15.2

Wenn man folgendes in das Terminal eingibt *„./relocation.py -s 0 -n 10 -c“,* dann benutzt man mit *„-s 0“* den Seed 0 und generiert mit *„-n 10“*, 10 verschiedene virtuelle Adressen.

Die höchste dieser virtuellen Adressen liegt bei der Adresse 929. Daher übergeben wir dem Programm die Flag -l, die den Bound Wert festlegt. Diesen initialisieren wir mit 930, dass die 929 reinpasst (da der Bound Wert z.B. 930 nicht mehr mitzählt).

Dies funktioniert lediglich für Seed 0 oder Seeds mit Werten geringer als 930. Bei Seed 3 hingegen gibt es bereits eine erneute Segment Violation, da eine erzeugte Adresse mit 1019 über 929 liegt.

15.3

Der physische Arbeitsspeicher ist in der Simulation 16 kB groß, was bei der Ausgabe des Programmes ersichtlich ist (ARG phys mem size 16k).

Um nun 16 kB zu errechnen, müssen wir folgende Rechnung machen:
16 * 1024 = 16384

Allerdings müssen wir noch den Bound von 100 abziehen, dass auch Platz da ist für die virtuelle Adresse. Damit kommen wir auf folgenden maximalen Base Wert um noch in den physischen Arbeitsspeicher zu passen.

16384 – 100 = 16284

Der höchstmögliche Base Wert liegt somit bei **16284.**

15.4

Wir haben den physischen Arbeitsspeicher auf 2 GB erhöht und den virtuellen Arbeitsspeicher auf 32 MB. Zudem haben wir die Base 2147483548 erhöht, was wieder 100 weniger ist, als vom maximalen Base Wert.

Um hier erneut den maximalen Base Wert zu berechnen von 2GB machen wir folgendes:
2 * 1024 * 1024 * 1024 = 2147483648

Nun muss nur noch die 100 abgezogen werden, dass Platz für die virtuelle Adresse vorhanden ist.
2147483648 – 100 = 2147483548

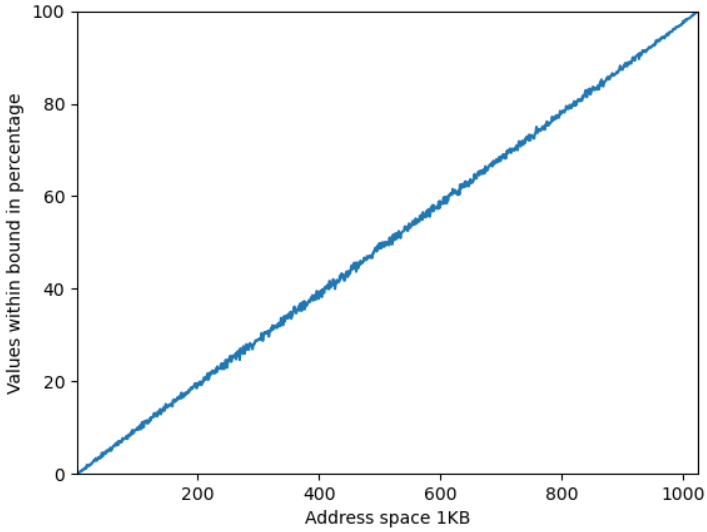
Der höchstmögliche Base Wert für 2GB physischen Arbeitsspeicher liegt bei **2147483548.**

Was hat sich verändert?

Die Wahrscheinlichkeit, dass eine zufällige virtuelle Adresse innerhalb des Bounds ist verringert sich linear. Die Wahrscheinlichkeit, dass eine zufällige virtuelle Adresse innerhalb liegt, ist weitaus geringer als in der Aufgabe 15.3 zuvor.

15.5

Wir haben einen physischen und virtuellen Adressraum von 1KB verwendet. Insgesamt haben wir mit 10.000 Seeds gearbeitet. Desto höher die virtuelle Adresse im Adressraum wird, desto höher ist die Wahrscheinlichkeit, dass er gültig ist. Bei genau einem KB ist die Wahrscheinlichkeit logischerweise genau 100%. Daher steigt die Wahrscheinlichkeit linear, dass die zufällige virtuelle Adresse innerhalb des Bound Registers liegt.




```python
import random
import matplotlib.pyplot as plt
import numpy as np

address_space = 1024
address_scale = [0] * address_space
seed_amount = 10000
counter = True

for a in range(seed_amount):
    for b in range(address_space):
        vaddr = int(address_space * random.random())
        if (vaddr < b):
            address_scale[b] = address_scale[b] + 1

for c in range(address_space):
    address_scale[c] = address_scale[c] / 100
    if (address_scale[c] == 1 and counter == True):
        print(c)
        counter = False

print(address_scale[1023])

fig = plt.figure()
x = np.linspace(1, address_space, address_space)
plt.plot(x, address_scale)
plt.margins(0)
plt.xlabel("Address space 1KB")
plt.ylim(0, 100)
plt.ylabel("Values within bound in percentage")
plt.savefig("address_space.png")
plt.show()
```

Wir haben wie bereits erwähnt mit einem physischen und virtuellen Adressraum von 1KB gerechnet. Zudem mit 10.000 Seeds.

Im Anschluss haben wir 10.000-mal die Schleife „a“ durchlaufen lassen für jeden Seed und in der Schleife jeweils für jeden Adressraum ein weiterer Durchlauf starten lassen mit der Schleife „b“.

In der zweiten Schleife „b“ generieren wir eine zufällige Adresse und überprüfen ob diese innerhalb des Bounds liegt. Wenn sie innerhalb des Bounds ist und damit gültig ist, addieren wir 1 in einer Variablen für die Adresse hinzu.

In einer neuen Schleife außerhalb der anderen beiden, teilen wir nun das Ergebnis der 10.000 Werte durch 100 um den Schnitt in Prozent zu erhalten.

Im Anschluss wurde lediglich der Plot ausgegeben.