操作系统-hw2

班级：人工智能2103

学号：202107030125

姓名：姚丁钰

第15章

15.1

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。

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main
章-机制:地址转换$ ./relocation.py -s 1 -c

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) --> SEGMENTATION VIOLATION
  VA  1: 0x00000105 (decimal:  261) --> VALID: 0x00003741 (decimal: 14145)
  VA  2: 0x000001fb (decimal:  507) --> SEGMENTATION VIOLATION
  VA  3: 0x000001cc (decimal:  460) --> SEGMENTATION VIOLATION
  VA  4: 0x0000029b (decimal:  667) --> SEGMENTATION VIOLATION
```

> Base的值为13884，limit的值为290
> 对于VA 0其地址值为782，大于limit的值290，故无法成功转换；
> 对于VA 1其地址值为261，小于limit的值290，将其加上Base的值13884为14145即为对应的物理内存中的地址；
> 对于VA 2其地址值为507，大于limit的值290，故无法成功转换；
> 对于VA 3其地址值为460，大于limit的值290，故无法成功转换；
> 对于VA 4其地址值为667，大于limit的值290，故无法成功转换；

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main
章-机制:地址转换$ ./relocation.py -s 2 -c

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) --> VALID: 0x00003ce2 (decimal: 15586)
  VA  1: 0x00000056 (decimal:   86) --> VALID: 0x00003cff (decimal: 15615)
  VA  2: 0x00000357 (decimal:  855) --> SEGMENTATION VIOLATION
  VA  3: 0x000002f1 (decimal:  753) --> SEGMENTATION VIOLATION
  VA  4: 0x000002ad (decimal:  685) --> SEGMENTATION VIOLATION
```

> Base的值为15529，limit的值为500
> 对于VA 0其地址值为57，小于limit的值500，将其加上Base的值15529为15586即为对应的物理内存中的地址；
> 对于VA 1其地址值为86，小于limit的值500，将其加上Base的值15529为15615即为对应的物理内存中的地址；
> 对于VA 2其地址值为855，大于limit的值500，故无法成功转换；
> 对于VA 3其地址值为753，大于limit的值500，故无法成功转换；
> 对于VA 4其地址值为685，大于limit的值500，故无法成功转换；

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main
章-机制:地址转换$ ./relocation.py -s 3 -c

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) --> SEGMENTATION VIOLATION
  VA  1: 0x0000026a (decimal:  618) --> SEGMENTATION VIOLATION
  VA  2: 0x00000280 (decimal:  640) --> SEGMENTATION VIOLATION
  VA  3: 0x00000043 (decimal:   67) --> VALID: 0x00002317 (decimal: 8983)
  VA  4: 0x0000000d (decimal:   13) --> VALID: 0x000022e1 (decimal: 8929)
```

> Base的值为8916，limit的值为316
> 对于VA 0其地址值为378，大于limit的值316，故无法成功转换；
> 对于VA 1其地址值为618，大于limit的值316，故无法成功转换；
> 对于VA 2其地址值为640，大于limit的值316，故无法成功转换；
> 对于VA 3其地址值为67，小于limit的值316，将其加上Base的值8916为8983即为对应的物理内存中的地址；
> 对于VA 4其地址值为13，小于limit的值316，将其加上Base的值8916为8929即为对应的物理内存中的地址。

15.3

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?

注:原文为 Base 而不是 limit,因此这里要求的是基址,而不是界限寄存器大小

根据README中的规则可知，模拟程序的内存大小为16K，那么按照地址转换的规则，基址寄存器的最大值应该为：16*1024-100=16284

> 使用命令./relocation.py -s 1 -n 10 -l 100 -b 16284尝试是否可以分配成功

```
cs18@games101vm: ~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/15.第十五章-机制:地址转换
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main章-机制:地址转换$ ./relocation.py -s 1 -n 10 -l 100 -b 16284

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

Base-and-Bounds register information:

  Base   : 0x00003f9c (decimal 16284)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x00000089 (decimal:  137) --> PA or segmentation violation?
  VA  1: 0x00000363 (decimal:  867) --> PA or segmentation violation?
  VA  2: 0x0000030e (decimal:  782) --> PA or segmentation violation?
  VA  3: 0x00000105 (decimal:  261) --> PA or segmentation violation?
  VA  4: 0x000001fb (decimal:  507) --> PA or segmentation violation?
  VA  5: 0x000001cc (decimal:  460) --> PA or segmentation violation?
  VA  6: 0x0000029b (decimal:  667) --> PA or segmentation violation?
  VA  7: 0x00000327 (decimal:  807) --> PA or segmentation violation?
  VA  8: 0x00000060 (decimal:   96) --> PA or segmentation violation?
  VA  9: 0x0000001d (decimal:   29) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
```

> 使用命令./relocation.py -s 1 -n 10 -l 100 -b 16285尝试是否可以分配成功

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/15.第十五章-机制:地址转换
$ ./relocation.py -s 1 -n 10 -l 100 -b 16285

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

Base-and-Bounds register information:

  Base   : 0x00003f9d (decimal 16285)
  Limit  : 100

Error: address space does not fit into physical memory with those base/bounds values.
Base + Limit: 16385   Psize: 16384
```

可以看到这样的话会导致错误，并提示我们Base+Limit的值为16385

第16章

16.1

1.先让我们用一个小地址空间来转换一些地址。这里有一组简单的参数和几个不同的随机种子。你可以转换这些地址吗?

```
segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0
segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1
segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2
```

> 输入命令./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0

```
cs18@games101vm: ~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/16.第十六章-分段
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTE/16.第十六章-分段$
./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512
 -L 20 -s 0
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> PA or segmentation violation?
  VA  1: 0x00000061 (decimal:   97) --> PA or segmentation violation?
  VA  2: 0x00000035 (decimal:   53) --> PA or segmentation violation?
  VA  3: 0x00000021 (decimal:   33) --> PA or segmentation violation?
  VA  4: 0x00000041 (decimal:   65) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment 0
grows in the positive direction, whereas segment 1 in the negative.
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/16.第十六章-
$ ./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c
ARG seed 0
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  1: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x00000035 (decimal:   53) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000021 (decimal:   33) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000041 (decimal:   65) --> SEGMENTATION VIOLATION (SEG1)
```

```
VA  0: 0x0000006c (decimal:  108) --> sPA(段1)(物理地址492)
VA  1: 0x00000061 (decimal:   97) --> segmentation violation(段1)
VA  2: 0x00000035 (decimal:   53) --> segmentation violation(段0)
VA  3: 0x00000021 (decimal:   33) --> segmentation violation(段0)
VA  4: 0x00000041 (decimal:   65) --> segmentation violation(段1)
```

> 虚拟地址空间大小为128，物理地址空间大小为512

> 同时物理空间被分成了两个段，其中第一个段segmentation 0的基址为0，大小为20，正向增长；第二个段segmentation 1的基址为512，大小为20，反向增长。
>
> 由于有两个段，分析后将虚拟地址转化的二进制的最高两位作为段的判断，00表示在segmentation 0而 01表示在segmentation 1
>
> 对于VA 0（108）：将其转化为二进制为：01101100，段(01)，偏移量(101100即-20)
> 故其对应segmentation 1这个段，物理地址为512+(-20)=492；
> 对于VA 1（97）：将其转化为二进制为：01100001，段(01)，偏移量(100001即-31)
> 由于段的大小为20，故超出了限制，段错误；
> 对于VA 2（53）：将其转化为二进制为：00110101，段(00)，偏移量(110101即-11)
> segmentation 0为正向增长的段，偏移量不能为负，段错误；
> 对于VA 3（33）：将其转化为二进制为：00100001，段(00)，偏移量(100001即-31)
> segmentation 0为正向增长的段，偏移量不能为负，段错误；
> 对于VA 4（65）：将其转化为二进制为：01000001，段(01)，偏移量(000001即1)
> segmentation 1为反向增长的段，偏移量不能为正，段错误；

> 输入命令./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/16.第十六章-
$ ./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1
ARG seed 1
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> PA or segmentation violation?
  VA  1: 0x0000006c (decimal:  108) --> PA or segmentation violation?
  VA  2: 0x00000061 (decimal:   97) --> PA or segmentation violation?
  VA  3: 0x00000020 (decimal:   32) --> PA or segmentation violation?
  VA  4: 0x0000003f (decimal:   63) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment 0
grows in the positive direction, whereas segment 1 in the negative.
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/16.第十六章-
$ ./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 1 -c
ARG seed 1
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> VALID in SEG0: 0x00000011 (decimal:   17)
  VA  1: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  2: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  3: 0x00000020 (decimal:   32) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x0000003f (decimal:   63) --> SEGMENTATION VIOLATION (SEG0)
```

```
  VA  0: 0x00000011 (decimal:   17) --> PA(段0)(物理地址:17)
  VA  1: 0x0000006c (decimal:  108) --> PA(段1)(物理地址:492)
  VA  2: 0x00000061 (decimal:   97) --> segmentation violation(段1)
  VA  3: 0x00000020 (decimal:   32) --> segmentation violation(段0)
  VA  4: 0x0000003f (decimal:   63) --> segmentation violation(段0)
```

> 对于VA 0（17）：将其转化为二进制为：00010001，段(00)，偏移量(010001即17)
> 故其对应segmentation 0这个段，物理地址为：0+17=17；
> 对于VA 1（108）：将其转化为二进制为：01101100，段(01)，偏移量(101100即-20)
> 故其对应segmentation 1这个段，物理地址为512+(-20)=492；
> 对于VA 2（97）：将其转化为二进制为：01100001，段(01)，偏移量(100001即-31)
> 由于段的大小为20，故超出了限制，段错误；
> 对于VA 3（32）：将其转化为二进制为：00100000，段(00)，偏移量(100001即-32)
> segmentation 0为正向增长的段，偏移量不能为负，段错误；
> 对于VA 4（63）：将其转化为二进制为：00111111，段(00)，偏移量(111111即-1)
> segmentation 0为正向增长的段，偏移量不能为负，段错误；

> 输入命令./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/16.第十六章-
$ ./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2
ARG seed 2
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> PA or segmentation violation?
  VA  1: 0x00000079 (decimal:  121) --> PA or segmentation violation?
  VA  2: 0x00000007 (decimal:    7) --> PA or segmentation violation?
  VA  3: 0x0000000a (decimal:   10) --> PA or segmentation violation?
  VA  4: 0x0000006a (decimal:  106) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple address space with two segments: the top
bit of the virtual address can thus be used to check whether the virtual address
is in segment 0 (topbit=0) or segment 1 (topbit=1). Note that the base/limit pairs
given to you grow in different directions, depending on the segment, i.e., segment 0
grows in the positive direction, whereas segment 1 in the negative.
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/16.第十六章-
$ ./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 2 -c
ARG seed 2
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> VALID in SEG1: 0x000001fa (decimal:  506)
  VA  1: 0x00000079 (decimal:  121) --> VALID in SEG1: 0x000001f9 (decimal:  505)
  VA  2: 0x00000007 (decimal:    7) --> VALID in SEG0: 0x00000007 (decimal:    7)
  VA  3: 0x0000000a (decimal:   10) --> VALID in SEG0: 0x0000000a (decimal:   10)
  VA  4: 0x0000006a (decimal:  106) --> SEGMENTATION VIOLATION (SEG1)
```

```
Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> PA(段1)(物理地址:506)
  VA  1: 0x00000079 (decimal:  121) --> PA(段1)(物理地址:505)
  VA  2: 0x00000007 (decimal:    7) --> PA(段0)(物理地址:7)
  VA  3: 0x0000000a (decimal:   10) --> PA(段0)(物理地址:10)
  VA  4: 0x0000006a (decimal:  106) --> segmentation violation(段1)
```

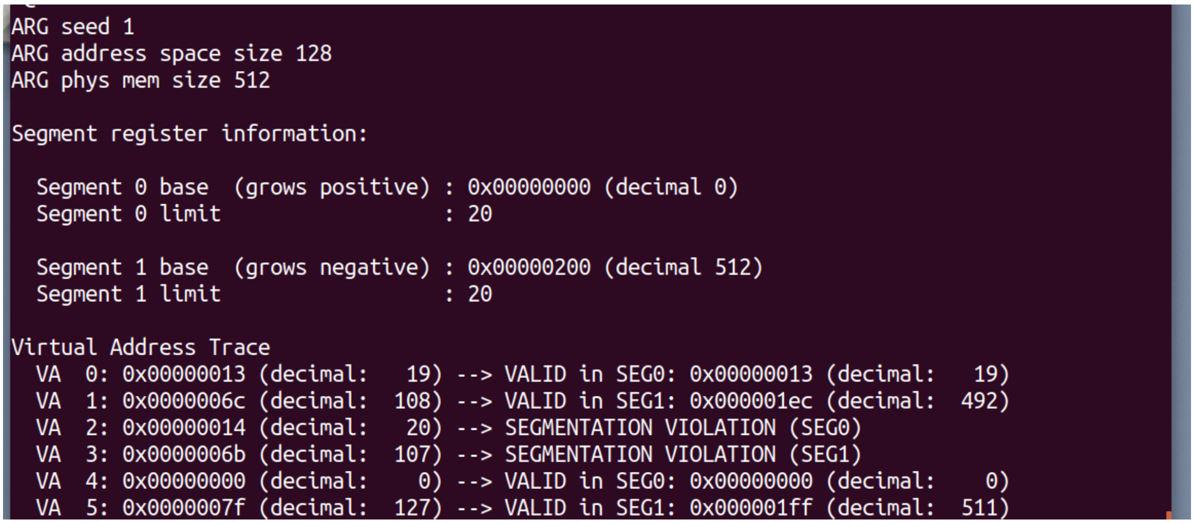
> 对于VA 0（122）：将其转化为二进制为：01111010，段(01)，偏移量(111010即-6)
> 故其对应segmentation 1这个段，物理地址为：512+(-6)=506；
> 对于VA 1（121）：将其转化为二进制为：01111001，段(01)，偏移量(111001即-7)
> 故其对应segmentation 1这个段，物理地址为：512+(-7)=505；

> 对于VA 2（7）：将其转化为二进制为：00000111，段(00)，偏移量(000111即7)
> 故其对应segmentation 0这个段，物理地址为：0+7=7；
> 对于VA 3（10）：将其转化为二进制为：00001010，段(00)，偏移量(001010即10)
> 故其对应segmengtation 0这个段，物理地址为：0+10=10；
> 对于VA 4（106）：将其转化为二进制为：01101010，段(01)，偏移量(101010即-22)
> 由于段的大小为20，故超出了限制，段错误；

16.2

现在,让我们看看是否理解了这个构建的小地址空间(使用上面问题的参数)段 0 中最高的合法虚拟地址是什么?段 1 中最低的合法虚拟地址是什么?在整个地址空间中,最低和最高的非法地址是什么?最后,如何运行带有 A 标志的 segmentation.py 来测试你是否正确?

> 段0 最高的合法虚拟地址 19   # 因为地址是从0开始的
> 段1 最低的合法虚拟地址 108
> 最高非法地址 107
> 最低非法地址 20



```
ARG seed 1
ARG address space size 128
ARG phys mem size 512

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                  : 20

  Segment 1 base  (grows negative) : 0x00000200 (decimal 512)
  Segment 1 limit                  : 20

Virtual Address Trace
  VA  0: 0x00000013 (decimal:   19) --> VALID in SEG0: 0x00000013 (decimal:   19)
  VA  1: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:  492)
  VA  2: 0x00000014 (decimal:   20) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x0000006b (decimal:  107) --> SEGMENTATION VIOLATION (SEG1)
  VA  4: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x00000000 (decimal:    0)
  VA  5: 0x0000007f (decimal:  127) --> VALID in SEG1: 0x000001ff (decimal:  511)
```

16.3

假设我们在一个 128 字节的物理内存中有一个很小的 16 字节地址空间。你会设置什么样的基址和界限,以便让模拟器为指定的地址流生成以下转换结果:有效,有效,违规,违规,有效,有效?假设用以下参数:

```
segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 ? --l0 ? --b1 ? --l1 ?
```

注: 原书问题为:valid, valid, violation, ..., violation, valid, valid,即要求 0,1,14,15 有效其余无效

--b0 指定段 0 基址寄存器值

--l0 指定段 0 界限寄存器值

只有前两个和最后两个是有效的，那么界限寄存器肯定是2。
基址寄存器的选择范围很大， b0可以从0到124，b1可以从4到128。

那么只需确保两个段的size为2，同时二者的基址相差至少为4即可

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/16.第十六章-
$ ./segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 110 --l0 2
 --b1 114 --l1 2 -c
ARG seed 0
ARG address space size 16
ARG phys mem size 128

Segment register information:

  Segment 0 base  (grows positive) : 0x0000006e (decimal 110)
  Segment 0 limit                  : 2

  Segment 1 base  (grows negative) : 0x00000072 (decimal 114)
  Segment 1 limit                  : 2

Virtual Address Trace
  VA  0: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x0000006e (decimal:  110)
  VA  1: 0x00000001 (decimal:    1) --> VALID in SEG0: 0x0000006f (decimal:  111)
  VA  2: 0x00000002 (decimal:    2) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000003 (decimal:    3) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000004 (decimal:    4) --> SEGMENTATION VIOLATION (SEG0)
  VA  5: 0x00000005 (decimal:    5) --> SEGMENTATION VIOLATION (SEG0)
  VA  6: 0x00000006 (decimal:    6) --> SEGMENTATION VIOLATION (SEG0)
  VA  7: 0x00000007 (decimal:    7) --> SEGMENTATION VIOLATION (SEG0)
  VA  8: 0x00000008 (decimal:    8) --> SEGMENTATION VIOLATION (SEG1)
  VA  9: 0x00000009 (decimal:    9) --> SEGMENTATION VIOLATION (SEG1)
  VA 10: 0x0000000a (decimal:   10) --> SEGMENTATION VIOLATION (SEG1)
  VA 11: 0x0000000b (decimal:   11) --> SEGMENTATION VIOLATION (SEG1)
  VA 12: 0x0000000c (decimal:   12) --> SEGMENTATION VIOLATION (SEG1)
  VA 13: 0x0000000d (decimal:   13) --> SEGMENTATION VIOLATION (SEG1)
  VA 14: 0x0000000e (decimal:   14) --> VALID in SEG1: 0x00000070 (decimal:  112)
  VA 15: 0x0000000f (decimal:   15) --> VALID in SEG1: 0x00000071 (decimal:  113)
```

第17章

17.1

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？

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main
章-空闲空间管理$ ./malloc.py -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?
```

> 可以看到，模拟程序模拟了一块大小为100的空间，基地址为1000，头部的大小为0，同时采用最优匹配的匹配策略。

> 第一次分配大小为3的空间，将会返回1000，此时List有一个块大小为97
> free之后将会返回0，此时List有两个块大小分别为3、97；
>
> 第二次分配大小为5的空间，将会返回1003，此时List有两个块大小分别为3、92
> free之后将会返回0，此时List有三个块大小分别为3、5、92
>
> 第三次分配大小为8的空间，将会返回1008，此时List有三个块大小为3、5、84
> free之后将会返回0，此时List有四个块大小分别为3、5、8、84
>
> 第四次分配大小为8的空间，根据最优分配的规则，将会返回1008，此时List有三个块大小为3、5、84，free之后将会返回0，此时List有四个块大小分别为3、5、8、84
>
> 第五次分配大小为2的空间，根据最优分配，将会返回1000，此时List有四个块大小分别为1、5、8、84
>
> 第六次分配大小为7的块，根据最优分配，将会返回1008，此时List有四个块大小分别为1、5、1、84

> 由于没有合并,空闲空间碎片越来越多

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main章-空闲空间管理$ ./mall
oc.py -n 10 -H 0 -p BEST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

17.3

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？

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main章-空闲空间管理$ ./mal
loc.py -n 10 -H 0 p FIRST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

> 使用FIRST分配策略进行分配时searched的元素将会减少，也就是说由于首次分配只需要找到第一个足够大的空闲块即可，不需要像最优分配那样每次遍历所有的空闲块，所以在查找空闲块花费的时间将会大大减少。

17.4

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。

> 对比不同分配策略、不同空闲列表排序方式下的分配情况可以看出：
> 不同的空闲列表排序方式对最优(BEST)和最差(WORST)分配这种分配策略的效率没有影响
> 而对于像首次(FIRST)分配这样的分配策略，不同的空闲列表排序方式将会影响其分配策略的效率
> 这与分配策略的具体思路是相关的：最优和最差分配无论空闲块如何排列，每次分配都需要遍历所有的空闲块，而首次分配只需要找到首个足够大的空闲块，所以与空闲块的排列有关。

- BEST分配策略：

```
vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-理$ ./malloc.py -n 10 -H 0 p
 BEST -s 0 -l ADDRSORT -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ .
/malloc.py -n 10 -H 0 p BEST -s 0 -l SIZESORT+ -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ .
/malloc.py -n 10 -H 0 p BEST -s 0 -l SIZESORT- -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 2 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 3 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 1008 (searched 4 elements)
Free List [ Size 3 ]:  [ addr:1016 sz:84 ] [ addr:1003 sz:5 ] [ addr:1000 sz:3 ]

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

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

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

- FIRST分配策略：

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ .
/malloc.py -n 10 -H 0 p FIRST -s 0 -l ADDRSORT -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ ./malloc.py -n 10 -H 0 p FIRST -s 0 -l SIZESORT+ -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ ./malloc.py -n 10 -H 0 p FIRST -s 0 -l SIZESORT- -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 2 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 3 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 1008 (searched 4 elements)
Free List [ Size 3 ]:  [ addr:1016 sz:84 ] [ addr:1003 sz:5 ] [ addr:1000 sz:3 ]

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

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

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

- WORST分配策略

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ ./malloc.py -n 10 -H 0 p WORST -s 0 -l ADDRSORT -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ ./malloc.py -n 10 -H 0 p WORST -s 0 -l SIZESORT+ -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 4 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 4 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 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [ addr:1016 sz:84 ]
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/17.第十七章-空闲空间管理$ ./malloc.py -n 10 -H 0 p WORST -s 0 -l SIZESORT- -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
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 2 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 3 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 1008 (searched 4 elements)
Free List [ Size 3 ]:  [ addr:1016 sz:84 ] [ addr:1003 sz:5 ] [ addr:1000 sz:3 ]

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

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

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

> 如果是最优匹配或者最差匹配：空闲列表的排序不影响搜索策略的搜索速度
>
> 如果是首次匹配：如果使用SIZESORT-匹配更快，但是碎片化更严重
> 如果使用SIZESORT+匹配稍慢，但是碎片化程度会低一点