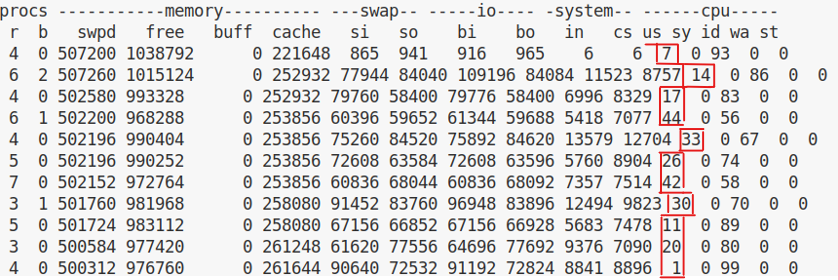
**Gruppe 15 | Tobias Schoch, Luis Nothvogel**

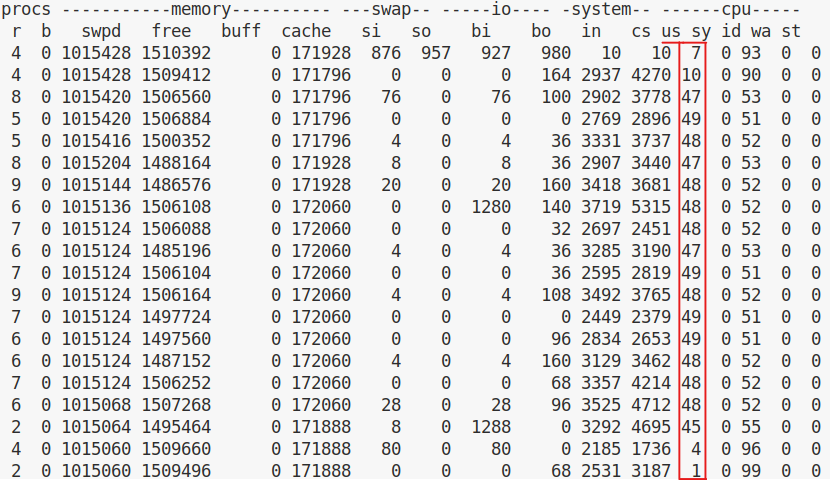
**Simulation wurde auf dem HTWG Container ausgeführt**

**21.1**

**Beim Container:**

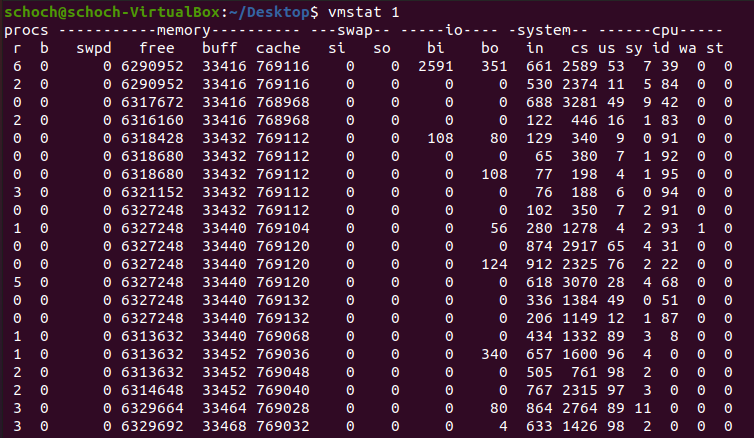
****

Wenn man mem.c mit 1 aufruft, kann man eigentlich kaum eine Veränderung der Werte feststellen. 1 MB ist aber auch nicht sonderlich viel, die hier benötigt werden da der Container bis zu 2 GB physikalischen Speicher hat und darauf nochmal 3 GB Swap Space kommen. Um das ganze laufen zu lassen muss nichts geswapped werden, es kann eigentlich immer komplett in die Speicher geladen werden. Man kann anhand der us-Statistik sehen ab wann das Programm gestartet wurde, da der Wert dort in die Höhe schießt, auch kann man daran dann ablesen wann es noch läuft und ab wann es gestoppt wurde. Die user time statistik gibt gemäß Manpage an wie viel Zeit damit verbraucht non-kernel code auszuführen. Ob die user time colum hier Sinn ergibt, ist schwierig zu sagen, da noch andere Prozesse zusätzlich laufen, die das ganze beeinflussen. Da es aber mehr wird, wenn man das Programm ausführt scheint es erstmal Sinn zu ergeben.



Wenn man mehr als eine Instanz laufen lässt, (hier 3) sieht man, dass sich im Vergleich zu vorher nicht wirklich viel ändert. Da wir ja immer noch jeweils 1 MB pro Instanz benötigen ist das immer noch kaum merklich.

**I7 4790K 1 CPU-Kern + 980TI + 8GB Ram für die VM:**

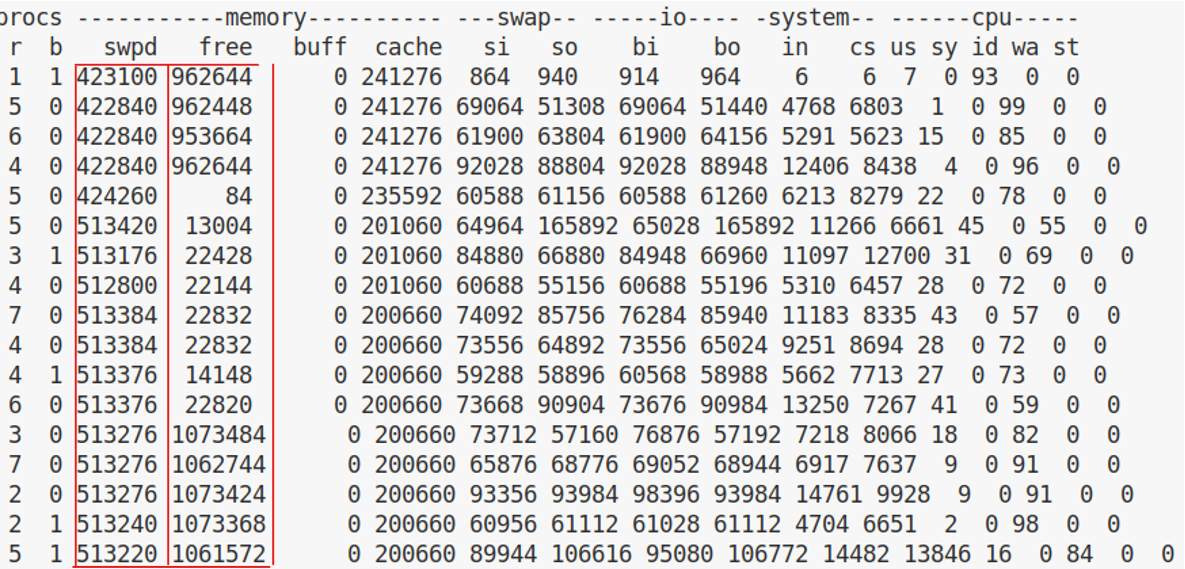


Auch hier ist kein sichtlicher Unterschied zu erkennen. 6,3 GB sind frei von 8GB was Sinn ergibt, da OS und andere Programme wie z.B. der offene Firefox auch Ram verbrauchen.

Die user-time Statistik ähnelt der Containerausführung.

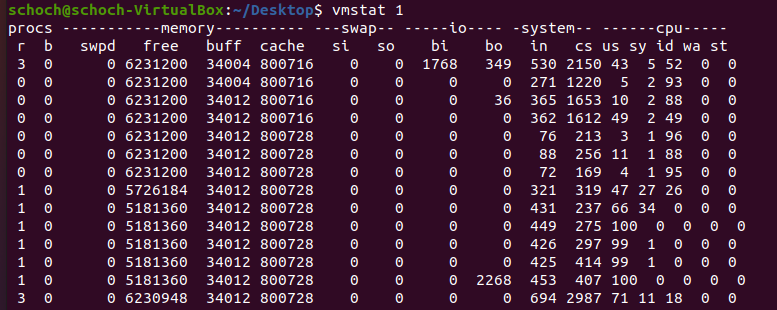
**21.2**

**Beim Container:**

****

Hier kann man recht leicht erkennen ab wann das Programm gestartet wurde. Nämlich sobald der free wert fällt. Denn dieses Mal führen wir das Programm ja mit 1,024 GB aus. Also die Hälfte unsere verfügbaren Rams, dies bedeutet natürlich, dass mehr Speicher allokiert werden muss. Man kann auch sehr leicht erkennen ab wann das Programm gestoppt wird, nämlich sobald der free wert wieder drastisch in die Höhe schießt. Das System hat danach unerwartet viel mehr Speicher frei als davor. Man müsste eigentlich davon ausgehen, dass das System wieder genau gleich viel wie vorher zu Verfügung hat.

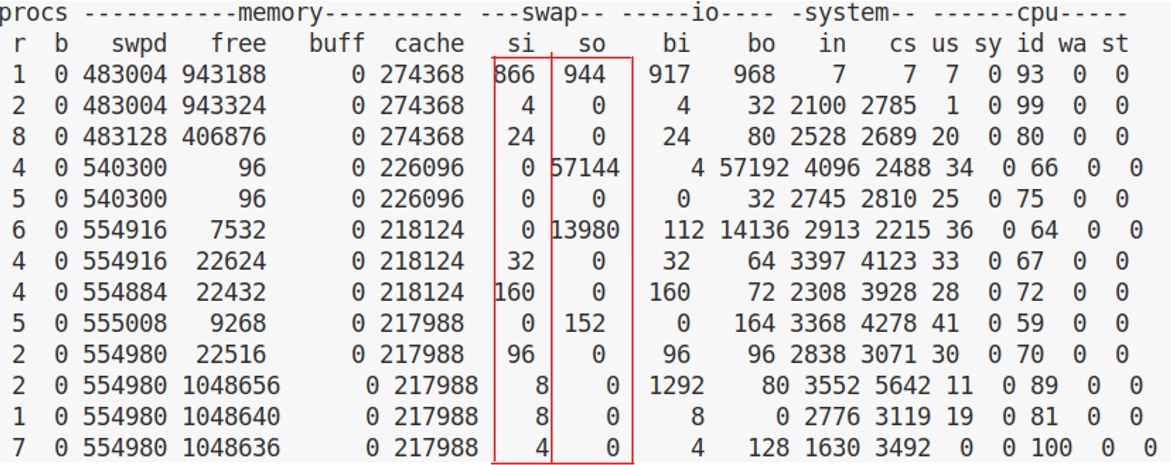
**I7 4790K 1 CPU-Kern + 980TI + 8GB Ram für die VM:**



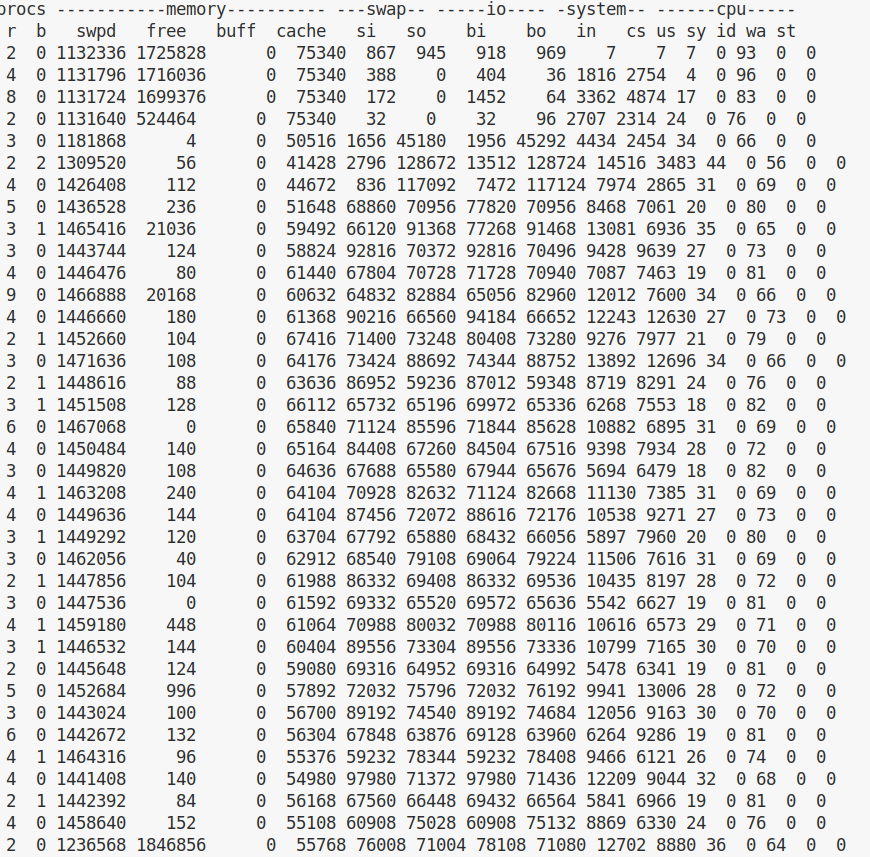
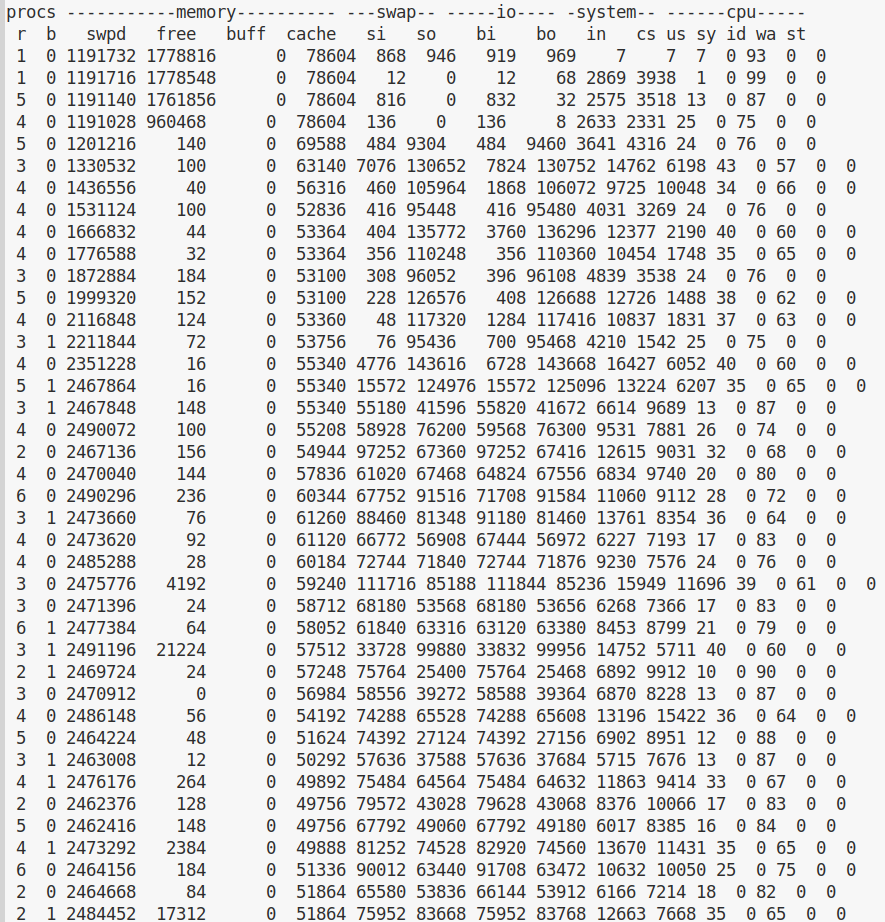
Auch hier erkennt man gut, wo der Prozess begonnen hat um Speicher zu reservieren. In der letzten Zeile wird der Prozess dann geschlossen und steigt ungefähr wieder um die zuvor reservierte Menge an, nachdem diese befreit wurde. Da 1024MB leicht zu reservieren sind in unserem 8GB System, muss nicht auf die Disk geswapped werden.

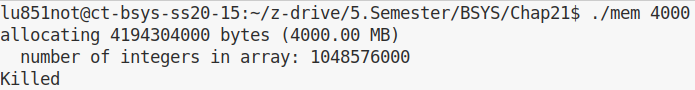
**21.3**

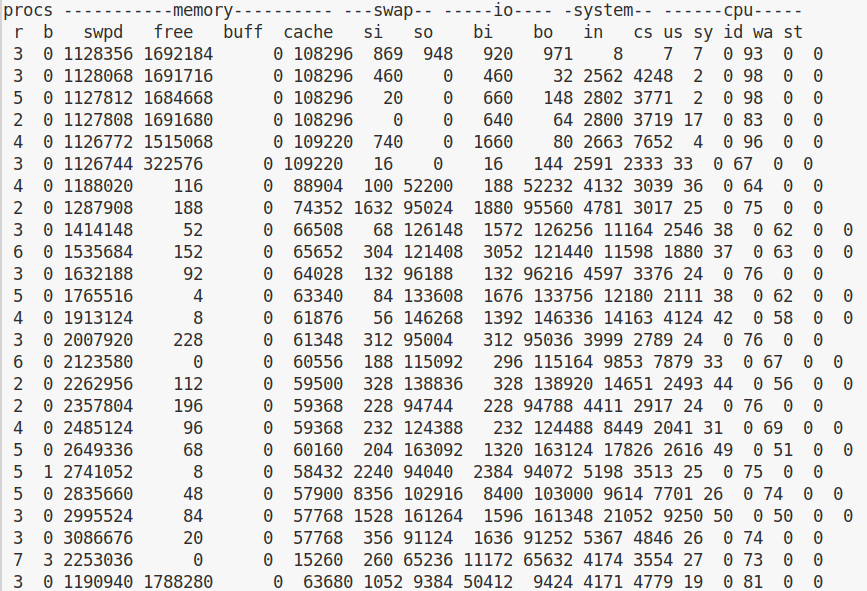
**Beim Container: 2 GB Ram**

****

Die Si spalte gibt an wie viel von der Disk reingeswapped wurde. Und die so spalte gibt an wie viel zur Disk geswapped wurde. Ja sie geben auch mal nicht 0 Werte zurück. Man kann dies sehr gut beobachten, wenn der free Wert bei 96 ist. Da musste sehr viel Speicher allokiert werden und anscheinend so viel, dass eine bestimmte Menge zur Disk ausgeswapped werden musste. Ähnliches gilt für den Si wert, denn es kann nun mal sein, dass ein Teil des Programms auf der Disk ausgelagert wurde und nur bei Bedarf reinswapped wird, daher die kleineren zahlen.

****

****

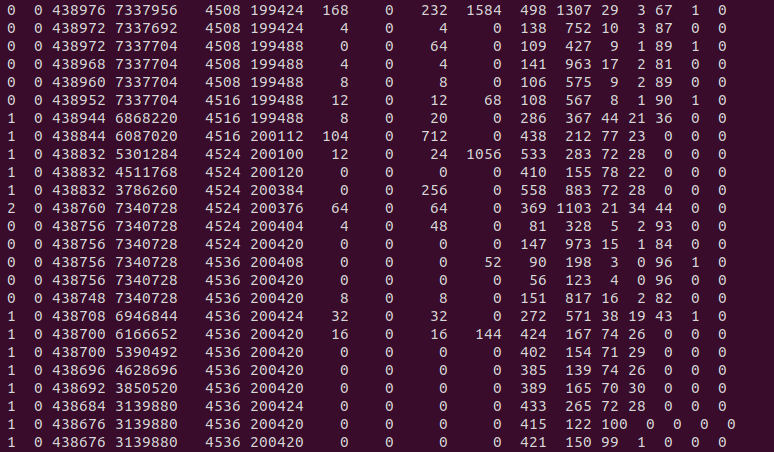
****

Bei größeren MB werten verhält es sich logischerweise etwas anders. Denn es muss im Fall mehr geswapped werden. Das spiegelt sich auch in den Si & So Werten wider. Die Werte ergeben auch Sinn. Denn es wird in diesem Fall nur die Arraystelle reingeswapped die gerade beschrieben wird. Daher ist der Si Wert so gering.

Da der freie Speicher sehr leer ist und zum Teil sogar auf 0 sinkt, muss auch dementsprechend viel Speicher auf die Disk geswapped werden. Daher ist auch dementsprechend der „so“ Wert hoch.

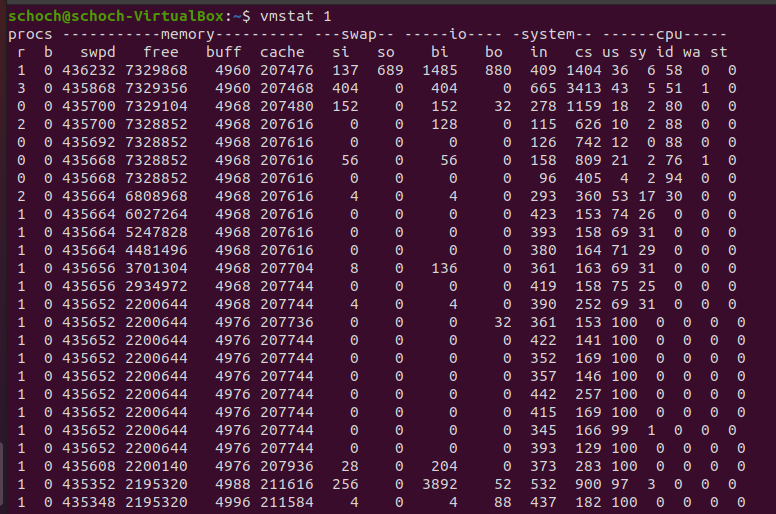
**I7 4790K 1 CPU-Kern + 980TI + 8GB Ram für die VM:**

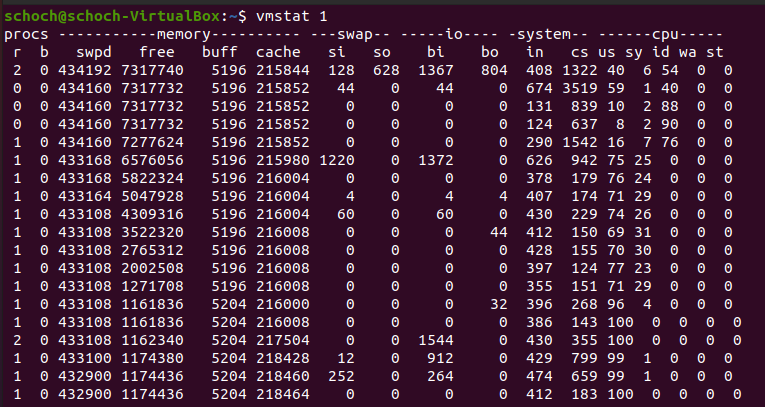
**4GB reserviert**

Nachdem wir nun mehrmals Speicher reserviert hatten, ist nun der erste geswappte Speicher ersichtlich in der „swpd“ Spalte.

Interessant ist hier, dass auf diesem System „so“ komplett 0 hat, was auch Sinn macht, da genügend Restspeicher vorhanden ist auf der Memory (3,1GB) und nichts geswapped werden muss. Si hingegen lädt ein paar Pages von der Disk zurück auf die Memory.

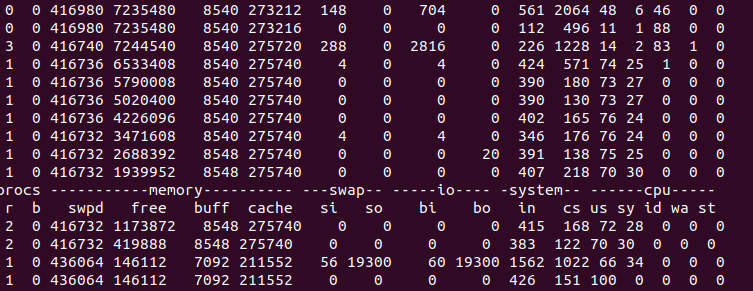
**5GB reserviert**

Während der Swap Speicher sich weiterhin langsam entleert, werden auch 5GB leicht reserviert. So bleiben danach noch 2,2GB Ram übrig.

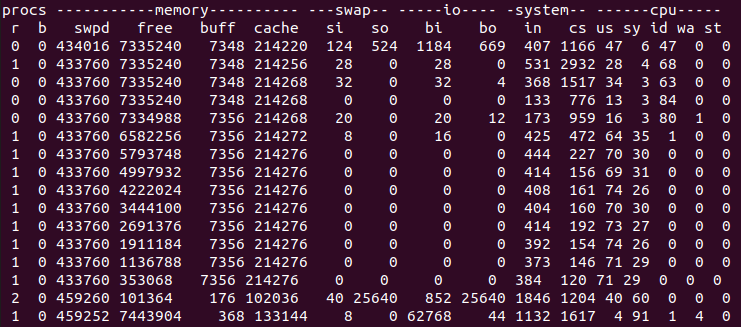
**6GB reserviert**

Der Swap Speicher wird weiterhin entleert, während das System auch 6 GB reserviert und danach noch 1,1GB übrighat.

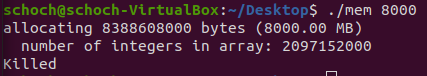
**7 GB reserviert**

Nachdem das System beinahe die gesamten 7 GB reserviert und nur noch 150 MB übrighat, wurden vom System 19300 kb (19,3 MB) geswapped auf die Disk, was man schön in der vorletzten Zeile sehen kann.

**8 GB reserviert**

Das System schafft es nicht 8 GB zu reservieren. Es versucht es jedoch, bis es merkt, dass weder die 459 MB Swapspeicher reichen, noch die restlichen 7,3 GB freier Speicher.

Daher gibt es im Anschluss Killed zurück.

****Virtual Box erlaubt es nicht mehr RAM zu reservieren, als das System zur Verfügung hat.

Daher funktioniert es auch nicht mehr als 8GB RAM zu reservieren.

**21.4**

**Beim Container: mit gleichem Screenshot wie bei 21.3**

Man kann sehen das bei steigender übergebenen MB zahl, steigt auch die Bi & Bo Zeit. Was logisch ist, da wir immer mehr swappen müssen bei größer werdendem Array. Die CPU utilization time verändert sich nicht großartig.

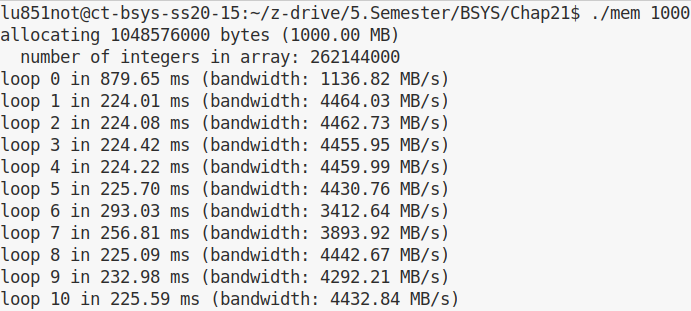
Bi- und Bo beschreibt Blocks, die zwischen Disk und Memory gesendet werden, in Blocks pro Sekunde.

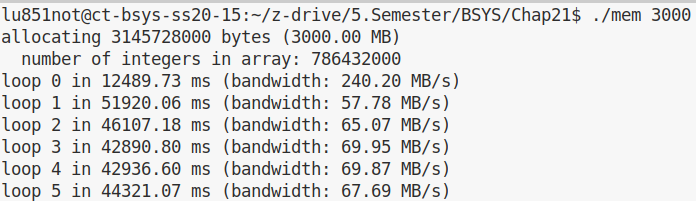
**I7 4790K 1 CPU-Kern + 980TI + 8GB Ram für die VM:**

Die Bi- und Bo-Blocks verhalten sich sehr ähnlich zu si und so. Dabei werden logischerweise die Blocks zwischen Disk und Memory geswapped werden.

**21.5**

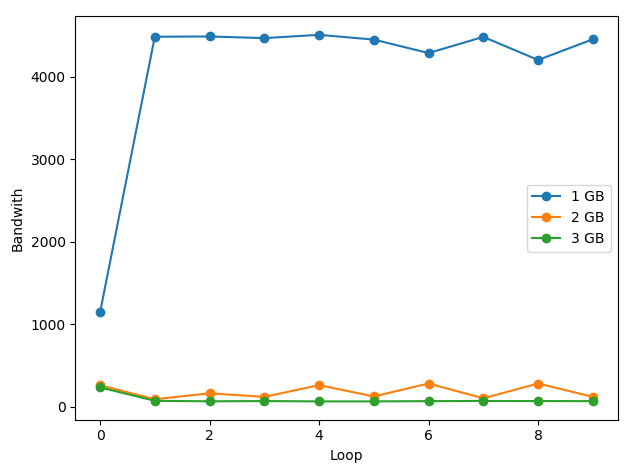
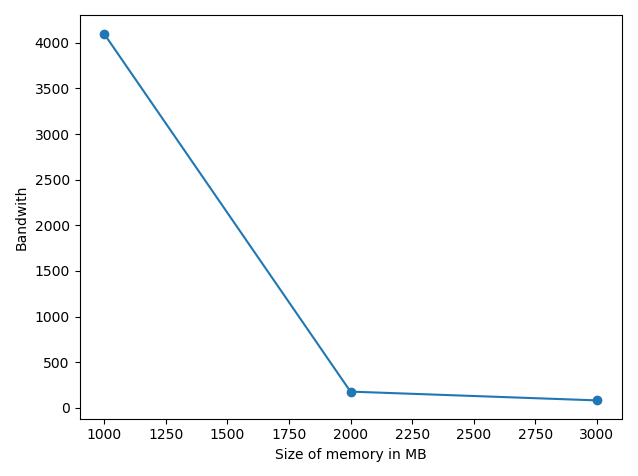
**Beim Container:**

****

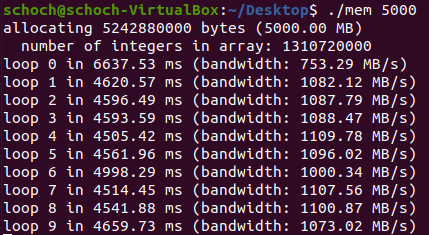
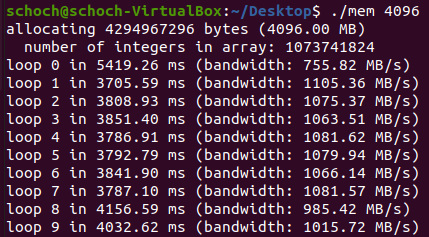
****

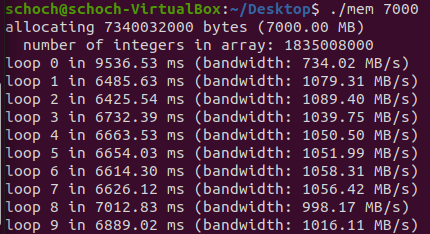
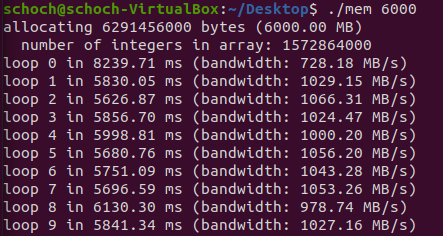
Beim ersten Fall, also wo wir eine Größe von einem 1 GB nehmen, kann man sehen, dass das ganze recht schnell abläuft. Da nicht sonderlich viel geswapped werden muss, daher ist die Bandbreite und Geschwindigkeit auch recht hoch. Man sieht auch das Loop 0 im Vergleich zu Loop 1 eine deutlich langsamere Bandbreite hat, dies liegt daran, dass dies der erste speicherzugriff ist und das ganze erst in den Cache muss, danach ist es deutlich schneller da es ja schon im Cache ist.

Beim zweiten Fall sieht es allerdings etwas anders aus, dort nehmen wir 3 GB als Größe. D.H es passt auf jeden Fall nicht in dem Ram rein und es muss geswapped werden und dies spiegelt sich auch in der Zeit und in der Bandbreite wider. Denn dadurch, dass immer auf die Disk gewartet werden muss, und diese nicht grade schnell ist im Vergleich zum Memory, dauert das ganze länger. Man sieht auch das Loop 0 im Vergleich zu Loop 1 deutlich schneller ist. Dies liegt daran, dass für die späteren Loops immer geswapped werden muss, daher dauert dies bei diesen länger.

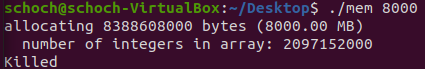


**I7 4790K 1 CPU-Kern + 980TI + 8GB Ram für die VM:**

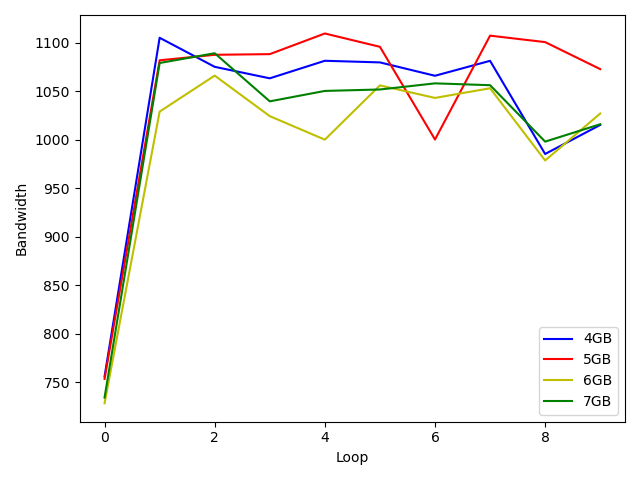
****

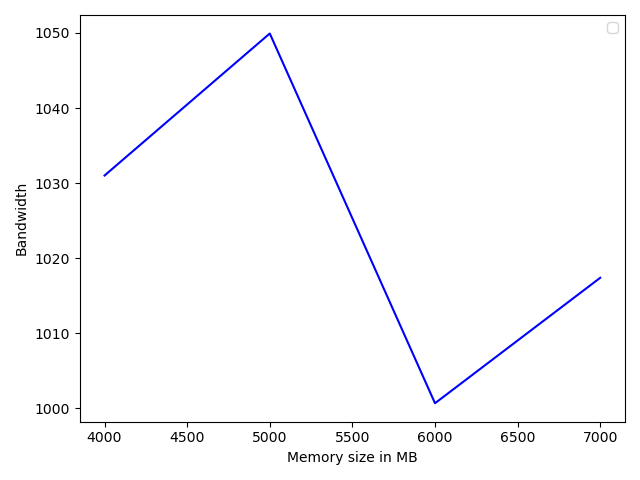
****

Die Reservierung von 4GB, 5GB, 6GB, und 7GB läuft sehr schnell ab und ungefähr der selben Bandbreite, da hierfür unsere Memory große genug ist. Loop 0 ist stets ca. 25% langsamer als die anderen Loops, da beim ersten Durchlauf noch keine Cacheeinträge vorhanden sind.

****

Da unser Swap Speicher sehr klein ist, kann aufwärts von 8GB kein Speicher reserviert werden. Bei einer Speicherreservierung von 8GB versucht das System noch den Speicher zu reservieren, bis es merkt, dass der Swap Speicher auch nicht ausreicht und beendet den Prozess. Bei 12GB hingegen, versucht das System es erst gar nicht und gibt die Fehlermeldung zurück: „Memory allocation failed“. Dies passiert, wenn das System NULL zurück gibt bei der Reservierung.

Hier kann man nochmal schön sehen, wie die Loops verlaufen. Wie bereits erwähnt, ist die erste Loop etwas langsamer und hat danach einen gleichmäßigen Verlauf.

Hier sehen wir zwar einen immensen Ausschlag, wenn man sich allerdings die Skalierung der y-Achse anschaut wirkt der Ausschlag nur minimal. Der Speicher wurde in allen 4 Versuchen sehr gleichmäßig reserviert.

**21.6**

**Beim Container: **

Wenn man mem.c mit Werten ab 4 GB aufruft wird es wie im Screenshot oben einfach gekillt. Die Grenze wird hier vermutlich die Größe des Speichers + Swap Space Größe sein also in unserem Fall: 2 GB + 3 GB = 5 GB. Allerdings nur wenn sonst nichts anderes laufen würde, was nun mal nicht passiert. Hier im Bild oben sieht man, dass vom Swap Space schon ca. 1 GB belegt sind, d.h. es bleiben 2 GB übrig. Wenn jetzt der RAM auch nicht komplett frei ist kommen wir ab 4 nicht mehr hin. 2 GB könnten geswapped werden aber der Rest müsste in den Hauptspeicher geladen werden und das geht nur wenn dieser komplett leer ist.

**I7 4790K 1 CPU-Kern + 980TI + 8GB Ram für die VM:**

****

Die Größe des Swapbereiches in der Disk, ist 459 MB groß. Interessanterweise ist der Swapbereich im Container 3 GB groß, also fast 6-mal so groß. Davon ist aufgrund der 7 GB Speicherreservierung nahezu alles belegt.

**21.7**

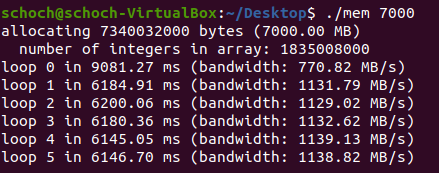
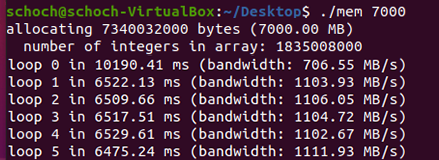
Es ist schwer die in-memory Performance vom RAM zu erreichen mit einer SSD oder einer HDD. Ein kleiner Statistikvergleich wie schnell meine unterschiedlichen Hardware Komponenten sind:

HDD: 130 MB/s

SSD: 530 MB/s

RAM: 16 GB/s

Wir haben die Ergebnisse in zwei identischen VM’s laufen lassen, die lediglich unterschiedliche Festplatten haben. Die eine mit SSD die andere mit HDD. Daher ist der Hauptfaktor beider Systeme nur die Geschwindigkeit der Festplatte.

Wir konnten zwar Swapspeicher aktivieren und deaktivieren auf den jeweiligen Systemen, aber leider keine andere Partition zuweisen, da wir einer VM leider keine mehrfachen Festplatten geben konnten.

Wie zu erwarten ist die SSD etwas schneller. Um genau zu sein, 30MB/s Warum ist die SSD lediglich nur ein paar MB/s schneller? Dies könnte an unserem sehr kleinen Swapspeicher liegen. Wir haben nur einen sehr kleinen Swapspeicher mit einer Größe von 459MB. Da die SSD 5-mal so schnell ist wie die HDD und nur so einen kleinen Unterschied macht, ist es bei unserem kleinen Swapspeicher nahezu unmöglich an die Geschwindigkeit des RAMS von 16GB/s ranzukommen. Wäre der Swapspeicher größer, dann wäre vermutlich auch die Bandweite des Systems mit SSD schneller.