Malloc Lab Report

胡译文
2021201719
May 3, 2023

Contents

Abstrct

我们基于 `mm-implicit.c`，首先在空闲内存块中插入空闲指针，从而实现了一个高吞吐量的基于 `Explicit List` 的内存管理系统；然后再通过一个 `Segregated List` 和一个可以复用内存的 `mm_realloc` 函数进一步提升了内存利用率。

Solutions

实验的分数由两部分组成，一项是**内存利用率**（util），另一项是**吞吐量**（thru）。是否高效复用内存决定了内存利用率，查找算法的效率决定了吞吐量。我们的优化从隐式链表入手，主要分两步改进了程序。

Implicit list

我们在 `x86-64` 的机器上直接运行隐式链表的结果会发生错误。这是由于 `32` 位系统和 `64` 位系统在某些变量类型的长度不同导致的，具体而言，`size_t` 类型（一般来说本质上是 `unsigned long` 类型）和指针类型的不同会导致结果的不同。

在修复好 `mm-implicit.c` 在 `64` 位机器上的运行后，我们得到一个由较为不错的内存利用率和较低的吞吐量组成的结果：

```
Perf index = 44 (util) + 7 (thru) =
50/100
```

要优化吞吐量，必须改进查找空闲内存块的策略。一个自然的想法是采用 `Segregated List`。

Improving throughput

`Segregated List` 对于吞吐量的优化主要体现在：（1）显式链表查询，和（2）特定大小查询。显示链表可以跳过非空闲块，加速查询步骤；而通过分离不同大小范围的空闲块，可以更快找到目标大小的块。

优化后结果如下：

```
Results for mm malloc:
trace valid  util     ops      secs  Kops
 0      yes   98%    5694  0.000228 24952
 1      yes   97%    5848  0.000257 22737
 2      yes   98%    6648  0.000345 19258
 3      yes   99%    5380  0.000310 17344
 4      yes   66%   14400  0.000394 36520
 5      yes   93%    4800  0.000353 13613
 6      yes   90%    4800  0.000390 12308
 7      yes   55%   12000  0.000360 33296
 8      yes   51%   24000  0.001190 20160
 9      yes   25%   14401  0.056748   254
10      yes   28%   14401  0.013233  1088
Total          73%  112372  0.073810  1522

Perf index = 44 (util) + 40 (thru) =
84/100
```

我们可以看到，最后几个 trace 的利用率依然很低，他们的特点是包含较多的 `realloc` 函数。

Improving utilization rate

要进一步提升内存利用率，主要的点在于复用内存，即在 realloc 的时候不会总是重新分配并拷贝数据，而是试图直接在原位置延伸内存长度。

```
Results for mm malloc:
trace valid  util     ops      secs  Kops
 0      yes   98%    5694  0.000230 24800
 1      yes   97%    5848  0.000276 21188
 2      yes   98%    6648  0.000292 22791
 3      yes   99%    5380  0.000295 18250
 4      yes   66%   14400  0.000395 36428
 5      yes   93%    4800  0.000576  8330
 6      yes   90%    4800  0.000352 13636
 7      yes   55%   12000  0.000443 27070
 8      yes   51%   24000  0.001060 22642
 9      yes   93%   14401  0.000352 40877
10      yes   22%   14401  0.000377 38209
Total         78%  112372  0.004648 24176

Perf index = 47 (util) + 40 (thru) =
87/100
```

Hyperparameters optimization

最后涉及到一些优化调参：更小的内存池大小 CHUNKSIZE = (1<<9) 可以进一步提高内存利用率。

```
Perf index = 48 (util) + 40 (thru) =
88/100c
```

DEBUGGING

我们使用 x86-64 的 Ubuntu 虚拟机进行测试，并在 ics.ruc.rvalue.moe 服务器最终运行。

Compilation

我们使用了如下的最终发布版和调试版的编译指令。

- Release 编译命令：

```
cp mm-hyw.c mm.c && make clean && make
&& ./mdriver -V
```

- Debug 编译命令：

```
cp mm-hyw.c mm.c && make clean && make
CFLAGS="-g -gdwarf-2 -g3 -DDEBUG" &&
sudo gdb --args ./mdriver "-V"
```

为了使 GDB 能正确地显示源代码（layout src）、解析宏并执行调试代码，我们在编译的过程中添加了额外的参数 -g -gdwarf-2 -g3 -DDEBUG。

Debug tricks

在接下来的部分中，我们介绍一些调试中用到的技巧，对于定位错误位置有所帮助。

- 在调试中，我们使用了一种陷阱，使得 GDB 可以在捕获异常的同时不发生跳转：

```
printf("Bad epilogue header %d\n",
*nullp);
```

其中 nullp 是一个值为 NULL 的 int* 指针。

- 两个常用的函数为 mem_heap_lo() 和 mem_heap_hi()，用于检查分配堆的边界。
- 为了便于调试，我们在关键函数中嵌入了计数器，配合 GDB 的断点功能可以方便地复现故障代码位置。

```
#ifdef DEBUG
static int cnt = 0;
cnt++;
#endif
```

- 我们还扩展了 mm_explicit.c 中的堆检查器，主要修复了 FTRP(bp) 宏在 epilogue 处返回错误的结果的问题，使得堆检查器可以对 epilogue 进行检查。

Performance analysis

最后使用 perf 进行性能分析

```
0.66%  mdriver  [.] mm_malloc
0.62%  mdriver  [.] coalesce
0.52%  mdriver  [.] regist
0.51%  mdriver  [.] unregist
0.40%  mdriver  [.] place
0.35%  mdriver  [.] mm_free
0.32%  mdriver  [.] eval_mm_sp
0.05%  mdriver  [.] read_trace
0.05%  mdriver  [.] mm_realloc
0.05%  mdriver  [.] extend_hea
0.03%  mdriver  [.] mem_heap_l
0.03%  mdriver  [.] mem_sbrk
0.01%  mdriver  [.] malloc@plt
```

达到 Segregated Lis 的性能瓶颈。