;
15     return 0;
16 }
```

code/vm/malloc/mm.c

图 9-44　mm_init：创建带一个初始空闲块的堆

code/vm/malloc/mm.c

```
1  static void *extend_heap(size_t words)
2  {
3      char *bp;
4      size_t size;
5
6      /* Allocate an even number of words to maintain alignment */
7      size = (words % 2) ? (words+1) * WSIZE : words * WSIZE;
8      if ((long)(bp = mem_sbrk(size)) == -1)
9          return NULL;
10
11     /* Initialize free block header/footer and the epilogue header */
12     PUT(HDRP(bp), PACK(size, 0));         /* Free block header */
13     PUT(FTRP(bp), PACK(size, 0));         /* Free block footer */
14     PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1)); /* New epilogue header */
15
16     /* Coalesce if the previous block was free */
17     return coalesce(bp);
18 }
```

code/vm/malloc/mm.c

图 9-45　extend_heap：用一个新的空闲块扩展堆

extend_heap 函数会在两种不同的环境中被调用：1)当堆被初始化时；2)当 mm_malloc 不能找到一个合适的匹配块时。为了保持对齐，extend_heap 将请求大小向上舍入为最接近的 2 字(8 字节)的倍数，然后向内存系统请求额外的堆空间(第 7～9 行)。

extend_heap 函数的剩余部分(第 12～17 行)有点儿微妙。堆开始于一个双字对齐的边界，并且每次对 extend_heap 的调用都返回一个块，该块的大小是双字的整数倍。因此，对 mem_sbrk 的每次调用都返回一个双字对齐的内存片，紧跟在结尾块的头部后面。这个头部变成了新的空闲块的头部(第 12 行)，并且这个片的最后一个字变成了新的结尾