



# İŞLETİM SİSTEMLERİ DERSİ

## LABORATUVAR FÖYÜ

Dr. Halil ARSLAN



### Laboratuvar -7

#### Adres Uzayı (Memory Space) Soyutlaması & Adres Çevrimi (Address Translation)

#### İçindekiler

Adres Uzayı (Memory Space) Soyutlaması.....	2
Ödev (Simülasyon).....	3
Adres Çevrimi (Address Translation).....	10
Ödev (Simülasyon).....	11

#### Öğrencinin

Adı Soyadı :Berkehan Dağlayan

Okul No :2021123048

Teslim Tarihi :27.12.2023

İmza :

## Adres Uzayı (Memory Space) Soyutlaması

Bu bölümde kullanılan açıklamalar kaynak dosya olarak eklenmiştir. Ödev ekinde bulunan [13\\_AddressSpaces\\_new.pptx](#) dosyasını inceleyebilirsiniz. Ayrıca ödevlerde kaynak kitap olarak kullanılan **Operating Systems: Three Easy Pieces** kitabının ilgili bölümünü (Chapter 13) [buradan](#) okuyabilirsiniz.

## Ödev (Simülasyon)

Bu ödevde, Linux tabanlı sistemlerde sanal bellek kullanımını incelemek için birkaç yararlı araç öğreneceğiz. Bu sadece neyin mümkün olduğuna dair kısa bir ipucu olacaktır; gerçek bir uzman olmak için kendi başınızda daha derinlere dalmanız gerekecek (her zaman olduğu gibi!).

1. Kontrol etmeniz gereken ilk Linux aracı, çok basit bir araçtır. Öncelikle `man free` yazın ve kılavuz sayfasının tamamını okuyun; kısa, endişelenmeyin!

*/\*Sistemdeki RAM kullanım durumunu ve swap alanını gösteriyor bu da bize ramin yeterli olup olmadığını gösterir.*

*\*/*

2. Şimdi, belki yararlı olabilecek bazı bağımsız değişkenleri kullanarak (örneğin, `-m`, bellek toplamlarını megabayt cinsinden görüntülemek için) `free` komutunu çalıştırın. Sisteminizde ne kadar bellek var? Ne kadar boş? Bu sayılar sezgilerinizle eşleşiyor mu?

*/\*Kolay okunabilmesi için -h seçeneğini kullandım. VM'de Linux'a 8 GB atamıştim. 6,5 GB'ı boş imiş. Windows gibi rami yormadığı için beklenilen bir şeydi.*

```
berkehan@berkehan-vm:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:       7,7Gi       941Mi      4,9Gi       42Mi       1,9Gi      6,5Gi
Swap:      2,0Gi          0B      2,0Gi
```

*\*/*

3. Daha sonra, belli miktarda bellek kullanan, `memory-user.c` adı verilen küçük bir program oluşturun. Bu program, kullanacağı megabayt bellek sayısını argüman olarak almalıdır. Çalıştırıldığında, bir dizi tahsis etmeli ve her girişe dokunarak dizi boyunca sürekli akış sağlamalıdır. Program bunu süresiz olarak veya belki de komut satırında belirtilen belirli bir süre boyunca yapmalıdır.

*/\*İnternetten malloc komutu ile bellekte yer tahsis eden bir uygulama arakladım. İlk kısımda MB ikinci kısımda ise ne kadar süre aktif kalacağını (veya yazılmazsa süresiz bir şekilde.) girebiliyoruz.*

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(int argc, char *argv[]) {
6     if (argc < 2) {
7         fprintf(stderr, "Usage: %s <Memory in MB> [Duration in seconds]\n", argv[0]);
8         return 1;
9     }
10
11    int memorySizeMB = atoi(argv[1]);
12    int duration = (argc >= 3) ? atoi(argv[2]) : -1;
13
14    // Calculate the size of the array
15    size_t arraySize = memorySizeMB * 1024 * 1024;
16    char *array = malloc(arraySize);
17
18    if (array == NULL) {
19        fprintf(stderr, "Failed to allocate memory.\n");
20        return 1;
21    }
22
23    // Continuously loop through the array
24    while (duration != 0) {
25        for (size_t i = 0; i < arraySize; i++) {
26            array[i] += 1; // Touch each entry
27        }
28
29        if (duration > 0) {
30            duration--;
31        }
32
33        sleep(1); // Wait for 1 second
34    }
35
36    free(array);
37    return 0;
38}

```

\*/

- Önemli olarak gördüğünüz notları yazınız. Şimdi, `memory-user.c` programınızı çalıştırırken aynı zamanda (farklı bir terminal penceresinde, ancak aynı makinede) `free` toolu'nu da çalıştırınız. Programınız çalışırken bellek kullanım toplamları nasıl değişir? `memory-user.c` programını sonlandırdığınızda ne olacak? Rakamlar bekentilerinize uyuyor mu? Bunu farklı miktarlarda bellek kullanımı için deneyiniz. Gerçekten büyük miktarda bellek kullandığınızda ne olur?

/\* Bellek kullanımı mantıklı bir şekilde girilen `memory-user` komutlarında beklenilen gibi girilen MB miktarı kadar makinenin kullanılabilir ram'inden düştü. Programı sonlandırdığımızda ise eski haline geri dönüyor. Büyük miktarda bellek kullandığında bellek kullanımı düşmedi veya ben erkenden kapattığım için herhangi bir çökme ile karşılaşmadım. Hata da verebilirdi.

The image shows two terminal windows side-by-side. The left window displays the execution of the `memory-user` program with increasing memory sizes (300, 600, 900, 9000, 100000000) and a duration of 300 seconds. The right window shows the corresponding `free -h` command outputs, which show the total, used, free, shared, buffer/cache, and available memory statistics across Mem and Swap sections. The memory usage grows significantly with the program's execution, demonstrating a memory leak.

	berkehan@berkehan-vm: ~	berkehan@berkehan-vm: ~
berkehan@berkehan-vm: ~/Desktop/c programlar \$ ./memory-user 300	total used free shared buff/cache available Mem: 7,7G 1,6G 4,6G 42M 2,1G 6,4G Swap: 2,0G 0B 2,0G	
berkehan@berkehan-vm: ~/Desktop/c programlar \$ ./memory-user 600	total used free shared buff/cache available Mem: 7,7G 1,3G 4,3G 42M 2,1G 6,1G Swap: 2,0G 0B 2,0G	
berkehan@berkehan-vm: ~/Desktop/c programlar \$ ./memory-user 900	total used free shared buff/cache available Mem: 7,7G 1,9G 3,7G 42M 2,1G 5,5G Swap: 2,0G 0B 2,0G	
berkehan@berkehan-vm: ~/Desktop/c programlar \$ ./memory-user 9000	total used free shared buff/cache available Mem: 7,7G 1,8G 3,8G 42M 2,1G 5,6G Swap: 2,0G 0B 2,0G	
berkehan@berkehan-vm: ~/Desktop/c programlar \$ ./memory-user 100000000	total used free shared buff/cache available Mem: 7,7G 1,3G 4,3G 43M 2,1G 6,2G Swap: 2,0G 0B 2,0G	
berkehan@berkehan-vm: ~/Desktop/c programlar \$	total used free shared buff/cache available Mem: 7,7G 1,3G 4,3G 43M 2,1G 6,2G Swap: 2,0G 0B 2,0G	

\*/

5. pmap olarak bilinen bir aracı daha deneyelim. Biraz zaman ayırin ve pmap kılavuz sayfasını ayrıntılı olarak okuyun.

```
/* pmap komutu belirli bir sürecin bellek haritasını göstermek için kullanılır. Detaylı bellek kullanımlarını gösterir ve bellek sizıntılarını görebilmemizi sağlar.
```

```
*/
```

6. `pmap`'i kullanmak için ilgilendiğiniz sürecin **process ID'sini** bilmeniz gereklidir. Bu nedenle, önce tüm süreçlerin bir listesini görmek için `ps auxw`'yu çalıştırın; daha sonra tarayıcı gibi ilginç bir tane seçin. Bu durumda ayrıca `memory-user.c` programınızı da kullanabilirsiniz (aslında, size kolaylık sağlamak için bu programın `getpid()` çağrısını yapmasını ve PID'sini yazdırmasını bile sağlayabilirsiniz).

/\*Programa PID'ini yazdırın bir satır ekledim. Aslında ps auxw komutuyla da bulunabilirildi.

```
berkehan@berkehan-vm: ~/Desktop/c programlari$ ./memory-user 1000
PID: 4218

berkehan@berkehan-vm: ~
```

User	Process ID	CPU Usage	Memory Usage	Memory Type	Memory Address	Memory Size	Memory State	Memory Type	Memory Address	Memory Size	Memory State
gvfsd-network	3167	0.0	0.1	319568	9472	?	Sl	05:31	0:00	/usr/libexec/gvfsd-dnssd	--spawner :1.6 /org/gtk/gvfs/exec_spaw/
root	3214	0.0	0.0	0	0	?	I	05:32	0:00	[kworker/0:2]	--spawner :1.6 /org/gtk/gvfs/exec_spaw/3
root	3758	0.0	0.0	0	0	?	I	05:44	0:00	[kworker/1:1-events]	
root	3855	0.0	0.0	0	0	?	I	05:50	0:00	[kworker/u256:0-ext4-rsv-conversion]	
root	3898	0.0	0.0	0	0	?	I	05:55	0:00	[kworker/1:2-cgroup_destroy]	
root	3936	0.0	0.0	0	0	?	I	05:57	0:00	[kworker/u256:1-flush-8:0]	
root	4061	0.0	0.0	0	0	?	I	06:06	0:00	[kworker/u256:3-events_unbound]	
root	4062	0.0	0.0	0	0	?	I	06:06	0:00	[kworker/u256:4-flush-8:0]	
berkehan	4181	1.5	0.6	566576	54720	?	Ssl	06:06	0:00	/usr/libexec/gnome-terminal-server	
berkehan	4199	0.0	0.0	14024	5632	pts/0	Ss	06:06	0:00	bash	
berkehan	4218	60.2	12.6	1026776	1025536	pts/0	R+	06:07	0:10	./memory-user 1000	
berkehan	4228	0.0	0.0	14020	5632	pts/1	Ss	06:07	0:00	bash	
berkehan	4238	0.0	0.0	15416	3584	pts/1	R+	06:07	0:00	ps auwx	

```
berkehan@berkehan-vm:~$ pmap 4218
4218: ./memory-user 1000
00005581c2fb3000      4K r---- memory-user
00005581c2fb4000      4K r-x-- memory-user
00005581c2fb5000      4K r---- memory-user
00005581c2fb6000      4K r---- memory-user
00005581c2fb7000      4K rw--- memory-user
00005581c3084000    132K rw--- [ anon ]
00007f52a3fff000 1024004K rw--- [ anon ]
00007f52e2800000     160K r---- libc.so.6
00007f52e2828000   1620K r-x-- libc.so.6
00007f52e29bd000    352K r---- libc.so.6
00007f52e2a15000     16K r---- libc.so.6
00007f52e2a19000     8K rw--- libc.so.6
00007f52e2a1b000    52K rw--- [ anon ]
00007f52e2ba5000    12K rw--- [ anon ]
00007f52e2bb7000     8K rw--- [ anon ]
00007f52e2bb9000     8K r---- ld-linux-x86-64.so.2
00007f52e2bbb000   168K r-x-- ld-linux-x86-64.so.2
00007f52e2be5000    44K r---- ld-linux-x86-64.so.2
00007f52e2bf1000     8K r---- ld-linux-x86-64.so.2
00007f52e2bf3000     8K rw--- ld-linux-x86-64.so.2
00007ffdc153b000   132K rw--- [ stack ]
00007ffdc15ad000    16K r---- [ anon ]
00007ffdc15b1000     8K r-x-- [ anon ]
ffffffffffff600000     4K --x-- [ anon ]
total          1026780K
```

\*/

7. Şimdi süreçle ilgili birçok ayrıntıyı ortaya çıkarmak için çeşitli işaretler (-X gibi) kullanarak bu süreçlerin bazlarında pmap'i çalıştırın. Ne görüporsunuz? Basit code/stack/heap anlayışımızın aksine, modern bir adres alanını kaç farklı varlık oluşturur?

/\* Programımızın bellek kullanımı görünüyor. Yazdığımız kodu görüyoruz. (mb miktarı). Modern adres alanı sadece 3 kısımdan oluşuyor. 1. Programın kendi kodu 2. Heap Alanı 3. Stack Alanı 4. Paylaşılan Kütüphaneler 5. Özel sistem Bölümleri. Toplam 5 adet farklı varlık oluşturur.

```
berkehan@berkehan-vm:~$ pmap -x 4218
4218: ./memory-user 1000
Address          Kbytes   RSS  Dirty Mode Mapping
00005581c2fb3000        4      4      0 r---- memory-user
00005581c2fb4000        4      4      0 r-x-- memory-user
00005581c2fb5000        4      4      0 r---- memory-user
00005581c2fb6000        4      4      4 r---- memory-user
00005581c2fb7000        4      4      4 rw--- memory-user
00005581c3084000     132      4      4 rw--- [ anon ]
00007f52a3ffff000 1024004 1024004 1024004 rw--- [ anon ]
00007f52e2800000       160    160      0 r---- libc.so.6
00007f52e2828000     1620   1036      0 r-x-- libc.so.6
00007f52e29bd000      352    124      0 r---- libc.so.6
00007f52e2a15000       16     16     16 r---- libc.so.6
00007f52e2a19000       8      8      8 rw--- libc.so.6
00007f52e2a1b000      52     20     20 rw--- [ anon ]
00007f52e2ba5000      12     8      8 rw--- [ anon ]
00007f52e2bb7000       8      4      4 rw--- [ anon ]
00007f52e2bb9000       8      8      0 r---- ld-linux-x86-64.so.2
00007f52e2bbb000     168    168      0 r-x-- ld-linux-x86-64.so.2
00007f52e2be5000      44    40      0 r---- ld-linux-x86-64.so.2
00007f52e2bf1000       8      8      8 r---- ld-linux-x86-64.so.2
00007f52e2bf3000       8      8      8 rw--- ld-linux-x86-64.so.2
00007fffdc153b000     132    12     12 rw--- [ stack ]
00007fffdc15ad000      16     0      0 r---- [ anon ]
00007fffdc15b1000       8      4      0 r-x-- [ anon ]
ffffffffff600000       4      0      0 --x-- [ anon ]
-----
total kB      1026780 1025652 1024100
```

\*/

8. Son olarak, pmap'i memory-user.c programınızda farklı miktarlarda kullanılan bellekle çalıştıralım.  
Burada ne görüyorunuz? pmap çıktısı beklenilerinize uyuyor mu?

/\*10, 100 ve 1000 mb komutlarını kullandım. Memory-user programı bellek miktarlarını başarıyla tahsis etmiş.

```
berkehan@berkehan-vm:~$ pmap -x 10
berkehan@berkehan-vm:~$ ./memory-user 10
PID: 5075
berkehan@berkehan-vm:~$ pmap -x 100
berkehan@berkehan-vm:~$ ./memory-user 100
PID: 5078
berkehan@berkehan-vm:~$ pmap -x 1000
berkehan@berkehan-vm:~$ ./memory-user 1000
PID: 5078
berkehan@berkehan-vm:~$ pmap -x 5075
berkehan@berkehan-vm:~$ ./memory-user 10
5075: ./memory-user 10
Address          Kbytes   RSS  Dirty Mode Mapping
00005581c2fb3000        4      4      0 r---- memory-user
00005581c2fb4000        4      4      0 r-x-- memory-user
00005581c2fb5000        4      4      0 r---- memory-user
00005581c2fb6000        4      4      4 r---- memory-user
00005581c2fb7000        4      4      4 rw--- memory-user
00005581c3084000     132      4      4 rw--- [ anon ]
00007f52a3ffff000 1024004 1024004 1024004 rw--- [ anon ]
00007f52e2800000       160    160      0 r---- libc.so.6
00007f52e2828000     1620   1036      0 r-x-- libc.so.6
00007f52e29bd000      352    124      0 r---- libc.so.6
00007f52e2a15000       16     16     16 r---- libc.so.6
00007f52e2a19000       8      8      8 rw--- libc.so.6
00007f52e2a1b000      52     20     20 rw--- [ anon ]
00007f52e2ba5000      12     8      8 rw--- [ anon ]
00007f52e2bb7000       8      4      4 rw--- [ anon ]
00007f52e2bb9000       8      8      0 r---- ld-linux-x86-64.so.2
00007f52e2bbb000     168    168      0 r-x-- ld-linux-x86-64.so.2
00007f52e2be5000      44    40      0 r---- ld-linux-x86-64.so.2
00007f52e2bf1000       8      8      8 r---- ld-linux-x86-64.so.2
00007f52e2bf3000       8      8      8 rw--- ld-linux-x86-64.so.2
00007fffdc153b000     132    12     12 rw--- [ stack ]
00007fffdc15ad000      16     0      0 r---- [ anon ]
00007fffdc15b1000       8      4      0 r-x-- [ anon ]
ffffffffff600000       4      0      0 --x-- [ anon ]
-----
total kB      13020  11896  10344
berkehan@berkehan-vm:~$
```

```
berkehan@berkehan-vm:~$ pmap -x 100
berkehan@berkehan-vm:~$ ./memory-user 100
PID: 5078
berkehan@berkehan-vm:~$ pmap -x 1000
berkehan@berkehan-vm:~$ ./memory-user 1000
PID: 5078
berkehan@berkehan-vm:~$ pmap -x 5078
berkehan@berkehan-vm:~$ ./memory-user 100
5078: ./memory-user 100
Address          Kbytes   RSS  Dirty Mode Mapping
00005581c2fb3000        4      4      0 r---- memory-user
00005581c2fb4000        4      4      0 r-x-- memory-user
00005581c2fb5000        4      4      0 r---- memory-user
00005581c2fb6000        4      4      4 r---- memory-user
00005581c2fb7000        4      4      4 rw--- memory-user
00005581c3084000     132      4      4 rw--- [ anon ]
00007f52a3ffff000 102404 102404 102404 rw--- [ anon ]
00007fd9d86600000       160    168      0 r---- libc.so.6
00007fd9d86628000     1620   1036      0 r-x-- libc.so.6
00007fd9d86640000      352    124      0 r---- libc.so.6
00007fd9d86658000      16     16     16 r---- libc.so.6
00007fd9d866819800      8      8      8 rw--- libc.so.6
00007fd9d866819800      8      8      0 r---- ld-linux-x86-64.so.2
00007fd9d86688000      52     20     20 rw--- [ anon ]
00007fd9d86688000      12     8      8 rw--- [ anon ]
00007fd9d86688000      12     8      4 rw--- [ anon ]
00007fd9d86688000      8      8      0 r---- ld-linux-x86-64.so.2
00007fd9d86688000      8      8      0 r---- ld-linux-x86-64.so.2
00007fd9d86688000      168    168      0 r-x-- ld-linux-x86-64.so.2
00007fd9d86694000      44    40      0 r---- ld-linux-x86-64.so.2
00007fd9d86694000      8      8      0 r---- ld-linux-x86-64.so.2
00007fd9d866a2000      8      8      8 rw--- ld-linux-x86-64.so.2
00007ffcc37870800      132    12     12 rw--- [ stack ]
00007ffcc37df0800      16     0      0 r---- [ anon ]
00007ffcc37df0800      8      4      0 r-x-- [ anon ]
00007ffcc37df0800      4      0      0 --x-- [ anon ]
-----
total kB      105180 104052 102580
berkehan@berkehan-vm:~$
```

```
berkehan@berkehan-vm:~/Desktop/c programlari$ ./memory-user 1000
PID: 5081
berkehan@berkehan-vm:~$ pmap -x 5081
5081: ./memory-user 1000
Address Kbytes RSS Dirty Mode Mapping
00005643580e4000 4 4 0 r---- memory-user
00005643580e5000 4 4 0 r-x-- memory-user
00005643580e6000 4 4 0 r---- memory-user
00005643580e7000 4 4 4 r---- memory-user
00005643580e8000 4 4 4 rw--- memory-user
00005643582ad000 132 4 4 rw--- [ anon ]
00007fed521ff000 1024004 1024004 1024004 rw--- [ anon ]
00007fed90a00000 160 160 0 r---- libc.so.6
00007fed90a29000 1620 1036 0 r-x-- libc.so.6
00007fed90b0bd000 352 124 0 r---- libc.so.6
00007fed90c15000 16 16 16 r---- libc.so.6
00007fed90c19000 8 8 8 rw--- libc.so.6
00007fed90c1b000 52 20 20 rw--- [ anon ]
00007fed90d57000 12 8 8 rw--- [ anon ]
00007fed90d69000 8 4 4 rw--- [ anon ]
00007fed90d6b000 8 8 0 r---- ld-linux-x86-64.so.2
00007fed90d6f000 168 168 0 r-x-- ld-linux-x86-64.so.2
00007fed90d97000 44 40 0 r---- ld-linux-x86-64.so.2
00007fed90da3000 8 8 8 r---- ld-linux-x86-64.so.2
00007fed90da5000 8 8 8 rw--- ld-linux-x86-64.so.2
00007ffffd8f761000 132 16 16 rw--- [ stack ]
00007ffffd8f79e000 16 0 0 r---- [ anon ]
00007ffffd8f7a2000 8 4 0 r-x-- [ anon ]
ffffffffff600000 4 0 0 --x-- [ anon ]
total kB 1026780 1025656 1024104
```

\*/

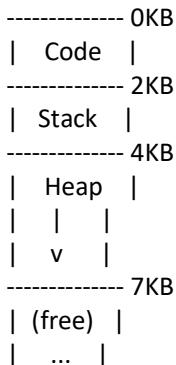
## Adres Çevrimi (Address Translation)

Bu bölümde kullanılan açıklamalar kaynak dosya olarak eklenmiştir. Ödev ekinde bulunan [15\\_BaseAndBound\\_new.pptx](#) dosyasını inceleyebilirsiniz. Ayrıca ödevlerde kaynak kitap olarak kullanılan **Operating Systems: Three Easy Pieces** kitabının ilgili bölümünü (Chapter 15) [buradan](#) okuyabilirsiniz.

## Ödev (Simülasyon)

[relocation.py](#)<sup>1</sup> programı, taban ve sınır kayıtları olan bir sistemde adres çevirilerinin nasıl yapıldığını görmeyi sağlar. Daha önce olduğu gibi, taban ve sınırlar konusundaki anlayışınızı test etmek amacıyla programı çalıştırmanın iki adımı vardır. İlk olarak, bir dizi çeviri oluşturmak için `-c` bayrağı olmadan çalıştırın ve adres çevirilerini kendiniz doğru şekilde gerçekleştiretip gerçekleştiremeyeceğinize bakın. Daha sonra işiniz bittiğinde cevaplarınızı kontrol etmek için `-c` bayrağıyla çalıştırın.

Bu ödevde, kanonik olandan biraz farklı, alanın karşıt uçlarında heap ve stack bulunan bir adres alanı varsayıcağız. Bunun yerine, adres alanının bir kod bölümüne, ardından sabit boyutlu (küçük) bir stack alanına ve hemen ardından aşağıya doğru büyuyen, aşağıdaki şekilde gördüğünüz gibi görünen bir heap alanına sahip olduğunu varsayıcağız. Bu konfigürasyonda, adres alanının daha yüksek bölgelerine doğru tek bir büyümeye yönü vardır.



Şekilde sınır kaydı, adres alanının sonunu temsil ettiği için 7~KB'ye ayarlanacaktır. Sınırlar dahilindeki herhangi bir adrese yapılan atıflar yasal (legal) kabul edilecektir; bu değerin üzerindeki referanslar sınırların dışındadır ve bu nedenle donanım bir istisna (exception) oluşturacaktır.

Varsayılan bayraklarla çalıştmak için komut satırına `relocation.py` yazın. Sonuç şöyle bir şey olmalı:

```
prompt> ./relocation.py
...
Base-and-Bounds register information:

  Base    : 0x00003082 (decimal 12418)
  Limit   : 472

  Virtual Address Trace
    VA 0: 0x01ae (decimal:430) -> PA or violation?
    VA 1: 0x0109 (decimal:265) -> PA or violation?
    VA 2: 0x020b (decimal:523) -> PA or violation?
    VA 3: 0x019e (decimal:414) -> PA or violation?
    VA 4: 0x0322 (decimal:802) -> PA or violation?
```

<sup>1</sup> <https://github.com/salihayesiyurt/fall-2023-bil3003-os-hw.git> reposunda yer alan hw7/vm-mechanism dizini içerisindeki [relocation.py](#) dosyasını kullanınız. Bir Python dosyasının nasıl çalıştırılacağına dair detaylı bilgiler için Teams grubundaki videolara bakınız.

Her sanal adres için, ya çevirdiği fiziksel adresi yazın VEYA bunun sınır dışı bir adres olduğunu (segmentasyon ihlali-segmentation violation) yazınız. Bu sorun için, belirli bir boyutta basit bir sanal adres alanı varsayılmışınız.

Gördüğünüz gibi ödev basitçe rastgele sanal adresler üretiyor. Her biri için sınırlar dahilinde olup olmadığını belirlemeli ve eğer öyleyse hangi fiziksel adrese çevrildiğini belirlemelisiniz. `-c` ("bunu benim için hesapla" bayrağı) ile çalıştırırmak bize bu çevirilerin sonuçlarını, yani bunların geçerli olup olmadığını ve geçerliyse ortaya çıkan fiziksel adresleri verir. Kolaylık sağlamak için, tüm sayılar hem onaltılık hem de ondalık olarak verilmiştir.

```
prompt> ./relocation.py -c
...
Virtual Address Trace
VA 0: 0x01ae (decimal:430) -> VALID: 0x00003230 (dec:12848)
VA 1: 0x0109 (decimal:265) -> VALID: 0x0000318b (dec:12683)
VA 2: 0x020b (decimal:523) -> SEGMENTATION VIOLATION
VA 3: 0x019e (decimal:414) -> VALID: 0x00003220 (dec:12832)
VA 4: 0x0322 (decimal:802) -> SEGMENTATION VIOLATION
```

Taban adresi 12418 (ondalık) olan adres 430, sınırlar dahilindedir (yani, 472 sınır kaydından küçüktür) ve dolayısıyla 12418 veya 12848'e eklenen 430'a çevrilir. Yukarıda gösterilen adreslerden birkaçı sınırların dışındadır. (523, 802), sınırları aşıkları için. Oldukça basit, değil mi? Aslında bu, taban ve sınırların (base and bounds) güzelliklerinden biridir: o kadar basit ki! 😊

Neler olup bittiğini daha iyi kontrol etmek için kullanabileceğiniz birkaç işaret vardır:

```
prompt> ./relocation.py -h
Usage: relocation.py [options]

Options:
-h, --help      show this help message and exit
-s SEED, --seed=SEED the random seed
-a ASIZE, --asize=ASIZE address space size (e.g., 16, 64k, 32m)
-p PSIZE, --physmem=PSIZE physical memory size (e.g., 16, 64k)
-n NUM, --addresses=NUM # of virtual addresses to generate
-b BASE, --b=BASE value of base register
-l LIMIT, --l=LIMIT value of limit register
-c, --compute   compute answers for me
```

Özellikle sanal adres alanı boyutunu (`-a`), fiziksel belleğin boyutunu (`-p`), oluşturulacak sanal adres sayısını (`-n`) ve bu süreç için taban ve sınır kayıtlarının değerlerini (sırasıyla `-b` ve `-l`) kontrol edebilirisiniz.

- 1, 2 ve 3 numaralı seed'ler ile çalıştırın ve süreç tarafından oluşturulan her sanal adresin sınırların içinde mi yoksa dışında mı olduğunu hesaplayın. Sınırlar dahilindeyse çeviriyi (translation) hesaplayınız.

*/\* Süper eğlenceli bir kısım. Limitlerin dışında olanlar segmentasyon ihlali eğer limitlerin içindedeyse base değere sanal adresini ekliyoruz ve fiziksel adresi buluyoruz. İşlemleri screenshotlar üzerinde yaptım.*

```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw7/vm-mechanism$ python3 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: 0x0000020e (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: 0x0000020b (decimal: 667) --> PA or segmentation violation?

Segmentation violation

14145

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.

```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw7/vm-mechanism$ python3 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: 0x000000357 (decimal: 95) --> PA or segmentation violation?  
 VA 3: 0x000002f1 (decimal: 753) --> PA or segmentation violation?  
 VA 4: 0x0000003d (decimal: 65) --> PA or segmentation violation?

15586  
 15615

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.

```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw7/vm-mechanism$ python3 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: 0x0000017e (decimal: 378) --> PA or segmentation violation?  
 VA 1: 0x0000026a (decimal: 618) --> PA or segmentation violation?  
 VA 2: 0x00000200 (decimal: 640) --> PA or segmentation violation?  
 VA 3: 0x00000043 (decimal: 67) --> PA or segmentation violation?  
 VA 4: 0x0000000d (decimal: 13) --> PA or segmentation violation?

8983  
 8929

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.

\*/

2. Şu bayraklarla çalıştırın: -s 0 -n 10. Oluşturulan tüm sanal adreslerin sınırlar dahilinde olduğundan emin olmak için -l'yi (sınır kaydı) hangi değere ayarladınız?

/\*930'a ayarladım.

```
berkehan@berkehan-vn:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw7/vm-mechanism$ python3 relocation.py -s 0 -n 10 -l 930

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

Base-and-Bounds register information:

Base : 0x00000360b (decimal 13835)
Limit : 930

Virtual Address Trace
VA 0: 0x00000308 (decimal: 776) --> PA or segmentation violation?
VA 1: 0x000001ae (decimal: 430) --> PA or segmentation violation?
VA 2: 0x00000109 (decimal: 265) --> PA or segmentation violation?
VA 3: 0x0000020b (decimal: 523) --> PA or segmentation violation?
VA 4: 0x0000019e (decimal: 414) --> PA or segmentation violation?
VA 5: 0x00000322 (decimal: 802) --> PA or segmentation violation?
VA 6: 0x00000136 (decimal: 310) --> PA or segmentation violation?
VA 7: 0x000001e8 (decimal: 488) --> PA or segmentation violation?
VA 8: 0x00000255 (decimal: 597) --> PA or segmentation violation?
VA 9: 0x000003a1 (decimal: 929) --> 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.
```

\*/

3. Şu bayraklarla çalıştırın: -s 1 -n 10 -l 100. Adres alanının hala bütünüyle fiziksel belleğe sigmasını sağlayacak şekilde tabanın ayarlanabileceği maksimum değer nedir?

/\*Fiziksel Bellek Boyutu – Sanal Adres Alanı Boyutu (Limit) = 16,384 byte – 100 byte= 16,284 byte.

```
berkehan@berkehan-vn:~/Desktop/hw-os/fall-2023-bil3003-os-hw/hw7/vm-mechanism$ python3 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?

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.
```

\*/

4. Yukarıdaki sorunlardan bazılarını daha büyük adres alanları (-a) ve fiziksel bellekler (-p) ile çalıştırınız.

/\*Limit 100 byte olarak ayarlandığı için hepsi segmentation violation olmuştur.

```
berkehan@berkehan-vm:~/Desktop/hw-os/fall-2023-bil3003-os-hw7/vm-mechanisms$ python3 relocation.py -s 1 -n 10 -l 100 -a 22k -p 66k -c
ARG seed 1
ARG address space size 22k
ARG phys mem size 66k

Base-and-Bounds register information:

Base   : 0x000002378 (decimal 9080)
Limit  : 100

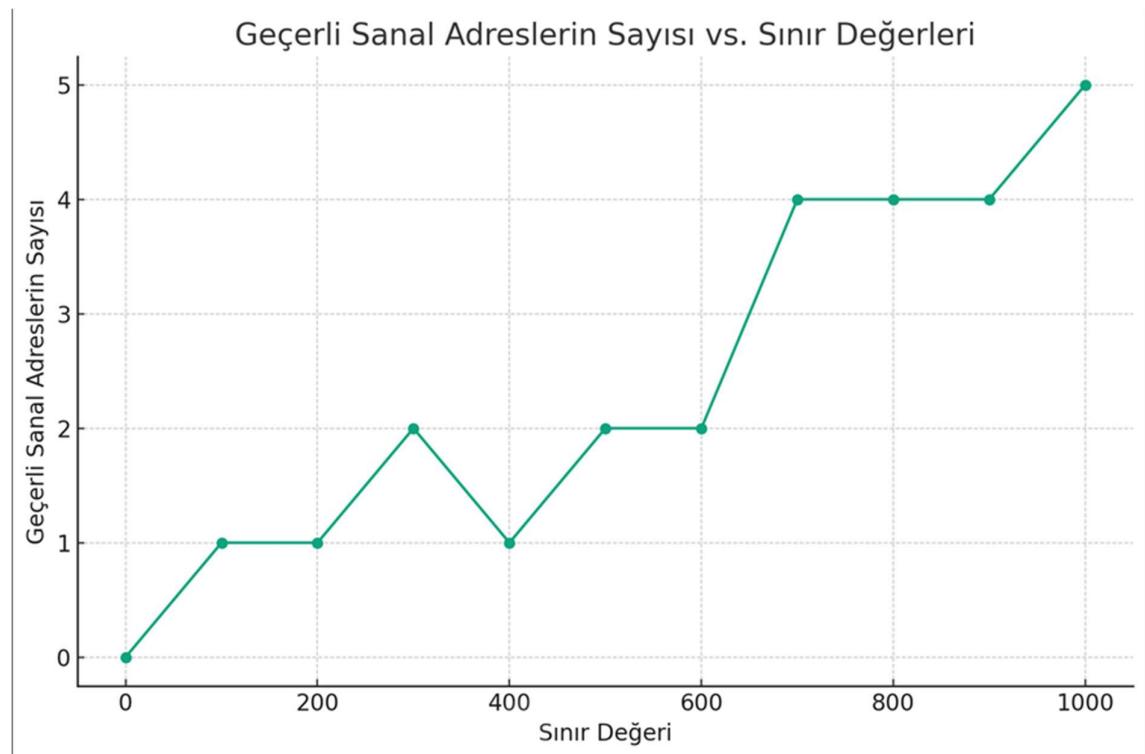
Virtual Address Trace
VA 0: 0x00004a92 (decimal: 19090) --> SEGMENTATION VIOLATION
VA 1: 0x00004336 (decimal: 17206) --> SEGMENTATION VIOLATION
VA 2: 0x00001672 (decimal: 5746) --> SEGMENTATION VIOLATION
VA 3: 0x00002b99 (decimal: 11161) --> SEGMENTATION VIOLATION
VA 4: 0x0000278e (decimal: 10126) --> SEGMENTATION VIOLATION
VA 5: 0x00003957 (decimal: 14679) --> SEGMENTATION VIOLATION
VA 6: 0x00004568 (decimal: 17768) --> SEGMENTATION VIOLATION
VA 7: 0x00000842 (decimal: 2114) --> SEGMENTATION VIOLATION
VA 8: 0x0000027e (decimal: 638) --> SEGMENTATION VIOLATION
VA 9: 0x0000498c (decimal: 18828) --> SEGMENTATION VIOLATION
```

\*/

5. Sınır kaydının değerinin bir fonksiyonu olarak rastgele oluşturulmuş sanal adreslerin ne kadarı geçerlidir? Sınır değerleri 0'dan adres alanının maksimum boyutuna kadar değişen farklı rastgele seed'ler ile çalıştırılarak bir grafik oluşturunuz.

*/\* Bir Python scripti yazarak işimi kolaylaştırdım. Kodun screenshotu aşağıdadır. Sınır değeri arttıkça geçerli sanal adreslerin sayısı da genellikle artmaktadır.*

```
1 import os
2 import random
3
4 max_address_space_size = 1024
5 limit_step = 100
6
7 for limit in range(0, max_address_space_size + 1, limit_step):
8     seed = random.randint(0, 1000)
9     command = f"python3 relocation.py -s {seed} -l {limit} -c"
10    os.system(command)
11 |
```



\* /