实验报告

刘雅迪
计 26

一、实验构思：

使用显示空闲链实现动态内存分配器。将堆组织成一个双向的空闲链表，在每个空闲块中，都包含一个 pred 和 succ 指针。这样首次适配的分配时间从块总数的线性时间减少到了空闲块数量的线性时间。

双向链表包含一个 free_list_headp 和 free_list_tailp 节点指针，以便对空闲链表进行插入修改等操作。

二、实验具体内容实现：

1. 定义的宏与函数：

```
#define ALIGNMENT 16
#define ALIGN(size) (((size) + (ALIGNMENT - 1)) & ~(ALIGNMENT - 1))
#define SIZE_T_SIZE (ALIGN(sizeof(size_t)))

/* Base constants and macros */
#define WSIZE       4
#define DSIZE       8
#define CHUNKSIZE   (1<<12)

#define MAX(x, y) ((x) > (y) ? (x) : (y))
#define PACK(size, alloc)   ((size) | (alloc))

#define GET(p)      (*(unsigned int *)(p))
#define PUT(p, val) (*(unsigned int *)(p) = val)

#define GET_PTR_VAL(p)  (*(unsigned long *)(p))
#define SET_PTR(p, ptr) (*(unsigned long *)(p) = (unsigned long)(ptr))

#define GET_PRED(p) ((char *)(*(unsigned long *)(p)))
#define GET_SUCC(p) ((char *)(*(unsigned long *)(p + DSIZE)))
#define SET_PRED(p, ptr)    (SET_PTR((char *)(p), ptr))
#define SET_SUCC(p, ptr)    (SET_PTR(((char *)(p)+(DSIZE)), ptr))

#define GET_SIZE(p)     (GET(p) & ~0x7)
#define GET_ALLOC(p)    (GET(p) & 0x1)

#define HDRP(bp)    ((char* )(bp) - WSIZE)
#define FTRP(bp)    ((char* )(bp) + GET_SIZE(HDRP(bp)) - DSIZE)

#define NEXT_BLKP(bp)   ((char *)(bp) + GET_SIZE(((char *)(bp) - WSIZE)))
#define PREV_BLKP(bp)   ((char *)(bp) - GET_SIZE(((char *)(bp) - DSIZE)))

static char *free_list_headp;
static char *free_list_tailp;
static void *extend_heap(size_t size); //拓展堆块
static void *find_fit(size_t size); //寻找空闲块
static void place(void *bp, size_t size); //分割空闲块
static void *coalesce(void *bp); //合并空闲块
```

2. int mm_init(void)

显式空闲链表初始化时分配一个空的堆区，头部和尾部都要添加一个 4 字节的 Padding，以满足对齐要求。

```
int mm_init(void)
{
    if ((free_list_headp = mem_sbrk(4*WSIZE+3*DSIZE)) == (void *)-1) {
        return -1;
    }
    PUT(free_list_headp, PACK(0, 0));
    PUT(free_list_headp+WSIZE, PACK(24, 1));
    free_list_headp += DSIZE;
    free_list_tailp = NEXT_BLKP(free_list_headp);
    SET_PRED(free_list_headp, NULL);
    SET_SUCC(free_list_headp, free_list_tailp);
    PUT(free_list_headp+(2*DSIZE), PACK(24, 1));
    PUT(HDRP(free_list_tailp), PACK(0, 1));
    SET_PRED(free_list_tailp, free_list_headp);
    PUT(free_list_tailp+DSIZE, PACK(0, 0));
    if (extend_heap(CHUNKSIZE/WSIZE) == NULL) {
        return -1;
    }

    return 0;
}
```

3. void *mm_malloc(size_t size);

设定 asize，如果 size<16bytes，那么用最小块大小 24bytes 代替；否则，需要只考虑头部尾部的开销，还需要进行对齐。然后在空闲链表使用首次适配的方式，使用 find_fit 函数进行顺序搜索，找到第一个适配的空闲块后运用 place 函数放置。

```
void *mm_malloc(size_t size)
{
    size_t asize;
    size_t extendsize;
    char *bp;

    if (size == 0)
        return NULL;

    if (size <= 2*DSIZE) {
        asize = 3*DSIZE;
    } else {
        asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);
    }
    asize = ALIGN(asize);
    if ((bp = find_fit(asize)) != NULL) {
        place(bp, asize);
        return bp;
    }

    extendsize = MAX(asize, CHUNKSIZE);
    if ((bp = extend_heap(extendsize/WSIZE)) == NULL)
        return NULL;
    place(bp, asize);

    return bp;
}
```

4. void mm_free(void *ptr);

在空闲链表中顺序查找，找到第一个空闲的块，然后设置相应的 PRED 和 SUCC 的值。释放掉这个块后调用 coalesce 对这个块进行合并，操作在常数时间内。

```
void mm_free(void *ptr)
{
    char *p;
    size_t size = GET_SIZE(HDRP(ptr));
    for (p = GET_SUCC(free_list_headp); ; p = GET_SUCC(p)) {
        if (ptr < (void *)p) {
            PUT(HDRP(ptr), PACK(size, 0));
            PUT(FTRP(ptr), PACK(size, 0));
            SET_SUCC(ptr, p);
            SET_PRED(ptr, GET_PRED(p));
            SET_SUCC(GET_PRED(p), ptr);
            SET_PRED(p, ptr);
            break;
        }
    }
    coalesce(ptr);
}
```

5. void *mm_realloc(void *ptr, size_t size);

首先对 ptr 和 size 进行判断：如果 ptr==NULL，调用 mm_malloc(size)；否则如果 size==NULL，调用 mm_free(ptr)。

进行到后面的代码时 ptr!=NULL 并且 size>0，首先对 size 进行调整，添加头部尾部等，然后判断 asize 和这个块的大小关系：

- 如果 asize == blockSize，什么都不做；
- 如果 asize < blockSize，与 place 函数操作类似，但是不直接调用 place 函数；
- 如果 asize > blockSize，考虑下一个块是否已分配并且考虑其大小。

```
void *mm_realloc(void *ptr, size_t size)
{
    void *oldptr = ptr;
    void *newptr;
    void *nextptr;
    void *pred;
    void *succ;
    char *p;
    size_t blockSize;
    size_t extendsize;
    size_t asize;
    size_t sizesum;

    if (ptr == NULL) {
        return mm_malloc(size);
    } else if (size == 0) {
        mm_free(ptr);
        return NULL;
    }
    if (size <= 2*DSIZE) {
        asize = 3*DSIZE;
    } else {
        asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);
    }
    asize = ALIGN(asize);
    blockSize = GET_SIZE(HDRP(ptr));
```

```
if (asize == blockSize) {
    return ptr;
} else if (asize < blockSize) {
    if (blockSize-asize >= 3*DSIZE) {
        PUT(HDRP(ptr), PACK(asize, 1));
        PUT(FTRP(ptr), PACK(asize, 1));
        nextptr = NEXT_BLKP(ptr);
        PUT(HDRP(nextptr), PACK(blockSize-asize, 0));
        PUT(FTRP(nextptr), PACK(blockSize-asize, 0));
        for (p = GET_SUCC(free_list_headp); ; p = GET_SUCC(p)) {
            if (nextptr < (void *)p) {
                pred = GET_PRED(p);
                succ = p;
                SET_PRED(nextptr, pred);
                SET_SUCC(nextptr, succ);
                SET_SUCC(pred, nextptr);
                SET_PRED(p, nextptr);
                break;
            }
        }
    }
    return ptr;
```

```
} else {
    nextptr = NEXT_BLKP(ptr);
    sizesum = GET_SIZE(HDRP(nextptr))+blockSize;
    if (!GET_ALLOC(HDRP(nextptr)) && sizesum >= asize) {
        pred = GET_PRED(nextptr);
        succ = GET_SUCC(nextptr);
        if (sizesum-asize >= 3*DSIZE) {
            PUT(HDRP(ptr), PACK(asize, 1));
            PUT(FTRP(ptr), PACK(asize, 1));
            nextptr = NEXT_BLKP(ptr);
            PUT(HDRP(nextptr), PACK(sizesum-asize, 0));
            PUT(FTRP(nextptr), PACK(sizesum-asize, 0));
            SET_PRED(nextptr, pred);
            SET_SUCC(nextptr, succ);
            SET_SUCC(pred, nextptr);
            SET_PRED(succ, nextptr);
        } else {
            PUT(HDRP(ptr), PACK(sizesum, 1));
            PUT(FTRP(ptr), PACK(sizesum, 1));
            SET_SUCC(pred, succ);
            SET_PRED(succ, pred);
        }
        return ptr;
    } else {
        newptr = find_fit(asize);
        if (newptr == NULL) {
            extendsize = MAX(asize, CHUNKSIZE);
            if ((newptr = extend_heap(extendsize/WSIZE)) == NULL) {
                return NULL;
            }
        }
        place(newptr, asize);
        memcpy(newptr, oldptr, blockSize-2*WSIZE);
        mm_free(oldptr);
        return newptr;
    }
}
```

三、实验感想：

这次实验对我来说很难，感觉是三个实验中最难的一个。程序可以运行的瞬间无疑是开心的，但是分数不高，尝试改了好久都不知道为什么吞吐率那么低，怎么改都没什么用。无奈只能放弃了，尽管性能分可能没那么高，但至少还有一些基础分数。希望助教哥哥 or 姐姐能手下留情，多给我点分数，为这门课画上一个完美一点的句号。

四、参考资料：

1. 《深入理解计算机系统》第九章
2. 《CSAPP(CMU 15-213)：Lab6 Malloclab 详解》，
https://blog.csdn.net/qq_42241839/article/details/123697377