第 15 章

1. 用种子 1、2 和 3 运行，并计算进程生成的每个虚拟地址是处于界限内还是界限外？如果在界限内，请计算地址转换

```
[redacted]   vm-mechanism % ./relocation.py -s 1

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x0000363c (decimal 13884)
  Limit  : 290

Virtual Address Trace
  VA  0: 0x0000030e (decimal:  782) --> PA or segmentation violation?
  VA  1: 0x00000105 (decimal:  261) --> PA or segmentation violation?
  VA  2: 0x000001fb (decimal:  507) --> PA or segmentation violation?
  VA  3: 0x000001cc (decimal:  460) --> PA or segmentation violation?
  VA  4: 0x0000029b (decimal:  667) --> PA or segmentation violation?
```

VA 0: 782 > 290，超出界限；

VA 1: 261 < 261，在界限内，物理地址 = 虚拟地址 + 基地址 = 13884 + 261 = 14145

VA 2: 507 > 290，超出界限；

VA 3: 460 > 290，超出界限；

VA 4: 667 > 290，超出界限；

```
[redacted]   vm-mechanism % ./relocation.py -s 2

ARG seed 2
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003ca9 (decimal 15529)
  Limit  : 500

Virtual Address Trace
  VA  0: 0x00000039 (decimal:   57) --> PA or segmentation violation?
  VA  1: 0x00000056 (decimal:   86) --> PA or segmentation violation?
  VA  2: 0x00000357 (decimal:  855) --> PA or segmentation violation?
  VA  3: 0x000002f1 (decimal:  753) --> PA or segmentation violation?
  VA  4: 0x000002ad (decimal:  685) --> PA or segmentation violation?
```

VA 0: 57 < 500，在界限内，物理地址 = 虚拟地址 + 基地址 = 15529 + 57 = 15586;

VA 1: 86 < 500，在界限内，物理地址 = 虚拟地址 + 基地址 = 15529 + 86 = 15615;

VA 2: 855 > 500，超出界限；

VA 3: 753 > 500，超出界限；

VA 4: 685 > 500，超出界限；

```
[redacted] vm-mechanism % ./relocation.py -s 3

ARG seed 3
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x000022d4 (decimal 8916)
  Limit  : 316

Virtual Address Trace
  VA  0: 0x0000017a (decimal:  378) --> PA or segmentation violation?
  VA  1: 0x0000026a (decimal:  618) --> PA or segmentation violation?
  VA  2: 0x00000280 (decimal:  640) --> PA or segmentation violation?
  VA  3: 0x00000043 (decimal:   67) --> PA or segmentation violation?
  VA  4: 0x0000000d (decimal:   13) --> PA or segmentation violation?
```

VA 0: 378 > 316，超出界限；
VA 1: 618 > 316，超出界限；
VA 2: 640 > 316，超出界限；
VA 3: 67 < 316，在界限内，物理地址 = 虚拟地址 + 基地址 = 8916 + 67 = 8983;
VA 4: 13 < 316，在界限内，物理地址 = 虚拟地址 + 基地址 = 8916 + 13 = 8929;

3．使用以下标志运行：-s 1 -n 10 -l 100．可以设置界限的最大值是多少，以便地址空间仍然完全放在物理内存中？

根据 README 中提示，程序设置的参数：-s 1（使用种子 1 运行），-n 10（生成的虚拟地址数量），-l 100（限制寄存器的值，即界限的最大值）。

```
[redacted] vm-mechanism % ./relocation.py -s 1 -n 10 -l 100

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00000899 (decimal 2201)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x00000363 (decimal:  867) --> PA or segmentation violation?
  VA  1: 0x0000030e (decimal:  782) --> PA or segmentation violation?
  VA  2: 0x00000105 (decimal:  261) --> PA or segmentation violation?
  VA  3: 0x000001fb (decimal:  507) --> PA or segmentation violation?
  VA  4: 0x000001cc (decimal:  460) --> PA or segmentation violation?
  VA  5: 0x0000029b (decimal:  667) --> PA or segmentation violation?
  VA  6: 0x00000327 (decimal:  807) --> PA or segmentation violation?
  VA  7: 0x00000060 (decimal:   96) --> PA or segmentation violation?
  VA  8: 0x0000001d (decimal:   29) --> PA or segmentation violation?
  VA  9: 0x00000357 (decimal:  855) --> PA or segmentation violation?
```

可以看到地址空间大小为 1k，物理内存大小为 16k，要使地址空间完全放在物理内存中，则界限的最大值应是 16 * 1024 - 100 = 16284。

第 17 章

1. 首先运行 flag -n 10 -H 0 -p BEST -s 0 来产生一些随机分配和释放。你能预测 malloc()/free()会返回什么吗？你可以在每次请求后猜测空闲列表的状态吗？随着时间的推移，你对空闲列表有什么发现？

```
[redacted] 'm-freespace % ./malloc.py flag -n 10 -H 0 -p BEST -s 0
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute False

ptr[0] = Alloc(3) returned ?
List?

Free(ptr[0])
returned ?
List?

ptr[1] = Alloc(5) returned ?
List?

Free(ptr[1])
returned ?
List?

ptr[2] = Alloc(8) returned ?
List?

Free(ptr[2])
returned ?
List?

ptr[3] = Alloc(8) returned ?
List?

Free(ptr[3])
returned ?
List?

ptr[4] = Alloc(2) returned ?
List?

ptr[5] = Alloc(7) returned ?
List?
```

运行可知参数：
堆的大小：100
堆的起始地址：1000
头块大小：0（不分配头块）
对齐参数 -1（即不对齐）
基本策略：最优匹配
列表顺序：按起始地址排列

是否合并：否
操作数量：10
每次分配空间的最大值：10
分配操作占比：50%
操作顺序：无参数，随机
计算结果：否
ptr[0] = Alloc(3) returned 1000
List [size 1]: [addr:1003 len:97]

Free(ptr[0])
returned 0
List [size 2]: [addr:1000 len 3] ----> [addr: 1003 len:97]

ptr[1] = Alloc(5) returned 1003
List [size 2]: [addr:1000 len 3] ----> [addr: 1008 len:92]

Free(ptr[1])
returned 0
List [size 3]: [addr:1000 len 3] ---->[addr: 1003 len:5] ----> [addr: 1008 len:92]

ptr[2] = Alloc(8) returned 1008
List [size 3]: [addr:1000 len 3] ---->[addr: 1003 len:5] ----> [addr: 1016 len:84]

Free(ptr[2])
returned 0
List [size 4]: [addr:1000 len 3] ---->[addr: 1003 len:5] ----> [addr: 1008 len:8]
----> [addr: 1016 len:84]

ptr[3] = Alloc(8) returned 1008
List [size 3]: [addr:1000 len 3] ---->[addr: 1003 len:5] ----> [addr: 1016 len:84]

Free(ptr[3])
returned 0
List [size 4]: [addr:1000 len 3] ---->[addr: 1003 len:5] ----> [addr: 1008 len:8]
----> [addr: 1016 len:84]

ptr[4] = Alloc(2) returned 1000
List [size 4]: [addr:1002 len 1] ---->[addr: 1003 len:5] ----> [addr: 1008 len:8]
----> [addr: 1016 len:84]

ptr[5] = Alloc(7) returned 1008
List [size 4]: [addr:1002 len 1] ---->[addr: 1003 len:5] ----> [addr: 1015 len:1]
----> [addr: 1016 len:84]

随着时间推移，因为 10 次操作都没有合并空闲列表，可以发现空闲列表形成的碎片与来越多。

3. 如果使用首次匹配（-p FIRST)会如何？使用首次匹配时，什么变快了？

```
[redacted] vm-freespace % ./malloc.py flag -n 10 -H 0 -p FIRST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

可以看到运行结果和最优匹配策略的运行结果一样，但最优匹配每次分配空间时都会遍历一次空闲列表在从中挑选最小的满足用户要求的块，而首次匹配只要找到空间足够大的快就会把空间返回给用户，所以使用首次匹配策略遍历的次数变少了，查找分配空间的时间变快了。

4. 对于上述问题，列表在保持有序时，可能会影响某些策略找到空闲位置所需的时间。使用不同的空闲列表排序（-l ADDRSORT,-l SIZESORT +,-l SIZESORT -)查看策略和列表排序如何相互影响。

列表排序方式：按起始地址大小

```
[redacted] vm-freespace % ./malloc.py flag -n 10 -H 0 -p FIRST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

列表排序方式：按空闲块大小递增排序

```
[redacted] vm-freespace % ./malloc.py flag -n 10 -H 0 -p FIRST -s 0 -l SIZESORT+ -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder SIZESORT+
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

列表排序方式：按空闲块大小递减排序

```
vm-freespace % ./malloc.py flag -n 10 -H 0 -p FIRST -s 0 -l SIZESORT- -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder SIZESORT-
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1003 sz:97 ][ addr:1000 sz:3 ]

ptr[1] = Alloc(5) returned 1003 (searched 1 elements)
Free List [ Size 2 ]: [ addr:1008 sz:92 ][ addr:1000 sz:3 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1008 sz:92 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[2] = Alloc(8) returned 1008 (searched 1 elements)
Free List [ Size 3 ]: [ addr:1016 sz:84 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1016 sz:84 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[3] = Alloc(8) returned 1016 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1024 sz:76 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[3])
returned 0
Free List [ Size 5 ]: [ addr:1024 sz:76 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[4] = Alloc(2) returned 1024 (searched 1 elements)
Free List [ Size 5 ]: [ addr:1026 sz:74 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[5] = Alloc(7) returned 1026 (searched 1 elements)
Free List [ Size 5 ]: [ addr:1033 sz:67 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]
```

在分配和释放操作依次交替执行时，三种排序方式对最优匹配策略的影响不大，但三种排序都会影响 free 的时间，因为需要按照特定排序方式插入空闲块；

按照空闲块大小排序可以减少首次匹配策略分配空间的时间，但按照递减排序会产生更多空间碎片；

按照空闲块大小递减排序方式会让最差适应算法搜索更快。