Recitation 10: Malloc Lab

Andrew Faulring
15213 Section A
11 November 2002

Logistics

- faulring@cs.cmu.edu
- Office hours
  - NSH 2504
  - Tuesday 2–3
- Exam 2
  - Tuesday, 12 November, 6:00-7:20pm
  - Doherty Hall 2315
- Lab 6 (Malloc)
  - due next Tuesday, 19 November

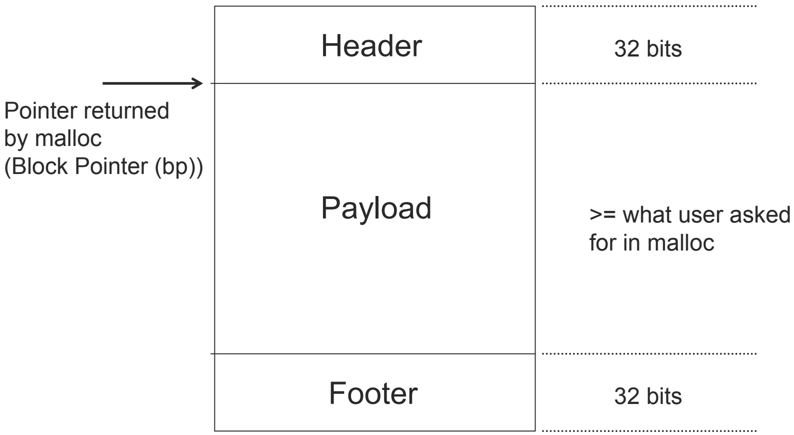
Today’s Plan

- The Malloc Lab
  - Understand mm-helper.c
  - Adding debugging info to mm-helper.c

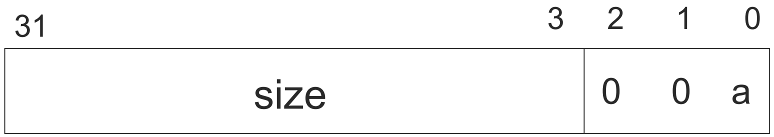
What does mm-helper.c do ?

- Implicit Free List
  - Header with each block – (size / allocated bit)
  - No separate Free List – free blocks linked implicitly by size fields in header
- First Fit
  - Searches free list from beginning and picks first block that fits
- Immediate Boundary Tag Coalescing
  - Footer (boundary tag), replica of header

Block Format

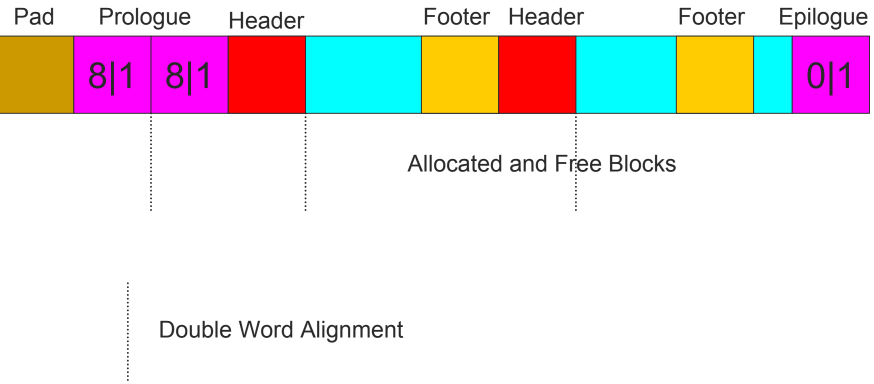


Header/Footer Format



- Double word alignment
  - Three lower-order bits of size always 0
- Pack size and allocated bits into a single integer
  - Size = 24 (0x18). Block is allocated
    Header = 0 x18 | 0x1 = 0x19

Heap Format



Very Useful Macros

- #define WSIZE 4
- #define DSIZE 8
- #define CHUNKSIZE (1<<12)
- #define OVERHEAD 8

Very Useful Macros

- #define PACK(size, alloc) ((size) | (alloc))
- #define GET(p) (*(size_t *)(p))
- #define PUT(p, val) (*(size_t *)(p) = (val))
- #define GET_SIZE(p) (GET(p) & ~0x7)
- #define GET_ALLOC(p) (GET(p) & 0x1)

Very Useful Macros

- #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)))

Initializing the heap

```
int mm_init(void) {
    if ((heap_listp = mem_sbrk(4*WSIZE)) == NULL)
        return -1;
    PUT(heap_listp, 0);
    PUT(heap_listp+WSIZE, PACK(OVERHEAD, 1));
    PUT(heap_listp+DSIZE, PACK(OVERHEAD, 1));
    PUT(heap_listp+WSIZE+DSIZE, PACK(0, 1));
    heap_listp += DSIZE;

    if (extend_heap(CHUNKSIZE/WSIZE) == NULL)
        return -1;

    return 0;
}
```

Extending the Heap

```
static void *extend_heap(size_t words) {
    char *bp;
    size_t size;

    size = (words % 2) ? (words+1) * WSIZE : words * WSIZE;
    if ((int)(bp = mem_sbrk(size)) < 0)
        return NULL;

    PUT(HDRP(bp), PACK(size, 0));
    PUT(FTRP(bp), PACK(size, 0));
    PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1));

    return coalesce(bp);
}
```

Coalescing

```
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) { ….. }

        else if (!prev_alloc && next_alloc) {
                size += GET_SIZE(HDRP(PREV_BLKP(bp)));
                PUT(FTRP(bp), PACK(size, 0));
                PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
                bp = PREV_BLKP(bp); }

        else { ……. }

        return bp;
}
```

Malloc

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

        if (size <= 0) return NULL;
        if (size <= DSIZE)
                 asize = DSIZE + OVERHEAD;
        else
                asize = DSIZE * ((size + (OVERHEAD) + (DSIZE-1)) / DSIZE);

        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; }
```

Finding First Fit

```
static void *find_fit(size_t asize) {
        void *bp;

        for (bp = heap_listp; GET_SIZE(HDRP(bp)) > 0; bp = NEXT_BLKP(bp))
                if (!GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp))))
                        return bp;

        return NULL;
}
```

Placing a block in a free chunk

```
static void place(void *bp, size_t asize) {
        size_t csize = GET_SIZE(HDRP(bp));

        if ((csize - asize) >= (DSIZE + OVERHEAD)) {
            PUT(HDRP(bp), PACK(asize, 1));
            PUT(FTRP(bp), PACK(asize, 1));
            bp = NEXT_BLKP(bp);
            PUT(HDRP(bp), PACK(csize-asize, 0));
            PUT(FTRP(bp), PACK(csize-asize, 0));
       }
       else {
            PUT(HDRP(bp), PACK(csize, 1));
            PUT(FTRP(bp), PACK(csize, 1));
       }
}
```

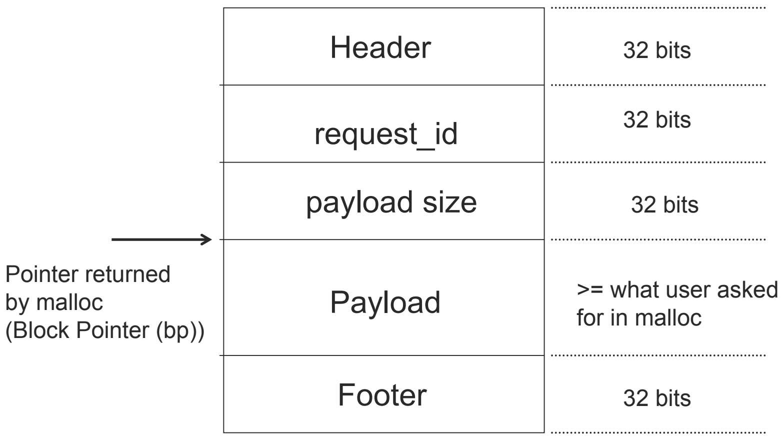
Free

```
void mm_free(void *bp) {
    size_t size = GET_SIZE(HDRP(bp));

    PUT(HDRP(bp), PACK(size, 0));
    PUT(FTRP(bp), PACK(size, 0));

    coalesce(bp);
}
```

Adding debugging information

- For each allocated block
  - request_id : malloc request counter (0..)
    - Initialize in mm_init
    - Increment in malloc
  - payload size : the memory requested by malloc
    - Can be different from the allocated size
- Where do we store this
  - In the allocated block header

Allocated Block Format



Header — 32 bits

request_id — 32 bits

payload size — 32 bits

Payload — >= what user asked for in malloc

Footer — 32 bits

Pointer returned by malloc (Block Pointer (bp))

One way to implement this

- Inside malloc
  - Allocate additional memory in malloc
  - OVERHEAD = 16

```
PUT(bp,request_counter);
PUT(bp+4,size);
return bp+DSIZE;
```

- Inside Free

```
bp = bp – DSIZE;
```

Heapcheck

- Put all sorts of sanity checks
- Scan the implicit list
  - like the first fit function
  - print request_id and size

Explicit Lists

- Separate Free List
  - Can find a free block quickly
- Change Free Block Format
  - Add prev pointer
  - Add next pointer
- Where to store free list pointer
  - Only one WORD
  - Can store in unused PAD word
- Some functions to add
  - static void insertfree_block(void * freeblkptr);
  - static void removefree_block(void * freeblkptr);