系统程序设计作业Unit4

2014302580341 卓越二班 余璞轩

> *八个测试例子在Darwin平台上手动测试无误，用`perl`文件测试最后一个bonus题结果有误，原因未知；其他平台未知，应该没很大区别。*

`checksum()`函数

```
int checksum(void *p, size_t s) {
    int *pp = (int *)p;
    int sum = 0;
    while (s > 0) {
        int x = *pp;
        int val = 0;
        for (int i = 0; i != 8; i++) {
            val += x & 0x01010101;
            x = x >> 1;
        }
        val += (val >> 16);
        val += (val >> 8);
        sum += val & 0xff;
        s -= 4;
        pp++;
    }
    return sum;
}
```

目的是计算checksum(4 Bytes, 32 bits) 中1的个数，是一种简单的用来判断header是否损坏的机制。实现起来比较简单。

`block`结构体

```
struct block {
    int adr;
    int flg;
};
```

简单拿来存储block信息的结构体，包含是否被占用的flg，和具体地址adr。

`MyMalloc()`函数

```c
void *MyMalloc(size_t size, char *filename, int linenumber) {
    size_t tsize, payload;
    char *bp;
    if (size == 0) {
        return NULL;
    }
    payload = size;
    /* total size to allocate */
    tsize = headerSize + payload + footerSize;
    bp = (char *)malloc(tsize);
    /* clear the whole part by 0 */
    memset(bp, 0, tsize);
    bp += CHECKSUMSIZE;
    strcpy(bp, filename);
    bp += FILENAMESIZE;
    PUT(bp, linenumber);
    bp += LINESIZE;
    PUT(bp, size);
    bp += BLOCKSIZE;
    PUT(bp, 0xCCDEADCC);
    bp += (FENCESIZE + payload);
    PUT(bp, 0xCCDEADCC);
    /* Roll back and fill in the checksum. */
    bp -= (payload + headerSize);
    PUT(bp, checksum(bp + CHECKSUMSIZE, headerSize - CHECKSUMSIZE));
    blk[0].flg++;
    for (int i = 1; i < MAXNUM; i++) {
        if (blk[i].flg == 0) {
            blk[i].flg = 1;
            blk[i].adr = (int)bp;
            break;
        }
    }
    return (void *)(bp + headerSize);
}
```

数据块的结构应该是这样的

CheckSum -> FileName -> LineNumber -> BlockSize -> "Fence" -> Block -> "Fence"

先将总共要allocate的区域用`malloc()`搞定，然后通过在当前区域不断变动地址，指定一些原始`malloc()`没有办法确定的值。

事实上这个函数的本质就是在一个指针(bp)前后跳动位置，并且用PUT这个宏在对应位置写入值。

```c
/* Read and write a word at address p */
#define GET(p) (*(unsigned int *)(p))
#define PUT(p,val) (*(unsigned int *)(p)=(val))
```

> *参考CSAPP中文版第572面*

`MyFree()`函数

```c
void MyFree(void *ptr, char *filename, int linenumber) {
    for (int i = 1; i <= MAXNUM; i++) {
        if (i == MAXNUM) {
            error(4, filename, linenumber);
            break;
        }
        char *p = (char *)(ptr - headerSize);
        if (blk[i].flg == 1 && blk[i].adr == (int)p) {
            int check = GET(p);
            if(check ^ checksum(p + CHECKSUMSIZE, headerSize - CHECKSUMSIZE)) {
                error(3, filename, linenumber);
            }
            char m_filename[FILENAMESIZE];
            strcpy(m_filename, p + CHECKSUMSIZE);
            int m_linenumber = GET(p + CHECKSUMSIZE + FILENAMESIZE);
            int head = GET(ptr - FENCESIZE);
            if(head ^ 0xCCDEADCC) {
                errorfl(1, m_filename, m_linenumber, filename, linenumber);
            }
            unsigned int payload = GET(ptr - FENCESIZE - BLOCKSIZE);
            int tail = GET(ptr + payload);
            if(tail ^ 0xCCDEADCC) {
                errorfl(2, m_filename, m_linenumber, filename, linenumber);
            }
            blk[i].flg = 0;
            blk[0].flg--;
            free(p);
            break;
        }
    }
}
```

这个函数的基本步骤就是:

在blk中找到符合地址的项后:

- 如果header损坏就报第三个错误
- header里fence损坏就报第一个错误
- footer里fence损坏就报第二个错误

没有错误的话就正常`free()`就行。

`AllocatedSize()`函数

```
int AllocatedSize() {
    int num = blk[0].flg;
    int sum = 0;
    for (int i = 1; i < MAXNUM && num; i++) {
        if (blk[i].flg == 0) {
            continue;
        }
        char *p = (char *)(blk[i].adr);
        /* sum += block size */
        sum += GET(p + CHECKSUMSIZE + FILENAMESIZE + LINESIZE);
        num--;
    }
    return sum;
}
```

把所有flg为1的block的size加起来就行了。

`PrintAllocatedBlocks()`函数

```
void PrintAllocatedBlocks() {
    int num = blk[0].flg;
    if(num) {
        printf("Allocated blocks are:\n");
    }
    for (int i = 1; i < MAXNUM && num; i++) {
        if (blk[i].flg == 1) {
            char *p = (char *)(blk[i].adr);
            int payload = GET(p + CHECKSUMSIZE + FILENAMESIZE + LINESIZE);
            char m_filename[FILENAMESIZE];
            strcpy(m_filename, p + CHECKSUMSIZE);
            unsigned int m_linenumber = GET(p + CHECKSUMSIZE + FILENAMESIZE);
            PRINTBLOCK(payload, m_filename, m_linenumber);
            num--;
        }
    }
    return;
}
```

把所有flg为1的block的信息通过MACRO(`PRINTBLOCK`)打印出来即可。

`HeapCheck()`函数

```
int HeapCheck() {
    int num=blk[0].flg;
    for (int i = 1; i < MAXNUM && num; i++) {
        if (blk[i].flg == 1) {
            char *p = (char *)(blk[i].adr);
            char m_filename[FILENAMESIZE];
            strcpy(m_filename, p + CHECKSUMSIZE);
            int m_linenumber = GET(p + CHECKSUMSIZE + FILENAMESIZE);
            int head = GET(p + headerSize - FENCESIZE);
            if(head ^ 0xCCDEADCC) {
                PRINTERROR(1, m_filename, m_linenumber);
                return -1;
            }
            unsigned int payload = GET(p + headerSize - FENCESIZE - BLOCKSIZE);
            int tail = GET(p + headerSize + payload);
            if(tail ^ 0xCCDEADCC) {
                PRINTERROR(2, m_filename, m_linenumber);
                return -1;
            }
        }
    }
    return 0;
}
```

把blk中flg为1的块的fence全部检查一次，没问题就返回0，否则就`PRINTERROR`并返回-1。

attach: `grader.pl`测试结果

```
Test #      Error?  Location    Reason
1 (10%):    1   .       .
2 (15%):    1   1       1
3 (15%):    1   1       1
4 (15%):    0   .       0
5 (15%):    1   1       1
6 (15%):    1   0.5     1
7 (15%):    1   1       1
8 (0% Bonus):   0   0       0
------------
Header Overwrite:   2.5/3
Footer Payload Overflow:    6/6
Freeing Unallocated Block:  6/6
Memory Leak:    1/3
Global List:    0/0
------------
Error Recognition:  80.3%
Best Error Message:     100%
Filename/Line Number:   100%
Memory Leak Analysis (Bonus):    N/A
------------
Total: 85%
```