OS LAB 3

PB18000227 艾语晨

隐式空闲链表管理

堆分配的空间利用情况：总共5MB的堆空间，分成一个一个的块（占用、空闲）并用链表连接。块内部由头部、有效载荷、填充组成

这个总空间是5MB，从 `mem_start_brk` 到 `mem_max_addr` ，用 `sbrk` 来扩展

需要自己实现的链表分配是在这个堆内部的一部分空间 (即原文档第7页的 `Mymalloc`)，放了一个链表，调用 `mem_brk` （封装为 `extend_heap` ）来进行扩容

2.4 放置策略

代码思路分析

> 采取首次适配。从头开始搜索链表，找到第一个大小合适的空闲内存块便返回

第一个块在 `heap_listp` 所指向的位置，判断这个块是否空闲，不是就跳转到下一个，直到 `Mymalloc` 的最后

没访问一个头块就查看空闲状态和大小，若满足要求则返回头块指针，若查找完全之后仍然没有则返回NULL

源代码

```c
static void *find_fit(size_t asize)
{

    char *start = heap_listp;      /* the starting addr of a block */
    while (GET(HDRP(start)) != 1) /* this block is not hdr */
    {
        if (!(GET_ALLOC(HDRP(start))) && GET_SIZE(HDRP(start)) >= asize)
        { /* it's free and large enough */
            return (void *)start;
        }
        else
        {
            start = NEXT_BLKP(start);
        }
    }
    return NULL; /* leave the demanding-space-task to caller */
}
```

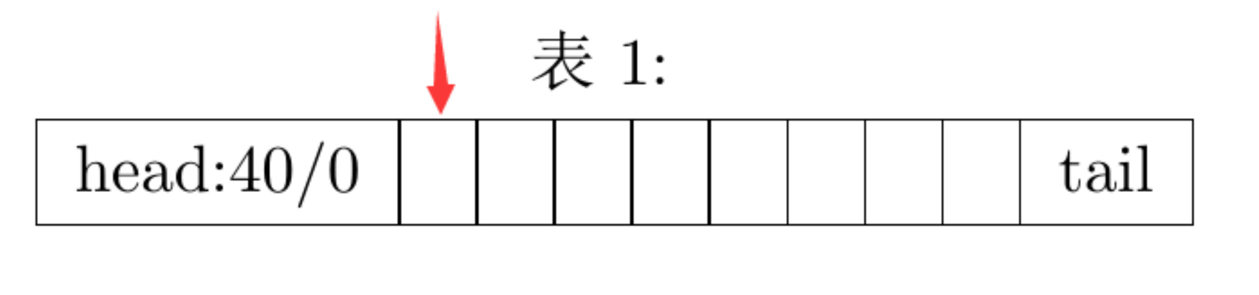
2.5 分割空闲块

代码思路分析

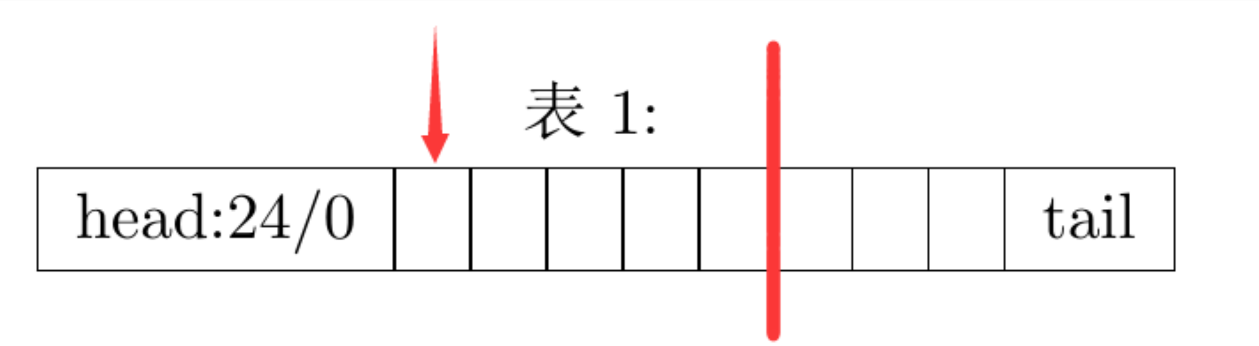
> 注意还是要双字对齐的

如果需要拆分，则修改传进来的头块大小和占有位，设置其尾块（指向原块剩余部分的指针随大小而由define修改），并设置剩余（空闲）块的头块，修改其尾块大小

整个空闲块如下：（箭头指向bp）



如果需要拆分，如下所示：



那么修改原来的头尾和分割点两侧新产生的头尾即可

出现过的问题

- 不需要重新对齐双字（见`mm_malloc()`实现）否则会两次乘以WSIZE

源代码

```
static void place(void *bp, size_t asize)
{
```

```c
    const size_t total_size = GET_SIZE(HDRP(bp));
    size_t rest = total_size - asize;

    if (rest >= MIN_BLK_SIZE)
    /* need split */
    {
        /* size_t true_allo_size; */
        /* Allocate an even number of words to maintain alignment */
        /* true_allo_size = (asize % 2) ? (asize + 1) * WSIZE : asize * WSIZE; */
        PUT(HDRP(bp), PACK(asize, 1));                       /* head of new block */
        PUT(FTRP(bp), PACK(asize, 1));                       /* foot of new block */
        PUT(HDRP(NEXT_BLKP(bp)), PACK(rest, 0));             /* head of rest block */
        PUT(FTRP(NEXT_BLKP(bp)), PACK(rest, 0));             /* foot of rest block */
    }
    else
    {
        PUT(HDRP(bp), PACK(total_size, 1)); /* head of new block (fillings included) */
        PUT(FTRP(bp), PACK(total_size, 1)); /* foot of new block */
    }
}
```

2.8 合并步骤

代码思路分析

就是把第二种和第三种结合一下

源代码

```c
static void *coalesce(void *bp)
{
    size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));
    if (prev_alloc && next_alloc)
    {
        return bp;
    }
    else if (prev_alloc && !next_alloc)
    {
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(bp), PACK(size, 0));
        PUT(FTRP(bp), PACK(size, 0));
    }
    else if (!prev_alloc && next_alloc)
    {
        size += GET_SIZE(FTRP(PREV_BLKP(bp)));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        PUT(FTRP(bp), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }
    else
    {
        size += GET_SIZE(HDRP(NEXT_BLKP(bp))) + GET_SIZE(FTRP((PREV_BLKP(bp))));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }
    return bp;
}

```

Makefile (第一部分)

代码思路分析

按照参考文档编写；编译没有用预定义而是采用 `cc`

源代码

```makefile
#
# Students' Makefile for the Malloc Lab
#

CC = gcc -g
CFLAGS = -Wall

# 待补充
OBJS = mm.o mmdriver.o memlib.o

mmdriver: $(OBJS)
# 待补充gcc命令（使用变量）
    cc -o mmdriver $(OBJS)

#待补充
mmdriver.o: mmdriver.h config.h mm.h memlib.h
memlib.o: config.h mm.h memlib.h
mm.o: config.h mm.h memlib.h

.PHONY : clean
clean:
    -rm -f *~ *.o mmdriver

```

显式空间链表管理

find_fit

代码思路分析

由于显式链表中空闲块直接由前驱后继关系，则无需判断遍历到的是否为空闲

源代码

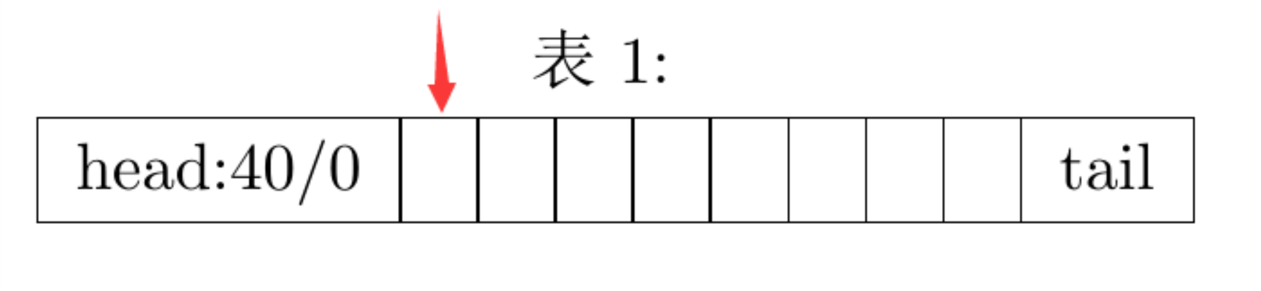
```c
static void *find_fit(size_t asize)
{
    char *bp = free_listp;
    if (free_listp == NULL)
        return NULL;

    while (bp != NULL) /*not end block;*/
    {
        if (GET_SIZE(HDRP(bp)) >= asize)
        {
            break;
        }
        else
        {
            bp = (char *)GET_SUCC(bp);
        }
    }
    return (bp != NULL ? ((void *)bp) : NULL);
}
```

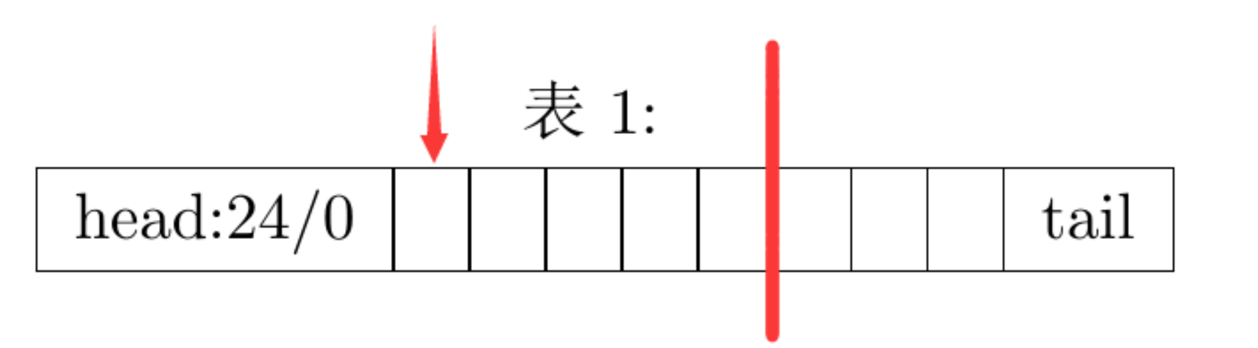
place

代码思路分析

整个空闲块如下：（箭头指向bp）



如果需要拆分，如下所示：



首先修改原有块头（其中PRE的allocated属性不变）

然后考虑后面新的空闲块：修改其头尾参数表，然后将其加入空闲块链表

如果不需要拆分，则需要额外修改在图示区域之后的那个块的PRE参数 (即1位) 为1

源代码

```
static void place(void *bp, size_t asize)
{
    size_t total_size = 0;
    size_t rest = 0;
    /* void *nbp; */
    /* nbp = NEXT_BLKP(bp); */
    delete_from_free_list(bp);
    /*remember notify next_blk, i am alloced*/
    total_size = GET_SIZE(HDRP(bp));
    rest = total_size - asize;

    if (rest >= MIN_BLK_SIZE) /*need split*/
    {
        /* to write the head of newly allocated block */
        /* actually the pre-infro hasn't been changed, thus still use G.. func */
        PUT(HDRP(bp), PACK(asize, GET_PREV_ALLOC(HDRP(bp)), 1));
        /* change bp in advance can lessen addr calculation */
        PUT(HDRP(NEXT_BLKP(bp)), PACK(rest, 1, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(rest, 1, 0));
        add_to_free_list(NEXT_BLKP(bp));
    }
    else
    {
        PUT(HDRP(bp), PACK(total_size, GET_PREV_ALLOC(HDRP(bp)), 1));
        /* pre-allo infro of the next block should be changed */
        if (GET_ALLOC(HDRP(NEXT_BLKP(bp))) == 0)    /* free */
        {
            PUT(HDRP(NEXT_BLKP(bp)), PACK(GET_SIZE(HDRP(NEXT_BLKP(bp))), 1, 0));
            PUT(FTRP(NEXT_BLKP(bp)), PACK(GET_SIZE(FTRP(NEXT_BLKP(bp))), 1, 0));
        }
        else    /* allocated */
        {
            PUT(HDRP(NEXT_BLKP(bp)), PACK(GET_SIZE(HDRP(NEXT_BLKP(bp))), 1, 1));
            /* don't need to change tail in*/
            /* PUT(FTRP(nbp), PACK(GET_SIZE(nbp), 1, 1)); */
        }
    }
}
```

Makefile_for_all

仍然是参考了参考文档，在隐式链表的makefile上修改的

源代码

```
#
# Students' Makefile for the Malloc Lab
# it will generate two executable files
# mmdriver tests the implicit linked-list
# ep_mmdriver tests the explicit linked-list
#

CC = gcc -g
CFLAGS = -Wall

OBJS1 = memlib.o  mmdriver.o mm.o
OBJS2 = memlib.o  ep_mmdriver.o ep_mm.o
OBJS_ALL = memlib.o  mmdriver.o ep_mmdriver.o mm.o ep_mm.o

all : mmdriver ep_mmdriver

mmdriver : $(OBJS1)
    cc -o mmdriver $(OBJS1)
ep_mmdriver : $(OBJS2)
    cc -o ep_mmdriver $(OBJS2)

memlib.o : memlib.h config.h
mm_o :  mm.h memlib.h
ep_mm.o : ep_mm.h memlib.h
mmdriver.o :  mm.h memlib.h
ep_mmdriver.o : ep_mm.h memlib.h

.PHONY : clean
clean:
    -rm mmdriver ep_mmdriver $(OBJS_ALL)


```

P.S.

独立出来了一些`.h`文件，一并放在总目录下