9.11

A:虚拟内存地址格式

|------------------TLBT------------------|----TLBT---|

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

B: 地址翻译

|  |  |
| --- | --- |
| 参数 | 值 |
| VPN | 0x09 |
| TLB索引 | 0x01 |
| TLB 标记 | 0x02 |
| TLB 命中？（是/否） | 否 |
| 缺页？（是/否） | 否 |
| PPN | 0x17 |

C:物理地址格式

|-----------------------CT-------------------------|---------------CI---------------|-------CO-----|

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

D:物理地址引用

|  |  |
| --- | --- |
| 参数 | 值 |
| 字节偏移 | 0x00 |
| 索引缓存 | 0x0F |
| 缓存标记 | 0x17 |
| 缓存命中？（是/否） | 否 |
| 返回的缓存字节 | - |

9.13

A:虚拟内存地址格式

|------------------TLBT------------------|----TLBT---|

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

B: 地址翻译

|  |  |
| --- | --- |
| 参数 | 值 |
| VPN | 0x01 |
| TLB索引 | 0x01 |
| TLB 标记 | 0x00 |
| TLB 命中？（是/否） | 是 |
| 缺页？（是/否） | 否 |
| PPN | - |

C:物理地址格式：访问页面未分配，触发错误 Page Fault

D:物理地址引用：访问页面未分配，触发错误 Page Fault

9.15

块大小：前部需要

|  |  |  |
| --- | --- | --- |
| 请求 | 块大小（十进制字节） | 块头部（十六机制） |
| malloc(3) | 8 | 0x9 |
| malloc(11) | 16 | 0x11 |
| malloc(20) | 24 | 0x19 |
| malloc(21) | 28 | 0x21 |

9.17

/\*

\* @author: luo ruixin

\* @since : 21/12/2018

\*/

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

#include "mm.h"

#include "memlib.h"

/\*

\* If NEXT\_FIT defined use next fit search, else use first fit search

\*/

#define NEXT\_FIT x

/\* Team structure \*/

team\_t team **=** **{**

#ifdef NEXT\_FIT

"implicit next fit"**,**

#else

"implicit first fit"**,**

#endif

"Luo Ruixin"**,** "1170300821"**,**

""**,** ""

**};**

/\* $begin mallocmacros \*/

/\* Basic constants and macros \*/

#define WSIZE 4 /\* word size (bytes) \*/

#define DSIZE 8 /\* doubleword size (bytes) \*/

#define CHUNKSIZE (1<<12) /\* initial heap size (bytes) \*/

#define OVERHEAD 8 /\* overhead of header and footer (bytes) \*/

#define MAX(x, y) ((x) > (y)? (x) : (y))

/\* Pack a size and allocated bit into a word \*/

#define PACK(size, alloc) ((size) | (alloc))

/\* Read and write a word at address p \*/

#define GET(p) (\*(size\_t \*)(p))

#define PUT(p, val) (\*(size\_t \*)(p) = (val))

/\* Read the size and allocated fields from address p \*/

#define GET\_SIZE(p) (GET(p) & ~0x7)

#define GET\_ALLOC(p) (GET(p) & 0x1)

/\* Given block ptr bp, compute address of its header and footer \*/

#define HDRP(bp) ((char \*)(bp) - WSIZE)

#define FTRP(bp) ((char \*)(bp) + GET\_SIZE(HDRP(bp)) - DSIZE)

/\* Given block ptr bp, compute address of next and previous blocks \*/

#define NEXT\_BLKP(bp) ((char \*)(bp) + GET\_SIZE(((char \*)(bp) -WSIZE)))

#define PREV\_BLKP(bp) ((char \*)(bp) - GET\_SIZE(((char \*)(bp) -DSIZE)))

/\* $end mallocmacros \*/

/\* Global variables \*/

static char **\***heap\_listp**;** /\* pointer to first block \*/

#ifdef NEXT\_FIT

static char **\***rover**;** /\* next fit rover \*/

#endif

/\* function prototypes for internal helper routines \*/

static void **\***extend\_heap**(**size\_t words**);**//

static void place**(**void **\***bp**,** size\_t asize**);**//

static void **\***find\_fit**(**size\_t asize**);**//

static void **\***coalesce**(**void **\***bp**);**//

static void printblock**(**void **\***bp**);**//

static void checkblock**(**void **\***bp**);**//

/\*

\* mm\_init - Initialize the memory manager

\*/

/\* $begin mminit \*/

int mm\_init**(**void**)** **{**

/\* create the initial empty heap \*/

**if** **((**heap\_listp **=** mem\_sbrk**(**4 **\*** WSIZE**))** **==** **NULL)**

**return** **-**1**;**

PUT**(**heap\_listp**,** 0**);** /\* alignment padding \*/

PUT**(**heap\_listp **+** WSIZE**,** PACK**(**OVERHEAD**,** 1**));** /\* prologue header \*/

PUT**(**heap\_listp **+** DSIZE**,** PACK**(**OVERHEAD**,** 1**));** /\* prologue footer \*/

PUT**(**heap\_listp **+** WSIZE **+** DSIZE**,** PACK**(**0**,** 1**));** /\* epilogue header \*/

heap\_listp **+=** DSIZE**;**

#ifdef NEXT\_FIT

rover **=** heap\_listp**;**

#endif

/\* Extend the empty heap with a free block of CHUNKSIZE bytes \*/

**if** **(**extend\_heap**(**CHUNKSIZE **/** WSIZE**)** **==** **NULL)**

**return** **-**1**;**

**return** 0**;**

**}**

/\* $end mminit \*/

/\*

\* mm\_malloc - Allocate a block with at least size bytes of payload

\*/

/\* $begin mmmalloc \*/

void **\***mm\_malloc**(**size\_t size**)** **{**

size\_t asize**;** /\* adjusted block size \*/

size\_t extendsize**;** /\* amount to extend heap if no fit \*/

char **\***bp**;**

/\* Ignore spurious requests \*/

**if** **(**size **<=** 0**)**

**return** **NULL;**

/\* Adjust block size to include overhead and alignment reqs. \*/

**if** **(**size **<=** DSIZE**)**

asize **=** DSIZE **+** OVERHEAD**;**

**else**

asize **=** DSIZE **\*** **((**size **+** **(**OVERHEAD**)** **+** **(**DSIZE **-** 1**))** **/** DSIZE**);**

/\* Search the free list for a fit \*/

**if** **((**bp **=** find\_fit**(**asize**))** **!=** **NULL)** **{**

place**(**bp**,** asize**);**

**return** bp**;**

**}**

/\* No fit found. Get more memory and place the block \*/

extendsize **=** MAX**(**asize**,** CHUNKSIZE**);**

**if** **((**bp **=** extend\_heap**(**extendsize **/** WSIZE**))** **==** **NULL)**

**return** **NULL;**

place**(**bp**,** asize**);**

**return** bp**;**

**}**

/\* $end mmmalloc \*/

/\*

\* mm\_free - Free a block

\*/

/\* $begin mmfree \*/

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**);**

**}**

/\* $end mmfree \*/

/\*

\* mm\_realloc - naive implementation of mm\_realloc

\*/

void **\***mm\_realloc**(**void **\***ptr**,** size\_t size**)** **{**

void **\***newp**;**

size\_t copySize**;**

**if** **((**newp **=** mm\_malloc**(**size**))** **==** **NULL)** **{**

printf**(**"ERROR: mm\_malloc failed in mm\_realloc\n"**);**

exit**(**1**);**

**}**

copySize **=** GET\_SIZE**(**HDRP**(**ptr**));**

**if** **(**size **<** copySize**)**

copySize **=** size**;**

memcpy**(**newp**,** ptr**,** copySize**);**

mm\_free**(**ptr**);**

**return** newp**;**

**}**

/\*

\* mm\_checkheap - Check the heap for consistency

\*/

void mm\_checkheap**(**int verbose**)** **{**

char **\***bp **=** heap\_listp**;**

**if** **(**verbose**)**

printf**(**"Heap (%p):\n"**,** heap\_listp**);**

**if** **((**GET\_SIZE**(**HDRP**(**heap\_listp**))** **!=** DSIZE**)**

**||** **!**GET\_ALLOC**(**HDRP**(**heap\_listp**)))**

printf**(**"Bad prologue header\n"**);**

checkblock**(**heap\_listp**);**

**for** **(**bp **=** heap\_listp**;** GET\_SIZE**(**HDRP**(**bp**))** **>** 0**;** bp **=** NEXT\_BLKP**(**bp**))** **{**

**if** **(**verbose**)**

printblock**(**bp**);**

checkblock**(**bp**);**

**}**

**if** **(**verbose**)**

printblock**(**bp**);**

**if** **((**GET\_SIZE**(**HDRP**(**bp**))** **!=** 0**)** **||** **!(**GET\_ALLOC**(**HDRP**(**bp**))))**

printf**(**"Bad epilogue header\n"**);**

**}**

/\* The remaining routines are internal helper routines \*/

/\*

\* extend\_heap - Extend heap with free block and return its block

pointer

\*/

/\* $begin mmextendheap \*/

static void **\***extend\_heap**(**size\_t words**)** **{**

char **\***bp**;**

size\_t size**;**

/\* Allocate an even number of words to maintain alignment \*/

size **=** **(**words **%** 2**)** **?** **(**words **+** 1**)** **\*** WSIZE **:** words **\*** WSIZE**;**

**if** **((**bp **=** mem\_sbrk**(**size**))** **==** **(**void **\*)** **-**1**)**

**return** **NULL;**

/\* Initialize free block header/footer and the epilogue header \*/

PUT**(**HDRP**(**bp**),** PACK**(**size**,** 0**));** /\* free block header \*/

PUT**(**FTRP**(**bp**),** PACK**(**size**,** 0**));** /\* free block footer \*/

PUT**(**HDRP**(**NEXT\_BLKP**(**bp**)),** PACK**(**0**,** 1**));** /\* new epilogue header \*/

/\* Coalesce if the previous block was free \*/

**return** coalesce**(**bp**);**

**}**

/\* $end mmextendheap \*/

/\*

\* place - Place block of asize bytes at start of free block bp

\* and split if remainder would be at least minimum block size

\*/

/\* $begin mmplace \*/

/\* $begin mmplace-proto \*/

static void place**(**void **\***bp**,** size\_t asize**)**

/\* $end mmplace-proto \*/

**{**

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**));**

**}**

**}**

/\* $end mmplace \*/

/\*

\* find\_fit - Find a fit for a block with asize bytes

\*/

static void **\***find\_fit**(**size\_t asize**)** **{**

#ifdef NEXT\_FIT

char **\***oldrover **=** rover**;**

**for** **(;** GET\_SIZE**(**HDRP**(**rover**))** **>** 0**;** rover **=** NEXT\_BLKP**(**rover**))**

**if** **(!**GET\_ALLOC**(**HDRP**(**rover**))** **&&** **(**asize **<=** GET\_SIZE**(**HDRP**(**rover**))))**

**return** rover**;**

**for** **(**rover **=** heap\_listp**;** rover **<** oldrover**;** rover **=** NEXT\_BLKP**(**rover**))**

**if** **(!**GET\_ALLOC**(**HDRP**(**rover**))** **&&** **(**asize **<=** GET\_SIZE**(**HDRP**(**rover**))))**

**return** rover**;**

**return** **NULL;**

#else

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;**

#endif

**}**

/\*

\* coalesce - boundary tag coalescing. Return ptr to coalesced block

\*/

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**)** **{** //case1

**return** bp**;**

**}** **else** **if** **(**prev\_alloc **&&** **!**next\_alloc**)** **{** //case2

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**)** **{** //case3

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** **{** //case4

size **+=** GET\_SIZE**(**HDRP**(**PREV\_BLKP**(**bp**)))** **+**

GET\_SIZE**(**FTRP**(**NEXT\_BLKP**(**bp**)));**

PUT**(**HDRP**(**PREV\_BLKP**(**bp**)),** PACK**(**size**,** 0**));**

PUT**(**FTRP**(**NEXT\_BLKP**(**bp**)),** PACK**(**size**,** 0**));**

bp **=** PREV\_BLKP**(**bp**);**

**}**

**return** bp**;**

**}**

static void printblock**(**void **\***bp**)** **{**

size\_t hsize**,** halloc**,** fsize**,** falloc**;**

hsize **=** GET\_SIZE**(**HDRP**(**bp**));**

halloc **=** GET\_ALLOC**(**HDRP**(**bp**));**

fsize **=** GET\_SIZE**(**FTRP**(**bp**));**

falloc **=** GET\_ALLOC**(**FTRP**(**bp**));**

**if** **(**hsize **==** 0**)** **{**

printf**(**"%p: EOL\n"**,** bp**);**

**return;**

**}**

printf**(**"%p: header: [%d:%c] footer: [%d:%c]\n"**,** bp**,**

hsize**,** **(**halloc **?** 'a' **:** 'f'**),**

fsize**,** **(**falloc **?** 'a' **:** 'f'**));**

**}**

static void checkblock**(**void **\***bp**)** **{**

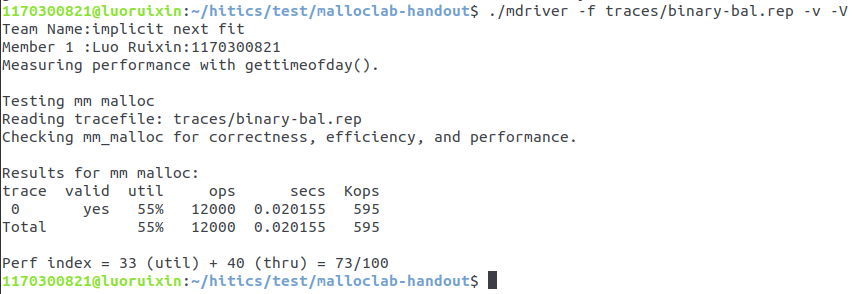
**if** **((**size\_t**)** bp **%** 8**)**

printf**(**"Error: %p is not doubleword aligned\n"**,** bp**);**

**if** **(**GET**(**HDRP**(**bp**))** **!=** GET**(**FTRP**(**bp**)))**

printf**(**"Error: header does not match footer\n"**);**

**}**



9.19

（1） a

a：正确。对于伙伴系统，如果要申请大小为33的空间，那么需要分配64个空间。如果申请大小为65的空间，那么块大小就需要128，所以最多可能有约50%的空间被浪费。

b：错误。最佳适配算法是要扫描所有的块，然后选择一个“浪费最少”的块，而首次匹配 算法从开始的块扫描，只要扫描到的块比要放入的数据大就放置，所以相比而言，首次适配 算法比最佳适配算法要快一些（平均）。

c：错误。P603 提出了使用 LIFO 的顺序维护链表，将新释放的块放置在链表的开始处。 使用 LIFO 的顺序和首次适配的放置策略，分配器会最先检查最近使用过的块，在这种情况 下，释放一个块可以在常数的时间内完成。所以不只有地址递增。使用边界标记的方法来回 收才会快速。

d：错误。，其实任何分配器都可能有外部碎片，只要剩余的空闲块大小和足够但是单个都不够，就会产生外部碎片。

（2）选 d

a:错误。块大小递减，首次适配很容易找到，所以分配性能会很高。

b:错误。应该按照块大小从小到大排会比较好。

c:错误。最佳匹配要选的是最终空闲空间最小的块。

d:正确。 块大小递增，那么最佳适配法找到的块和首次适配找到的块是同一个，因为最佳适配总是想找一个刚好大于请求块大小的空闲块。

（3）选 b

保守的意思就是所有可能被引用的堆都会被标记，int像指针，所以可能认为它表示的地址是正在被引用的（实际上它只是个int）。