15-213 背诵 Malloc

第二部分

您的助教

2022 年 3 月 21 日星期一

期中反馈

- 花 5 分钟填写 Piazza 上的匿名反馈表
- 请尽可能诚实，因为我们会在下半学期尝试做出合理调整

物流

- Malloc 实验室检查点于**明天**晚上 11:59 到期
- Malloc 实验室期末考试截止日期为 **3 月 29 日（星期二）**晚上 11:59
- 期末成绩的 7%（检查点 +4%）。
- 风格很重要！不要让你的努力付诸东流。
  - 有许多不同的实施方法，助教需要了解您的实施方法背后的细节。

- 检查点代码审查报名截止时间：**周四**晚上 11:59

议程

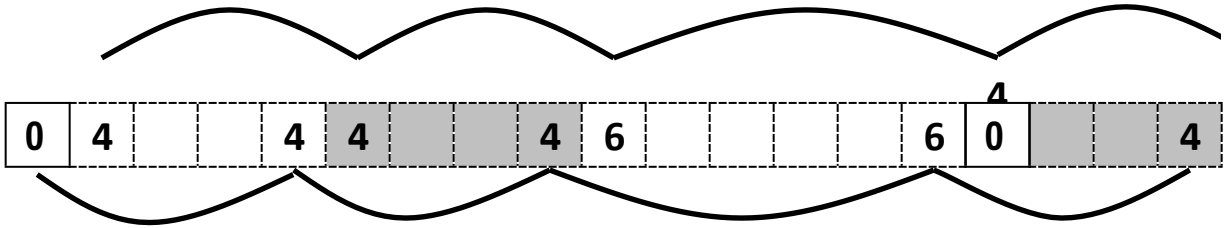
- 物流
- Malloc 实验室
- 检查点审查
- 活动 1
- 附录

了解您的代码

- 勾画出堆
- 添加仪器
- 使用工具

勾画出堆

- 从一个堆开始，这里是隐式列表



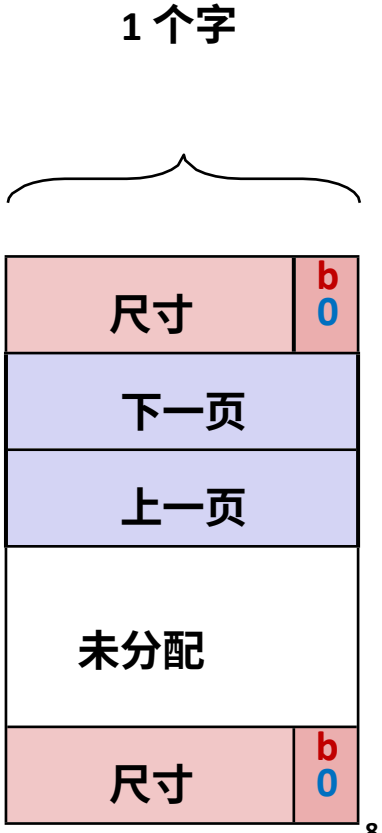
- 现在试试看，在这种情况下，extend_heap

```
block_t *block =
payload_too_header(bp);

write_block(block,size,false);

// 创建新的尾声标题
block_t *block_next = find_next(block);
```

```
write_epilogue(block_next);
```

勾画出堆

- **以下是基于第 19 和 20 课的自由组块**
  - 显式指针（将会定义明确，见撰写和 Piazza）
    - **这适用于您希望在结构体中添加的所有新字段**
  - 可选的边界标记



- **如果您对设计进行的更改超出了**
  - 画出来。
  - 如果您有虫子，图片可以帮助工作人员帮助您
  - 在文件头中添加数据结构的图片

    (可有可无，但我们会印象深刻)

免费

区块

常见问题

- **吞吐量非常低**
  - 哪项操作的吞吐量可能最大?
  - 提示：它使用循环!
  - 解决方案： ?

常见问题

- **吞吐量非常低**
  - 哪项操作的吞吐量可能最大?
  - 提示：它使用循环!
  - 解决方案：检测你的代码!

- **利用率非常低/内存不足**
  - 哪些操作会导致分配的内存超出需要?
  - 提示：它可以扩展您的内存容量!
  - 解决方案： ?

常见问题

- **吞吐量非常低**
  - 哪项操作的吞吐量可能最大?
  - 提示：它使用循环!
  - 解决方案：检测你的代码!
- **利用率非常低/内存不足**
  - 哪些操作会导致分配的内存超出需要?
  - 提示：它可以扩展您的内存容量!
  - 解决方案：检测你的代码!

添加仪器

- **请记住，测量结果可以为洞察力提供依据。**
  - 添加临时代码以了解 malloc 的各个方面
  - 代码可以违反样式规则或 128 字节限制，因为它是临时的
- **在做出改变之前深入了解绩效尤为重要**
  - 什么是昂贵的吞吐量?
  - 改变对利用率有多大益处?

添加仪器示例

- 在 `find_fit` 中搜索通常是最慢的步骤

- 您的代码效率如何？你怎么知道？
  - 计算查看的区块与调用的比率

```
static block_t *find_fit(size_t asize)
{
    block_t *block   call_count++;
    ;
    for (block = heap_listp; get_size(block) > 0;
                             block = find_next(block))
    {   block_count++;
        if (!(get_alloc(block)) && (asize <= get_size(block)))
        {
```

```
            返回块;
        }
    }
    return NULL; // 没有找到合适的
}
```

添加仪器 续

- **请求的规模有多大?**
  - 多少个 8 字节或更少?
  - 有多少个 16 字节或更少?
  - 还有什么尺寸?
- **你还能测量什么? 为什么?**
- **请记住，尽管系统性能各不相同**
  - mdriver 的轨迹是确定的

- 两次运行之间测量结果不应发生变化

使用工具

- **使用 mm_checkheap()**
  - 如果还没写，就写吧
  - 添加新功能时添加新不变式
  - 知道如何使用堆检查器。
    - 为什么需要堆检查器？两个原因
- **使用 gdb**
  - 你可以随时在 gdb 中调用 print 或 mm_checkheap。无需添加大量 printf。
  - 在崩溃时提供有用信息，如回溯信息。

- 编写只从 GDB 调用的打印空闲列表的辅助函数

写出自己的痕迹！

- 编写测试 malloc 和 free 简单序列的简短跟踪程序
- 阅读痕迹目录中的 README 文件和痕迹任务中的写法，了解需要如何编写痕迹文件

mdriver-emulate

- 测试 64 位地址空间

- 使用大小正确的掩码、常量和其他变量
- 小心大小类型之间的减法（可能导致下溢/溢出）
  - 注意：除此之外，还有许多其他问题。
- 在 mm_init 中重新初始化指针

乱码

- **Malloc 库返回一个数据块**
  - mdriver 将字节写入有效负载（使用 memcpy）
  - mdriver 将检查这些字节是否仍然存在
  - 如果 malloc 库覆盖了任何字节，则报告乱码字节
    - 还能检查其他类型的错误

- **现在怎么办?**

- **mm_checkheap 调用正在捕获它，对吗?**

- 如果没有，我们想找到乱码地址并观看它

乱码字节 GDB 和合同

- 拿出笔记本电脑

- 登录鲨鱼机
- wget http://www.cs.cmu.edu/~213/activities/rec9.tar
- tar -xvf rec9.tar
- cd rec9

- mm.c 是一个假的隐式列表实现。
  - 源代码基于 mm.c 启动代码

GDB 和合同练习

- 首先，让我们不带合同和 gdb 运行
- ./mdriver -c ./traces/syn-struct-short.rep

(输出示例)

```
ERROR [trace ./traces/syn-struct-short.rep, line 16]: block 1
(at 0x8000000a0) has 8 garbled bytes, starting at byte 16
ERROR [trace ./traces/syn-struct-short.rep, line 21]: block 4
(在 0x800000180 处）有 8 个乱码字节，从字节 16 开始
```

通过运行跟踪文件 "traces/syn-struct-short.rep"，完

成了正确性检查。

=> 不正确。

因 2 次错误而终止

在 GDB 中使用观察点

- gdb --args ./mdriver-dbg1 -c ./traces/syn-struct-short.rep
- 第一个出现乱码的地址是什么?
  - 使用 gdb watch 查找乱码发生的时间和原因。

```
(gdb) watch *0x8000000a0
(gdb) run
```

// 继续穿过休息区:

// write_block()

// 4 x memcpy

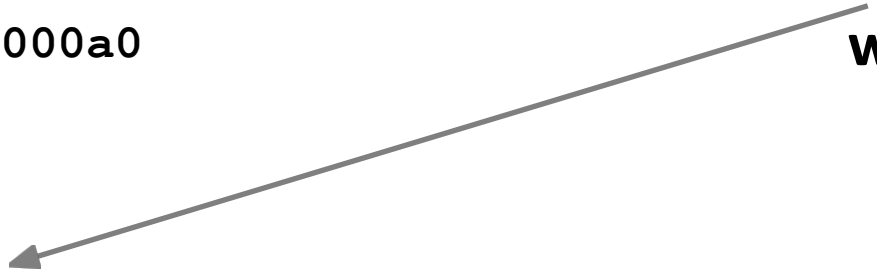
硬件监控点 1: *0x8000000a0

旧值 = 129 新值 =

32

```
write_block() at
mm.c:333
```

- 告诉我们要更仔细地查看 write_block()



我们在改写了

合同行使 续

- 现在让我们看看，当我们使用带有合同的文件时，会发生什么情况
  - ./mdriver-dbg2 -c ./traces/syn-struct-short.rep

```
mdriver-dbg: mm.c:331: void write_block(block_t *, size_t, _Bool)：断言
(unsigned long)footerp < ((long)block + size)' 失败。终止（核心
转储）
```

- 合同在第 331 行失败，这让我们对问题的根源有了更好的了解
- 打开 mm.c，尝试找出导致合约失败的原因

- **编写有效的合同可以节省大量的调试时间！**

使用我们工具的提示

- 使用 -D 选项运行 mdriver，以尽早发现乱码。使用 -V 运行它，以找出导致错误的跟踪。
- 需要注意的是，有时在最初的几次分配中就会出现错误。如果是这样，可以为 mm_malloc / mm_free 设置一个断点，然后逐行检查。
- 打印出局部变量，让自己相信它们的值是正确的。
- 对于 mdriver-emulate，仍可使用 `mem_read(address,8)`，而不是 `x /gx`，从模拟

的 64 位地址空间读取内存。

风格

- **条理清晰的代码更容易调试，也更容易评分！**
  - 模块化：尊重列表界面的辅助函数。
  - 文件：
    - 文件头：描述所有实施细节，包括块结构。
  - 代码结构：
    - 尽量不使用指针运算。
    - 在适当的情况下，用循环代替条件句。
  - 使用 git！
    - 确保经常***提交和推送***，并撰写描述性的提交信息

MallocLab

- **下周二到期**
- **期末成绩的 7%（检查点 + 4）**
  - 风格很重要！不要让你的努力付诸东流。
  - 有许多不同的实施方法，助教需要了解您的实施方法背后的细节。
- **阅读文章。它甚至列出了如何提高内存利用率的技巧。**
- **阅读 Piazza 上发布的 malloc 路线图**
- **橡皮鸭法**

- 如果你向一只橡皮鸭解释你的功能是什么

  你可能会在解释的过程中发现漏洞。

- 还记得上次背诵的 "调试思维过程 "幻灯片吗?