İŞLETİM SİSTEMLERİ DERSİ

LABORATUVAR FÖYÜ

Dr. Halil ARSLAN

Laboratuvar -8

Bölütleme (Segmentation) & Boş Alan Yönetimi (Free Space Management)

İçindekiler

Öğrencinin

Adı Soyadı	:Berkehan Dağlayan
Okul No	:2021123048
Teslim Tarihi	:01.01.2024
İmza	:

Bölütleme (Segmentation)

Bu bölümde kullanılan açıklamalar kaynak dosya olarak eklenmiştir. Ödev ekinde bulunan **16_Segmentation_new.pptx** dosyasını inceleyebilirsiniz. Ayrıca ödevlerde kaynak kitap olarak kullanılan **Operating Systems: Three Easy Pieces** kitabının ilgili bölümünü (Chapter 16) buradan okuyabilirsiniz.

Ödev (Simülasyon)

segmentation.py[1] kod dosyası, segmentasyonlu bir sistemde adres çevirilerinin nasıl yapıldığını görmenizi sağlar. Bu sistemin kullandığı bölümleme oldukça basittir: bir adres alanının yalnızca iki bölümü vardır; ayrıca işlem tarafından oluşturulan sanal adresin en üst biti, adresin hangi segmentte olduğunu belirler: *Segment 0* için 0 (örneğin, kodun ve heap'in bulunacağı yer) ve *Segment 1* için 1 (stack'in bulunduğu yer). Segment 0 pozitif yönde (daha yüksek adreslere doğru) büyürken, segment 1 negatif yönde büyür.

Görsel olarak adres alanı şuna benzer:

Hatırlayacağınız gibi segmentasyonda, segment başına bir base/limit register çifti bulunur. Dolayısıyla bu problemde iki base/limit çifti vardır. Segment-0 tabanı (base), segment 0'ın üst kısmının fiziksel belleğe hangi fiziksel adresin yerleştirildiğini, limit ise segmentin ne kadar büyük olduğunu söyler; segment-1 tabanı, segment 1'in alt kısmının fiziksel hafızada nereye yerleştirildiğini söyler ve buna karşılık gelen limit de bize segmentin ne kadar büyük olduğunu (veya negatif yönde ne kadar büyüdüğünü) söyler.

Daha önce olduğu gibi, segmentasyon konusunu anladığınızı test etmek amacıyla programı çalıştırmanın iki adımı vardır. İlk olarak, bir dizi çeviri oluşturmak için "-c" bayrağı olmadan çalıştırın ve adres çevirilerini kendiniz doğru bir şekilde gerçekleştirip gerçekleştiremeyeceğinize bakın. Daha sonra işiniz bittiğinde cevaplarınızı kontrol etmek için "-c" bayrağıyla çalıştırın.

Örneğin, varsayılan bayraklarla çalıştırmak için şunu yazın:

```
prompt> ./segmentation.py
```

[1] https://github.com/salihayesilyurt/fall-2023-bil3003-os-hw.git reposunda yer alan hw8/vm-segmentation dizini içerisindeki segmentation.py dosyasını kullanınız. Bir Python dosyasının nasıl çalıştırılacağına dair detaylı bilgiler için Teams grubundaki videolara bakınız.

Bunu görmelisiniz:

```
ARG seed 0
  ARG address space size 1k
  ARG phys mem size 16k

  Segment register information:

    Segment 0 base  (grows positive) : 0x00001aea (decimal 6890)
    Segment 0 limit                  : 472

    Segment 1 base  (grows negative) : 0x00001254 (decimal 4692)
    Segment 1 limit                  : 450

  Virtual Address Trace
    VA  0: 0x0000020b (decimal:  523) --> PA or segmentation violation?
    VA  1: 0x0000019e (decimal:  414) --> PA or segmentation violation?
    VA  2: 0x00000322 (decimal:  802) --> PA or segmentation violation?
    VA  3: 0x00000136 (decimal:  310) --> PA or segmentation violation?
    VA  4: 0x000001e8 (decimal:  488) --> 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.
```

Daha sonra sanal adres izlemesindeki çevirileri hesapladıktan sonra programı "-c" bayrağıyla tekrar çalıştırın. Aşağıdakileri göreceksiniz (gereksiz bilgiler hariç):

```
Virtual Address Trace
    VA  0: 0x0000020b (decimal:  523) --> SEGMENTATION VIOLATION (SEG1)
    VA  1: 0x0000019e (decimal:  414) --> VALID in SEG0: 0x00001c88 (decimal:
7304)
    VA  2: 0x00000322 (decimal:  802) --> VALID in SEG1: 0x00001176 (decimal:
4470)
    VA  3: 0x00000136 (decimal:  310) --> VALID in SEG0: 0x00001c20 (decimal:
7200)
    VA  4: 0x000001e8 (decimal:  488) --> SEGMENTATION VIOLATION (SEG0)
```

Gördüğünüz gibi -c ile program sizin için adresleri çevirir ve böylece segmentasyon kullanan bir sistemin adresleri nasıl çevirdiğini anlayıp anlamadığınızı kontrol edebilirsiniz.

Elbette kendinize farklı problemlere oluşturmak için kullanabileceğiniz bazı parametreler var. Özellikle önemli parametrelerden biri, farklı bir rastgele seed göndererek farklı problemler oluşturmanıza olanak tanıyan -s veya -seed parametresidir. Tabi, bir problem oluştururken ve çözerken aynı seed'i kullandığınızdan emin olunuz.

Farklı büyüklükteki adres alanları ve fiziksel belleklerle oynamak için kullanabileceğiniz bazı parametreler de vardır. Örneğin, küçük bir sistemde segmentasyonu denemek için şunu yazabilirsiniz:

```
prompt> ./segmentation.py -s 100 -a 16 -p 32
ARG seed 0
ARG address space size 16
ARG phys mem size 32

Segment register information:

  Segment 0 base  (grows positive) : 0x00000018 (decimal 24)
  Segment 0 limit                  : 4

  Segment 1 base  (grows negative) : 0x00000012 (decimal 18)
  Segment 1 limit                  : 5

Virtual Address Trace
  VA  0: 0x0000000c (decimal:   12) --> PA or segmentation violation?
  VA  1: 0x00000008 (decimal:    8) --> PA or segmentation violation?
  VA  2: 0x00000001 (decimal:    1) --> PA or segmentation violation?
  VA  3: 0x00000007 (decimal:    7) --> PA or segmentation violation?
  VA  4: 0x00000000 (decimal:    0) --> PA or segmentation violation?
```

Bu, programa 32 baytlık fiziksel belleğin herhangi bir yerine yerleştirilmiş 16 baytlık adres alanı için sanal adresler oluşturmasını söyler. Gördüğünüz gibi ortaya çıkan sanal adresler çok küçüktür (12, 8, 1, 7 ve 0). Gördüğünüz gibi program, uygun olan küçük taban register'larını ve limit değerlerini seçiyor. Cevapları görmek için -c ile çalıştırınız.

Bu örnek aynı zamanda her base çiftinin tam olarak ne anlama geldiğini de gösterecektir. Örneğin, segment 0'ın tabanı 24'lük (ondalık) bir fiziksel adrese ayarlanmıştır ve boyutu 4 bayttır. Dolayısıyla, 0, 1, 2 ve 3 numaralı sanal adresler 0 segmentindedir ve geçerlidir ve sırasıyla 24, 25, 26 ve 27 numaralı fiziksel adreslerle eşleşir.

Negatif yönde büyüyen Segment 1 biraz daha zorlayıcıdır. Yukarıdaki küçük örnekte, Segment 1'in base register'ı, 5 bayt büyüklüğünde fiziksel adres 18'e ayarlanmıştır. Bu, sanal adres alanının son beş baytının (bu durumda 11, 12, 13, 14 ve 15) geçerli sanal adresler olduğu ve bunların sırasıyla 13, 14, 15, 16 ve 17 numaralı fiziksel adreslerle eşleştiği anlamına gelir.

"Kilobayt", "megabayt" ve "gigabayt" gibi -a veya -p işaretleriyle ilettiğiniz değerlerin üzerine "k", "m" ve hatta "g" işaretini ekleyerek daha büyük değerler belirtebilirsiniz. Dolayısıyla, 32 MB'lık bir fiziksel bellekte 1 MB'lık adres alanıyla bazı çeviriler yapmak istiyorsanız şunu yazabilirsiniz:

```
prompt> ./segmentation.py -a 1m -p 32m
```

Daha da spesifik olmak istiyorsanız, temel base ve limit register değerlerini --b0, --l0, --b1 ve --l1 register'ları ile kendiniz ayarlayabilirsiniz. Bunları deneyiniz ve görünüz.

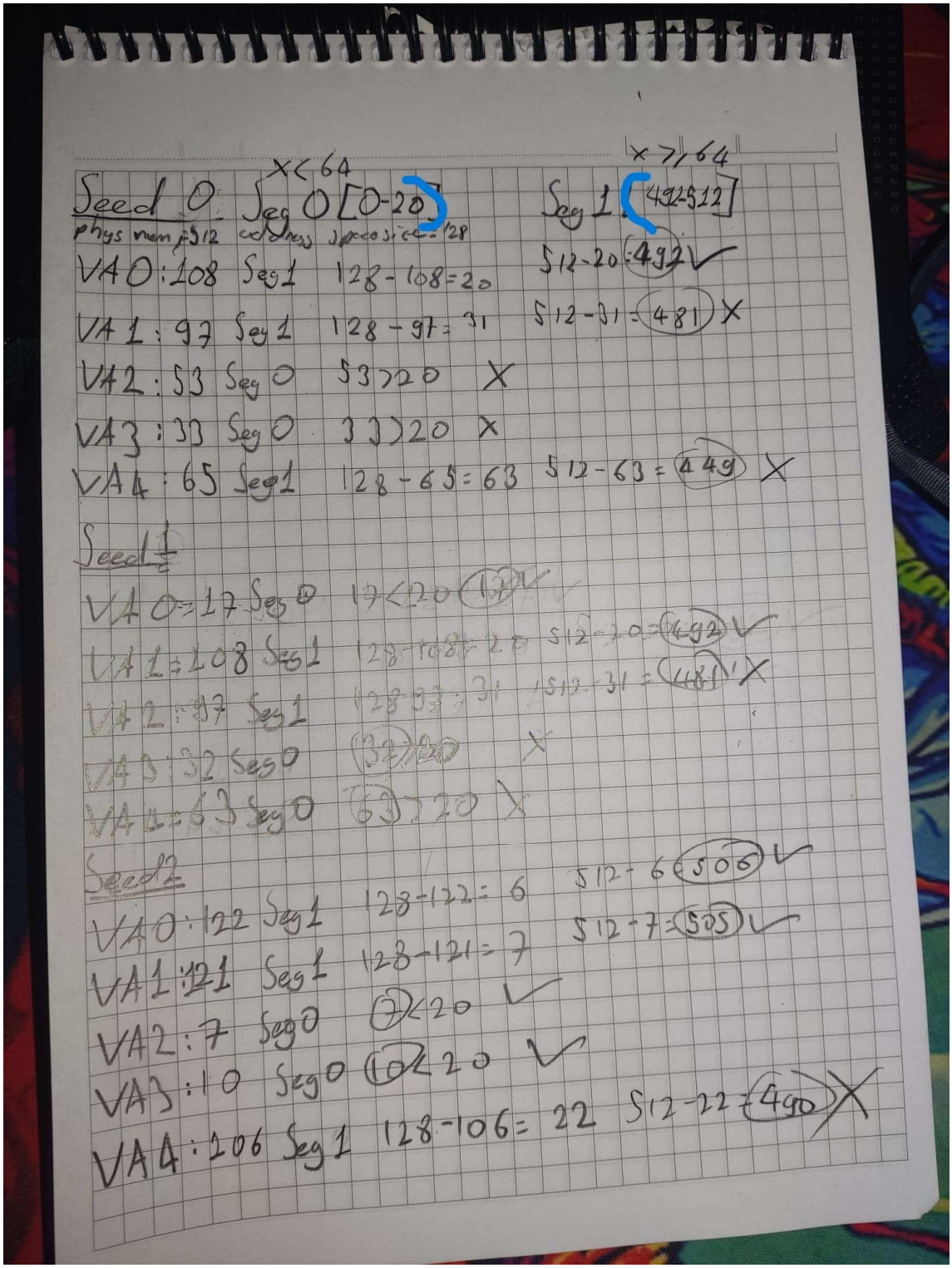
Bayrakların ve seçeneklerin tam listesini almak için.

```
prompt> ./segmentation.py -h
```


1. Öncelikle bazı adresleri çevirmek için küçük bir adres alanı kullanalım. İşte birkaç farklı seed'ler içeren basit bir parametre seti; adresleri tercüme edebilir misiniz?

```
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
```

/* *Adress Space Size*/2=x dersek; sanal adres x'ten büyük veya eşitse segment 1'de değilse segment 0'dadır. Segment 0 0-20 bayt , Segment 1 512-492 bayt aralığındadır. 3 soruda da aynı olduğu için hepsini buna bağlı hesaplayacağım.



*/

2. Şimdi (yukarıdaki sorudaki parametreleri kullanarak) oluşturduğumuz bu küçük adres alanını anlayıp anlamadığımızı görelim. Segment 0'daki en yüksek legal sanal adres nedir? Segment 1'deki en düşük legal sanal adrese ne dersiniz? Tüm bu adres alanındaki en düşük ve en yüksek *illegal*

adresler nelerdir? Son olarak, haklı olup olmadığınızı test etmek için `segmentation.py`'yi `-A` bayrağıyla nasıl çalıştırırsınız?

/* Segment 0'daki en yüksek legal sanal adres 19'dur. Segment 1'deki en düşük legal sanal adres 64'tür. Tüm bu adres alanındaki en düşük ve en yüksek illegal adresler 20 ve 128'dir. -A flag'ini 19,20,64,128

```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw8/vm-segmentation$ python3 segmentatic
n.py -a 128 -p 512 -b 0 -l 20 -B 512 -L 20 -s 0 -c -A 19,20,64,128
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: 0x00000013 (decimal:   19) --> VALID in SEG0: 0x00000013 (decimal:   19)
  VA  1: 0x00000014 (decimal:   20) --> SEGMENTATION VIOLATION (SEG0)
  VA  2: 0x00000040 (decimal:   64) --> SEGMENTATION VIOLATION (SEG1)
Error: virtual address 128 cannot be generated in an address space of size 128
```

*/

3. Diyelim ki 128 baytlık bir fiziksel bellekte 16 baytlık küçük bir adres alanımız var. Simülatörün belirtilen adres akışı için şu çeviri sonuçlarını üretmesini sağlamak amacıyla hangi temeli (base) ve sınırları (bound) kurarsınız: valid, valid, violation, ..., violation, valid, valid? Aşağıdaki parametreleri varsayalım:

```
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 ?
```

/* İlk 2 ve son 2 adresin valid diğerlerini violation olmasını sağlamamız gerekiyor. segmentation.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 0 --l0 2 --b1 126 --l1 2 komutunu kullandığımızda bunun gerçekleştiğini görüyoruz. Segment 0 için base değerini 0 limitini 2 olarak Segment 1 için base değerini 126 limit değerini 2 olarak ayarladığımızda 0 ve 1 için segment 0, 14 ve 15 için segment 1 olmasını sağladık. Limit değerlerini 2 seçtiğimiz için ilk 2 ve son 2 sayıyı aldık eğer ilk 3 ve son 3 istenseydi 3 vererek yapabilirdik. Aradaki değerler invalid oldu.


```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw8/vm-segmentation$ python3 segmentatio
n.py -a 16 -p 128 -A 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 --b0 0 --l0 2 --b1 126 --l1 2 -c
ARG seed 0
ARG address space size 16
ARG phys mem size 128

Segment register information:

  Segment 0 base  (grows positive) : 0x00000000 (decimal 0)
  Segment 0 limit                 : 2

  Segment 1 base  (grows negative) : 0x0000007e (decimal 126)
  Segment 1 limit                 : 2

Virtual Address Trace
  VA  0: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x00000000 (decimal:    0)
  VA  1: 0x00000001 (decimal:    1) --> VALID in SEG0: 0x00000001 (decimal:    1)
  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: 0x0000007c (decimal:  124)
  VA 15: 0x0000000f (decimal:   15) --> VALID in SEG1: 0x0000007d (decimal:  125)
```

*/

4. Rastgele oluşturulmuş sanal adreslerin kabaca %90'ının geçerli olduğu (bölümleme ihlallerinin (segmentation violations) olmadığı) bir problem oluşturmak istediğimizi varsayalım. Bunu yapmak için simülatörü nasıl yapılandırmalısınız? Bu sonucu elde etmek için hangi parametreler önemlidir?

   /* Segmentlerin Base ve Limit değerinin toplam kapladığı alan adres alanının yaklaşık %90'ını oluşturmalıdır. Adres alanı boyutu ve fiziksel bellek boyutunun yüksek olması işlemin başarılı olma olasılığını arttırır. -a,-p,--b0,--l0,--b1,--l1 parametrelerini kullanmamız gerekir.

   */

5. Simülatörü hiçbir sanal adresin geçerli olmayacağı şekilde çalıştırabilir misiniz? Nasıl?

   /* Her iki segmentin limitlerini 0 olarak belirlersek hiçbir sanal adres bu segmentler içine düşmez.


```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw8/vm-segmentation$ python3 segment
ation.py -a 128 -p 512 --b0 0 --l0 0 --b1 100 --l1 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                  : 0

  Segment 1 base  (grows negative) : 0x00000064 (decimal 100)
  Segment 1 limit                  : 0

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.
```

*/

Boş Alan Yönetimi (Free Space Management)

Bu bölümde kullanılan açıklamalar kaynak dosya olarak eklenmiştir. Ödev ekinde bulunan **17._FreeSpace_Management_new.pptx** dosyasını inceleyebilirsiniz. Ayrıca ödevlerde kaynak kitap olarak kullanılan **Operating Systems: Three Easy Pieces** kitabının ilgili bölümünü (Chapter 17) buradan okuyabilirsiniz.

Ödev (Simülasyon)

malloc.py[2] adlı bu program, basit bir bellek ayırıcının (memory allocator) nasıl çalıştığını görmenizi sağlar. Bazı seçenekler:

```
-h, --help            show this help message and exit
  -s SEED, --seed=SEED  the random seed
  -S HEAPSIZE, --size=HEAPSIZE
                        size of the heap
  -b BASEADDR, --baseAddr=BASEADDR
                        base address of heap
  -H HEADERSIZE, --headerSize=HEADERSIZE
                        size of the header
  -a ALIGNMENT, --alignment=ALIGNMENT
                        align allocated units to size; -1->no align
  -p POLICY, --policy=POLICY
                        list search (BEST, WORST, FIRST)
  -l ORDER, --listOrder=ORDER
                        list order (ADDRSORT, SIZESORT+, SIZESORT-, INSERT-FRONT,
INSERT-BACK)
  -C, --coalesce        coalesce the free list?
  -n OPSNUM, --numOps=OPSNUM
                        number of random ops to generate
  -r OPSRANGE, --range=OPSRANGE
                        max alloc size
  -P OPSPALLOC, --percentAlloc=OPSPALLOC
                        percent of ops that are allocs
  -A OPSLIST, --allocList=OPSLIST
                        instead of random, list of ops (+10,-0,etc)
  -c, --compute         compute answers for me
```

Bunu kullanmanın bir yolu, programın bazı rastgele tahsis/serbest işlemler üretmesini sağlamak ve sizin için free listenin nasıl görüneceğini ve ayrıca her işlemin başarısını veya başarısızlığını anlayıp çözemeyeceğinizi görmenizdir.

İşte basit bir örnek:

```
prompt> ./malloc.py -S 100 -b 1000 -H 4 -a 4 -l ADDRSORT -p BEST -n 5

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?
```

[2] https://github.com/salihayesilyurt/fall-2023-bil3003-os-hw.git reposunda yer alan hw8/vm-freespace dizini içerisindeki malloc.py dosyasını kullanınız. Bir Python dosyasının nasıl çalıştırılacağına dair detaylı bilgiler için Teams grubundaki videolara bakınız.

Bu örnekte, 1000 (-b 1000) adresinden başlayan, 100 bayt (-S 100) boyutunda bir heap belirtiyoruz. Tahsis edilen blok başına ilave 4 baytlık bir başlık belirtiyoruz (-H 4) ve tahsis edilen her alanın, boyut olarak en yakın 4 baytlık boş yığına (-a 4) yuvarlandığından emin oluyoruz. Free listenin adrese göre sıralı (artan) tutulmasını belirtiyoruz. Son olarak, "en uygun" serbest liste arama politikasını (-p BEST) belirliyoruz ve 5 rastgele işlemin (-n 5) oluşturulmasını istiyoruz. Bunu çalıştırmanın sonuçları yukarıda; İşiniz, her tahsis/serbest işlemin ne döndürdüğünü ve ayrıca her işlemden sonra free listenin durumunu bulmaktır.

Burada -c seçeneğini kullanarak sonuçlara bakıyoruz

```
prompt> ./malloc.py -S 100 -b 1000 -H 4 -a 4 -l ADDRSORT -p BEST -n 5 -c

ptr[0] = Alloc(3)  returned 1004 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1008 sz:92 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:8 ] [ addr:1008 sz:92 ]

ptr[1] = Alloc(5)  returned 1012 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:8 ] [ addr:1020 sz:80 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:8 ] [ addr:1008 sz:12 ] [ addr:1020 sz:80
]

ptr[2] = Alloc(8)  returned 1012 (searched 3 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:8 ] [ addr:1020 sz:80 ]

As you can see, the first allocation operation (an allocation) returns the
following information:

ptr[0] = Alloc(3)  returned 1004 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1008 sz:92 ]
```

Free listenin başlangıç durumu yalnızca büyük bir öğe olduğundan Alloc(3) isteğinin başarılı olacağını tahmin etmek kolaydır. Ayrıca, belleğin yalnızca ilk kısmını döndürecek ve geri kalanını free bir listeye dönüştürecektir. Döndürülen işaretçi başlığın hemen ötesinde olacaktır (adres:1004) ve tahsis edilen alan 4 bayta yuvarlanır ve serbest listede 1008'den başlayarak 92 bayt kalır

Bir sonraki işlem, önceki tahsis isteğinin sonuçlarını saklayan "ptr[0]" Free işlemidir. Tahmin edebileceğiniz gibi, bu Free başarılı olacak (böylece "0" değerini döndürecek) ve free liste artık biraz daha karmaşık görünüyor:

```
Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:8 ] [ addr:1008 sz:92 ]
```

Aslında, Free listeyi birleştirmediğimiz için artık üzerinde iki öğemiz var; ilki 8 bayt büyüklüğünde ve az önce döndürülen alanı tutuyor, ikincisi ise 92 baytlık chunk.

Gerçekten -C bayrağı aracılığıyla birleştirmeyi açabiliriz ve sonuç:

```
prompt> ./malloc.py -S 100 -b 1000 -H 4 -a 4 -l ADDRSORT -p BEST -n 5 -c -C
ptr[0] = Alloc(3)  returned 1004 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1008 sz:92 ]

Free(ptr[0]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]

ptr[1] = Alloc(5)  returned 1004 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1012 sz:88 ]

Free(ptr[1]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]

ptr[2] = Alloc(8)  returned 1004 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1012 sz:88 ]
```

Free işlemler gerçekleştiğinde free listesinin beklendiği gibi birleştirildiğini görebilirsiniz.

Keşfedilecek başka ilginç seçenekler de var.

- `-p BEST` veya `-p WORST` veya `-p FIRST`: Bu seçenek, bir ayırma isteği sırasında kullanılacak bir bellek yığınını (chunk) aramak için bu üç farklı stratejiyi kullanmanızı sağlar.
- `-l ADDRSORT` veya `-l SIZESORT+` veya `-l SIZESORT-` veya `-l INSERT-FRONT` veya `-l INSERT-BACK`: Bu seçenek, free listeyi belirli bir sırada tutmanıza olanak tanır; örneğin boş yığının adresine, boş boyutuna göre sıralanmış öbek (ya + ile artar ya da - ile azalır), ya da boş öbekleri boş listenin önüne (INSERT-FRONT) ya da arkasına (INSERT-BACK) döndürür.
- `-A list_of_ops`: Bu seçenek, rastgele oluşturulan istekler yerine tam bir istek dizisi belirtmenize olanak tanır. Örneğin, "-A +10, +10, +10,-0,-2" bayrağıyla çalıştırmak, 10 bayt boyutunda üç parça (artı başlık) ayıracak ve ardından ilkini ("-0") serbest bırakacaktır ve ardından üçüncüyü ("-2") serbest bırakır. O zaman free liste nasıl görünecek?

1. İlk olarak `-n 10 -H 0 -p BEST -s 0` işaretleriyle çalıştırarak birkaç rastgele tahsis ve serbest bırakma işlemi gerçekleştirin. alloc()/free() işlevinin ne döndüreceğini tahmin edebilir misiniz? Her istekten sonra free listenin durumunu tahmin edebilir misiniz? Zaman içinde free listeyle ilgili ne fark ettiniz?

/*alloc()/free() işlevinin ne döndüreceğini tahmin edebiliriz. Her istekten sonra free listin durumunu tahmin edebiliriz. ptr0'da 3 baytlık bir alan ayrılmış. 1000-1002 arası. Ptr0 serbest bırakılmış. Ptr1 ile 5 baytlık bir alan ayrılmış. 1003-1007. ptr1 serbest bırakılmış. Ptr2 ile 8 baytlık bir alan ayrılmış. 1008-1015 arası. Ptr2 serbest bırakılmış. Ptr3 ile 8 baytlık bir alan ayrılmış. 1008-1015 arası. Ptr3 serbest bırakılmış. Ptr 4 ile 2 baytlık bir alan ayrılmış. 1000-1001 arası. Ptr5 ile 7 baytlık bir alan ayrılmış 1007-1014 arası. Free list bir dolup bir boşalıyor. Eğer tahsil edimek alan büyükse ve daha önce boşaltılan alan bu alandan daha küçükse boşaltılan alan kullanılamayıp yeni alan tahsil ediliyor.

```
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 ]
```

*/

2. freelist arama yapmak için EN KÖTÜ uyum politikası kullanıldığında sonuçlar nasıl farklı olur (`-p WORST`)? Hangi değişiklikler?

/*WORST politikasında bellek tahsilinde en büyük serbest bellek blokları kullanılıyor. BEST politikasında kullanılabilirse küçük alanlar öncelikli. BEST politikasında 1000-1015 arası kullanılırken WORST politikası 1000-1032 arasını kullanmış.


```
size 100
baseAddr 1000
headerSize 0
alignment -1
policy WORST
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 1016 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1024 sz:76 ]

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

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

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

*/

3. FİRST fit (-p FIRST) kullanıldığında ne olacak? İlk fit'i kullandığınızda ne hızlanır?

/*FIRST politikası ilk serbest bulduğu bloğu kullanıyor. Bu da listenin başındaki blokların daha fazla kullanılmasını sağlıyor. Daha büyük blokların aranması olmadığı için arama hızı kısalır.


```
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 ]
```

*/

4. Yukarıdaki sorular için listenin nasıl sıralandığı, bazı poliçeler için boş yer bulma süresini etkileyebilir. Politikaların ve liste sıralamalarının nasıl etkileşimde bulunduğunu görmek için farklı serbest liste sıralamalarını (`-l ADDRSORT,-l SIZESORT+,-l SIZESORT-`) kullanınız.

/*Liste yaptım.

Fit Politikası / Liste Sıralaması	SIZESORT+ (Artan Boyut)	SIZESORT- (Azalan Boyut)	ADDRSORT (Adres)
BEST Fit	Küçük bloklar öncelikli, verimli küçük tahsisler. Büyük blokların hızla parçalanması.	Büyük bloklar öncelikli, büyük blokların hızla parçalanması. Küçük bloklar sonradan kullanılır.	En uygun boyutlu blok seçilir, adres sırasına göre ilerler. Küçük ve büyük bloklar dengeli kullanılır.
WORST Fit	En büyük bloğu bulmak için listenin sonuna kadar gidilir, tahsis süresi uzar.	En büyük bloklar hızlıca bulunur ve kullanılır, tahsis süresi kısalır.	En büyük bloğu bulmak için adres sırasını takip eder, genellikle listenin sonuna doğru büyük bloklar kullanılır.
FIRST Fit	Küçük bloklar öncelikli, verimli küçük tahsisler. `ADDRSORT` ile benzer sonuçlar.	Büyük bloklar öncelikli, ancak ilk uygun blok seçilir. `ADDRSORT` ile benzer sonuçlar.	İlk uygun bloğu seçer, genellikle serbest listenin başından başlar. Küçük ve büyük bloklar dengeli kullanılır.

*/

5. Free bir listenin birleştirilmesi oldukça önemli olabilir. Rastgele tahsislerin sayısını artırın (örneğin -n 1000'e kadar). Zaman içinde daha büyük tahsis isteklerine ne olur? Birleşmeyle ve birleştirme olmadan çalıştırın (yani, -C bayrağıyla ve olmadan). Sonuçlarda ne gibi farklılıklar görüyorsunuz? Her durumda free liste zaman içinde ne kadar büyük olur? Bu durumda listenin sırası önemli mi?

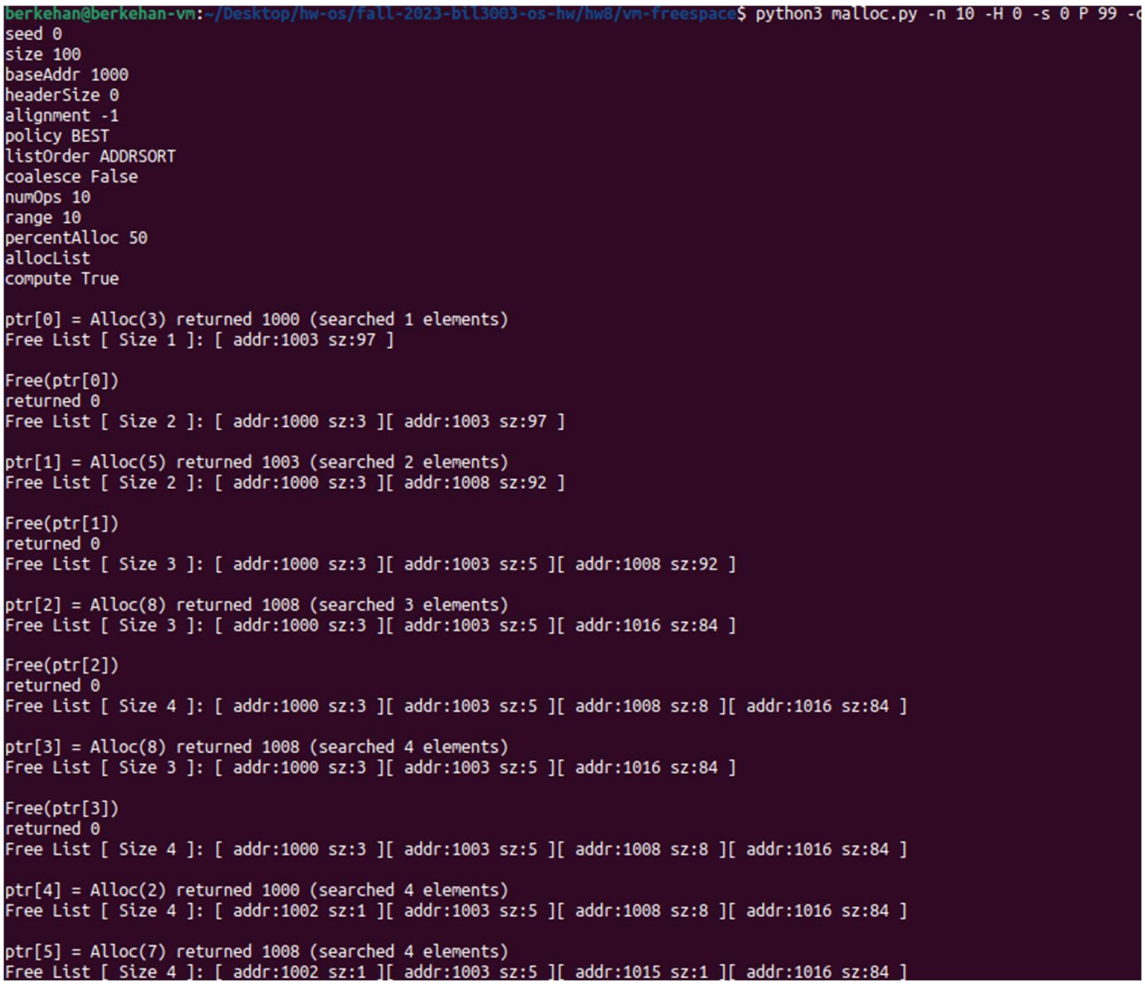
/* Birleştirme olmadan bellek parçalanması artar ve free list daha az kullanılabilir hale gelir. Birleştirme büyük bellek tahsillerinin daha etkin yapılmasını sağladığından büyük tahsis talepleri karşılanmayabilir. Free list birleştirme varken tek bir parça olarak kalıyor. Free list genel olarak büyüklüğü sabit kalır ancak birleştirme yapılarak daha az parçalı ve daha büyük bloklar içerebilir. Birleştirme varken listenin sırası birleştirme yokken olandan daha önem arz eder çünkü küçük blokların dolması free listi büyük bir blok olarak tutmaya yarar sağlar.

*/

6. Tahsis edilen yüzde kesri -P'yi 50'den yüksek bir değere değiştirdiğinizde ne olur? 100'e yaklaşıldığında tahsislere ne olacak? Yüzde 0'a yaklaştığında ne olacak?

/* Tahsis edilme miktarı arttıkça free list daha hızlı tükenir. 100’e yaklaştığında tahsis işlemi neredeyse her zaman olacağından free list hızla küçülür. Eğer tahsil edilme miktarı azaltılırsa free listin dolma ihtimali de azalır.


```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw8/vm-freespace$ python3 malloc.py -n 10 -H 0 -s 0 P 51 -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 ]
```


```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw8/vm-freespace$ python3 malloc.py -n 10 -H 0 -s 0 P 9 -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 ]
```

*/

7. Oldukça parçalanmış bir boş alan oluşturmak için ne tür spesifik taleplerde bulunabilirsiniz? Parçalanmış ücretsiz listeler oluşturmak için -A işaretini kullanın ve farklı politika ve seçeneklerin ücretsiz listenin organizasyonunu nasıl değiştirdiğini görün.

/* -C kullanılmayarak parçalanmayı maksimize edelim.

Çıktılara göre;

BEST politikasıyla SIZESORT- en az parçalanmaya yol açıyor ve SIZESORT+ en yüksek parçalanmaya yol açıyor.
WORST politikasıyla ADDRSORT en yüksek parçalanmaya yol açarken SIZESORT+ ve SIZESORT- aynı miktara yakın miktarda parçalanmaya yol açıyor.

FIRST politikasıyla ADDRSORT en yüksek parçalanmaya yol açarken SIZESORT+ ve SIZESORT- aynı miktara yakın miktarda parçalanmaya yol açıyor.

WORST en çok parçalanmaya yol açarken BEST en az parçalanmaya yol açmış

```
Running command 1: python3 malloc.py -n 1000 -H 0 -p BEST -l ADDRSORT -s 0 -P 70 -A +10,+20,+10,-1,+5,-0,+15 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 1000
range 10
percentAlloc 70
allocList +10,+20,+10,-1,+5,-0,+15
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(20) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1030 sz:70 ]

ptr[2] = Alloc(10) returned 1030 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1040 sz:60 ]

Free(ptr[1])
returned 0
Free List [ Size 2 ]: [ addr:1010 sz:20 ][ addr:1040 sz:60 ]

ptr[3] = Alloc(5) returned 1010 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1015 sz:15 ][ addr:1040 sz:60 ]

Free(ptr[0])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:10 ][ addr:1015 sz:15 ][ addr:1040 sz:60 ]

ptr[4] = Alloc(15) returned 1015 (searched 3 elements)
Free List [ Size 2 ]: [ addr:1000 sz:10 ][ addr:1040 sz:60 ]
```



```
Running command 2: python3 malloc.py -n 1000 -H 0 -p WORST -l ADDRSORT -s 0 -P 70 -A +10,+20,+10,-1,+5,-0,+15 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy WORST
listOrder ADDRSORT
coalesce False
numOps 1000
range 10
percentAlloc 70
allocList +10,+20,+10,-1,+5,-0,+15
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(20) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1030 sz:70 ]

ptr[2] = Alloc(10) returned 1030 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1040 sz:60 ]

Free(ptr[1])
returned 0
Free List [ Size 2 ]: [ addr:1010 sz:20 ][ addr:1040 sz:60 ]

ptr[3] = Alloc(5) returned 1040 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1010 sz:20 ][ addr:1045 sz:55 ]

Free(ptr[0])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:10 ][ addr:1010 sz:20 ][ addr:1045 sz:55 ]

ptr[4] = Alloc(15) returned 1045 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:10 ][ addr:1010 sz:20 ][ addr:1060 sz:40 ]
```

```
Running command 3: python3 malloc.py -n 1000 -H 0 -p FIRST -l ADDRSORT -s 0 -P 70 -A +10,+20,+10,-1,+5,-0,+15 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder ADDRSORT
coalesce False
numOps 1000
range 10
percentAlloc 70
allocList +10,+20,+10,-1,+5,-0,+15
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(20) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1030 sz:70 ]

ptr[2] = Alloc(10) returned 1030 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1040 sz:60 ]

Free(ptr[1])
returned 0
Free List [ Size 2 ]: [ addr:1010 sz:20 ][ addr:1040 sz:60 ]

ptr[3] = Alloc(5) returned 1010 (searched 1 elements)
Free List [ Size 2 ]: [ addr:1015 sz:15 ][ addr:1040 sz:60 ]

Free(ptr[0])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:10 ][ addr:1015 sz:15 ][ addr:1040 sz:60 ]

ptr[4] = Alloc(15) returned 1015 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:10 ][ addr:1040 sz:60 ]
```



```
Running command 4: python3 malloc.py -n 1000 -H 0 -p BEST -l SIZESORT+ -s 0 -P 70 -A +10,+20,+10,-1,+5,-0,+15 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder SIZESORT+
coalesce False
numOps 1000
range 10
percentAlloc 70
allocList +10,+20,+10,-1,+5,-0,+15
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(20) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1030 sz:70 ]

ptr[2] = Alloc(10) returned 1030 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1040 sz:60 ]

Free(ptr[1])
returned 0
Free List [ Size 2 ]: [ addr:1010 sz:20 ][ addr:1040 sz:60 ]

ptr[3] = Alloc(5) returned 1010 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1015 sz:15 ][ addr:1040 sz:60 ]

Free(ptr[0])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:10 ][ addr:1015 sz:15 ][ addr:1040 sz:60 ]

ptr[4] = Alloc(15) returned 1015 (searched 3 elements)
Free List [ Size 2 ]: [ addr:1000 sz:10 ][ addr:1040 sz:60 ]
```

```
Running command 5: python3 malloc.py -n 1000 -H 0 -p BEST -l SIZESORT- -s 0 -P 70 -A +10,+20,+10,-1,+5,-0,+15 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder SIZESORT-
coalesce False
numOps 1000
range 10
percentAlloc 70
allocList +10,+20,+10,-1,+5,-0,+15
compute True

ptr[0] = Alloc(10) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1010 sz:90 ]

ptr[1] = Alloc(20) returned 1010 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1030 sz:70 ]

ptr[2] = Alloc(10) returned 1030 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1040 sz:60 ]

Free(ptr[1])
returned 0
Free List [ Size 2 ]: [ addr:1040 sz:60 ][ addr:1010 sz:20 ]

ptr[3] = Alloc(5) returned 1010 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1040 sz:60 ][ addr:1015 sz:15 ]

Free(ptr[0])
returned 0
Free List [ Size 3 ]: [ addr:1040 sz:60 ][ addr:1015 sz:15 ][ addr:1000 sz:10 ]

ptr[4] = Alloc(15) returned 1015 (searched 3 elements)
Free List [ Size 2 ]: [ addr:1040 sz:60 ][ addr:1000 sz:10 ]
```

*/

