**Отчет по лабораторной работе №5**

Общие характеристики:

Система – Red Hat (64bit) («виртуалка» на Oracle VM с выделенными ~2ГБ оперативной памяти).

Всего доступной оперативной памяти 1.8ГБ в ненагруженной системе, в действительности же со всеми запущенными процессами свободной памяти к моменту запусков скриптов будет около 1.65ГБ.

Размер страницы виртуальной памяти = 4096КБ = 4МБ.

Параметры раздела подкачки (swap):

Size = 839676KB =~ 820MB

Used = 80156KB

**Эксперимент №1)**

**Этап 1)**

1. Запуск без слежения:  
    скрипт аварийно завершился с записью “Out of memory: Killed process6604 (mem.bash) total-vm:2635476kb, anon-rss:1662888kb, file-rss:0kb, shem-rss:0kb, UID:0 pgtables:4792kb oom\_score\_adj=0”.

Из этих параметров интересен anon-rss =~ 1.6ГБ реально выделенной оперативной памяти на момент убийства процесса.

1. Запуск со слежением:

Изначально размер параметров свободной памяти системы, следующий:

free Mem (оперативная память) = 1800MB (из них доступно 1.6ГБ), а free Swap (подкачка) = 730MB (остальные параметры здесь особо не меняются).

Далее до значения ~70MB (будем называть это «точкой перехода») free Mem снижается примерно на 3–4% (60-70MB) в секунду, и далее колеблется вокруг этого значения, а free Swap в свою очередь начинает падать после падения free Mem до 70MB, но падает в абсолютных цифрах несколько медленнее (55-65MB).

Значения параметров самого работающего скрипта меняются со следующей зависимостью:

До точки перехода соответственно Virt и Res растут на 65000-70000KB (соответствую уменьшению free Mem), СPU держится в районе 76–88%, а MEM увеличивается на 3.5–4.5%.

После точки перехода соответственно Res и Mem почти не меняются, держась на (1.6g и 87%), Virt растет на 60000KB в сек, CPU резко опускается до 40%, а далее поднимается до 60–70% и держится на этих значениях.

Верхние 5 процессов (systemd, x2top, system-journal, следящий скрипт “myWatcherScript”), не считая mem.bash, в сортировке по %MEM почти не меняются и занимают по 0.2%, иногда top-ы и следящий скрипт чуть колеблются; после точки перехода процессы ненадолго сбрасывают потребление до 0.1% и systemd становится ниже system-journal, а также появляется sh.

Report.log последнее значение размера массива до аварийной остановки X = 30000000.

Запись в логах dmesg аналогична вышеприведенной.

Графики:

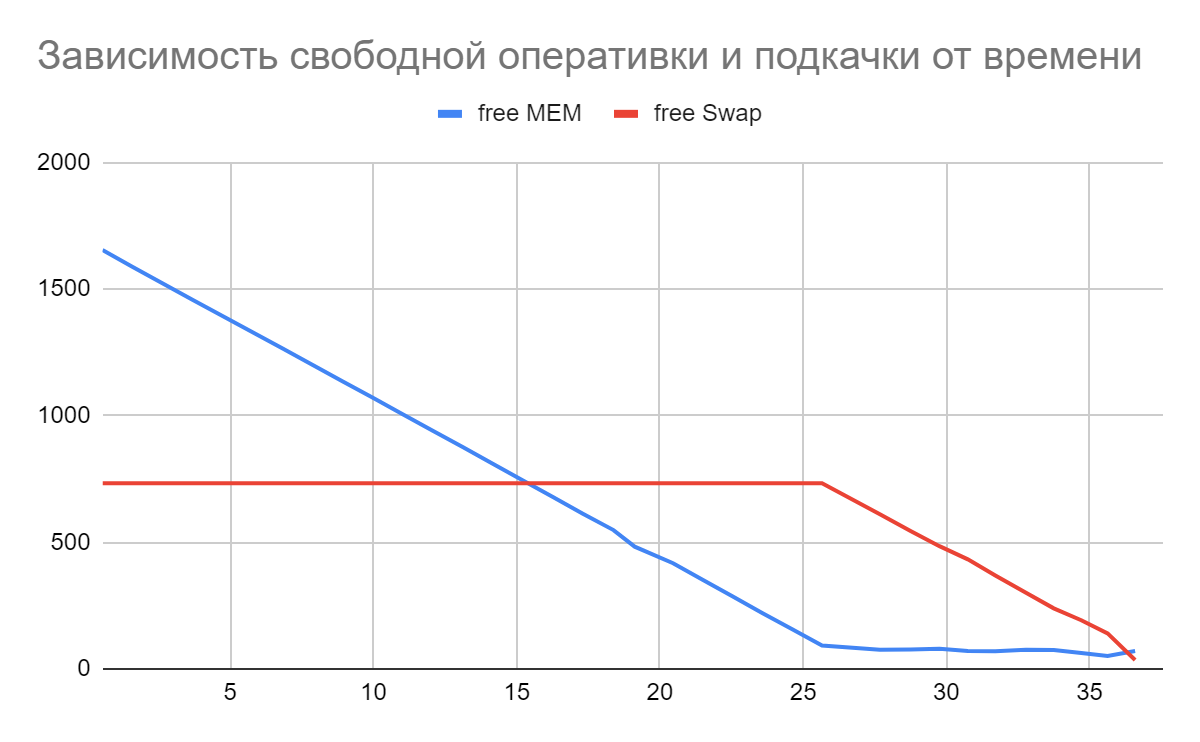
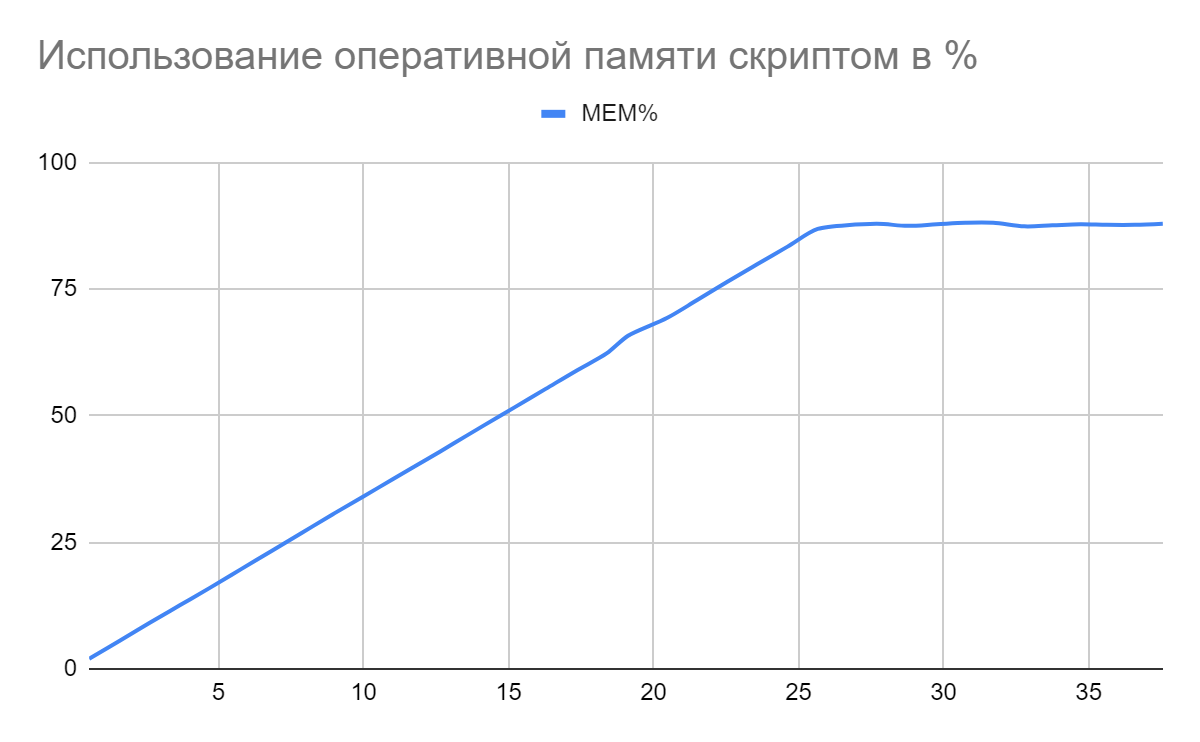
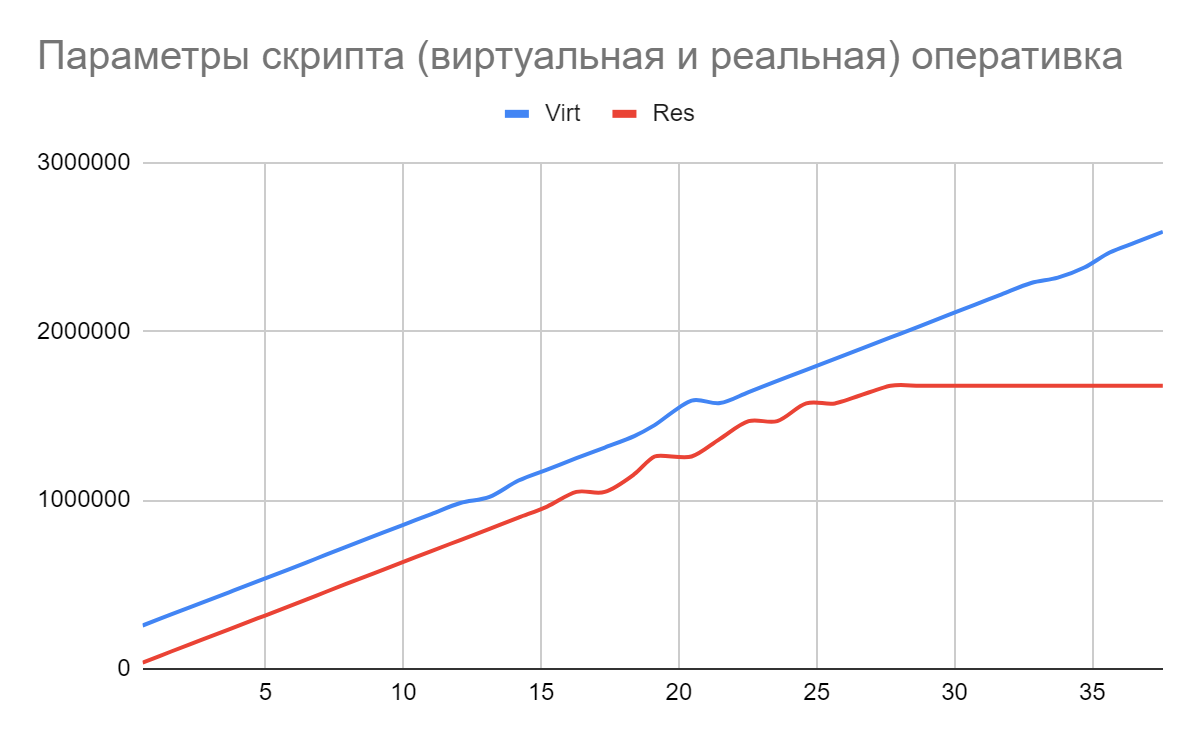


График 1 (Эксперимент 1.1)

На этом графике хорошо видно, что сначала для работы процесса выделялась реальная оперативная память (подкачка не тратилась), в момент дохода почти до нуля (в точке перехода) она перестала тратиться (осталось около 70MB на различные потенциальные системные нужды, в том числе на нужды, например киллера) и затем тратилась уже подкачка, причем с чуть меньшей скоростью (на 5–10%) из-за более долгого обращения. После дохода же до нуля подкачки – процесс убили.

График 2 (Эксперимент 1.1)

Здесь все понятно – зависимость использования памяти линейная и после точки перехода константная (т. к. по сути вся оперативка к тому моменту уже отдана процессу).

График 3 (Эксперимент 1.1)

Virt (кбайты всех страниц виртуальной памяти, выделенных скрипту) – общая «виртуальная» память (сумма оперативки, подкачки со сдвигом) отданная скрипту и на некоторую константу (~0.3ГБ) сдвинутая относительно реально задействованной памяти. Понятно, что на этапе до точки перехода она состоит из оперативки и растет параллельно ей, а после состоит из константно отданных 1.6ГБ оперативки и растущей подкачки.

В итоге Virt (что и зафиксировано в логах после убийства mem.bash) достигает 1.6 (реальная оперативная память, anon-rss) + 0.7 (подкачка) + 0.3 (примерная виртуальная константа в этом случае) =~ 2.6Гб виртуальной памяти (total-vm).

**Этап 2)**

Общая хронология, следующая (в принципе все аналогично одному запущенному процессу): сначала работали оба скрипта и вели себя в сумме, грубо говоря, как один скрипт до своей точки перехода (тратили реальную оперативку), затем потратив всю оперативку они начали тратить подкачку, затем, потратив подкачку один скрипт (потребляющий немного большее значение памяти) аварийно завершился, и освободилась вся подкачка и, главное, оперативная память, зарезервированная за убитым первым скриптом. Далее оставшийся скрипт вел себя так как будто был запущен сразу в одиночестве, но с меньшими значениями свободной оперативной памяти и подкачки.

В целом ничего особенно не изменилось, сначала free Mem опускается примерно на 70МБ в сек (около 4%), а при падении до минимума колеблется все также около 70МБ, далее освобождается около 800MB оперативки после смерти первого процесса и free Mem снова начинает падать до 70МБ. С подкачкой аналогично.

Virt, Res ведут себя все так же, но примерно в 2 раза медленнее когда работают оба процесса, т.к. free Mem и free Swap распределяются примерно поровну на два процесса.

MEM соответственно Res также распределяется равномерно на 2 процесса, а после остановки одного из них, концентрируется на оставшемся. CPU аналогично.

С верхними 5-ю процессами по мимо наших все аналогично первому этапу.

Вывод dmesg (сначала сообщение об аварийной остановки первого по out of memory, потом второго):

Out of memory: Killed process 4665 (mem2.bash) total-vm:1429268kB, anon-rss:833224kB, file-rss:0kB, shmem-rss:0kB, UID:0 pgtables:2440kB oom\_score\_adj:0 [17650.3901041 oom\_reaper: reaped process 4665 (mem2.bash), now anon-rss:0kB, file-rss:0kB, shmen-rs.

s:0kB

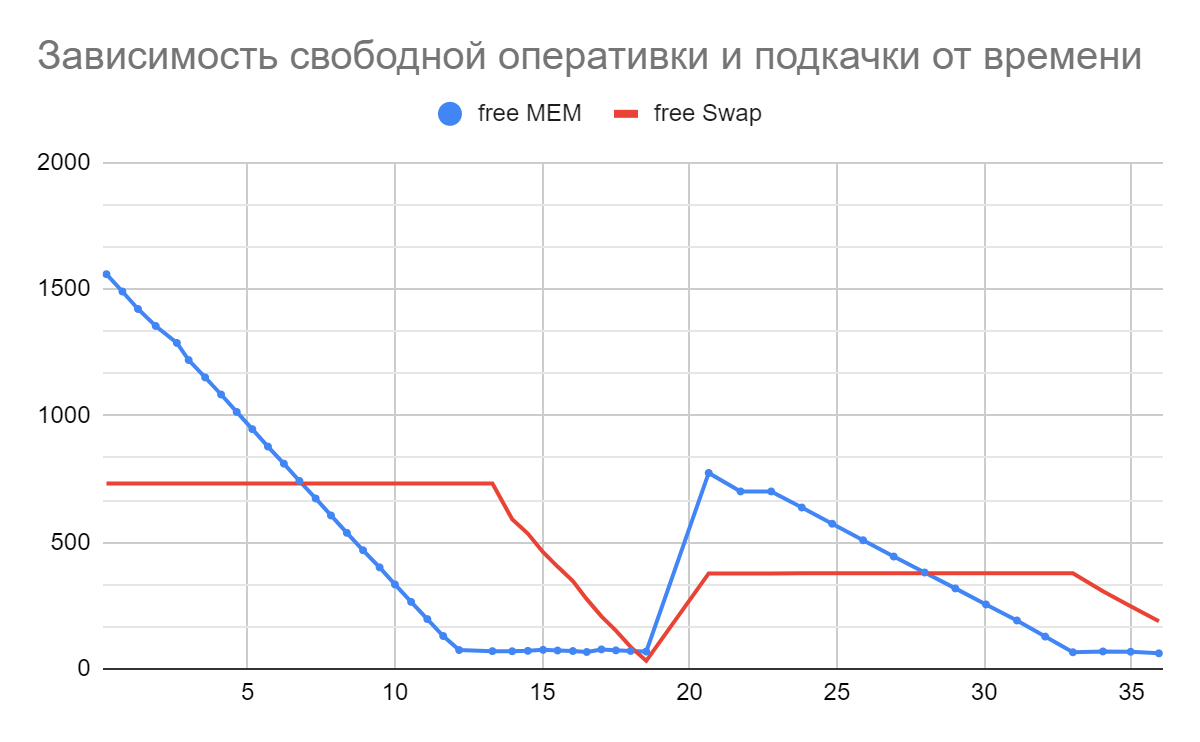
Out of memory: Killed process 4664 (mem.bash) total-vm:2612772kB, anon-rss:1641972kB, file-rss:0kB, shmem-rss:0kB, UID:0 pgtables:4752kB oom\_score\_adj:0 [17677.5564171 oom\_reaper: reaped process 4664 (mem.bash), now anon-rss:0kB, file-rss:0kB, shmem-rss

:0kB

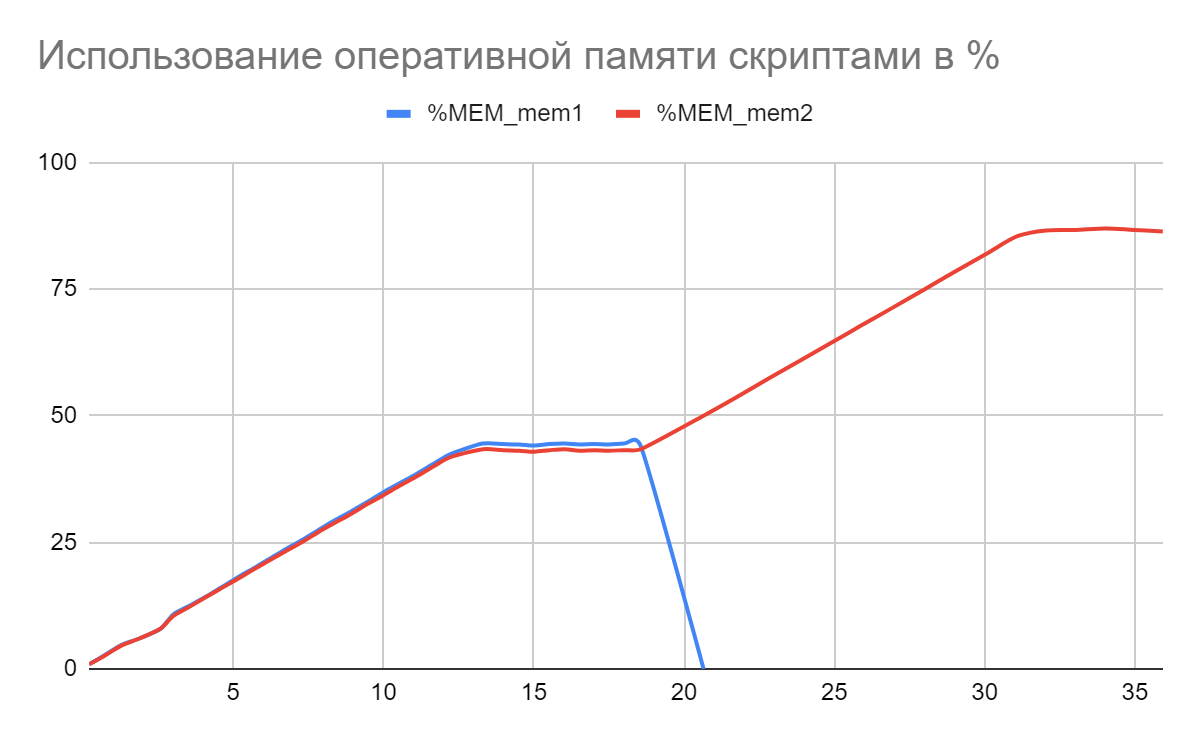
Report.log последнее значение размера массива до аварийной остановки X = 30000000.

Report2.log последнее значение размера массива до аварийной остановки Y = 15000000.

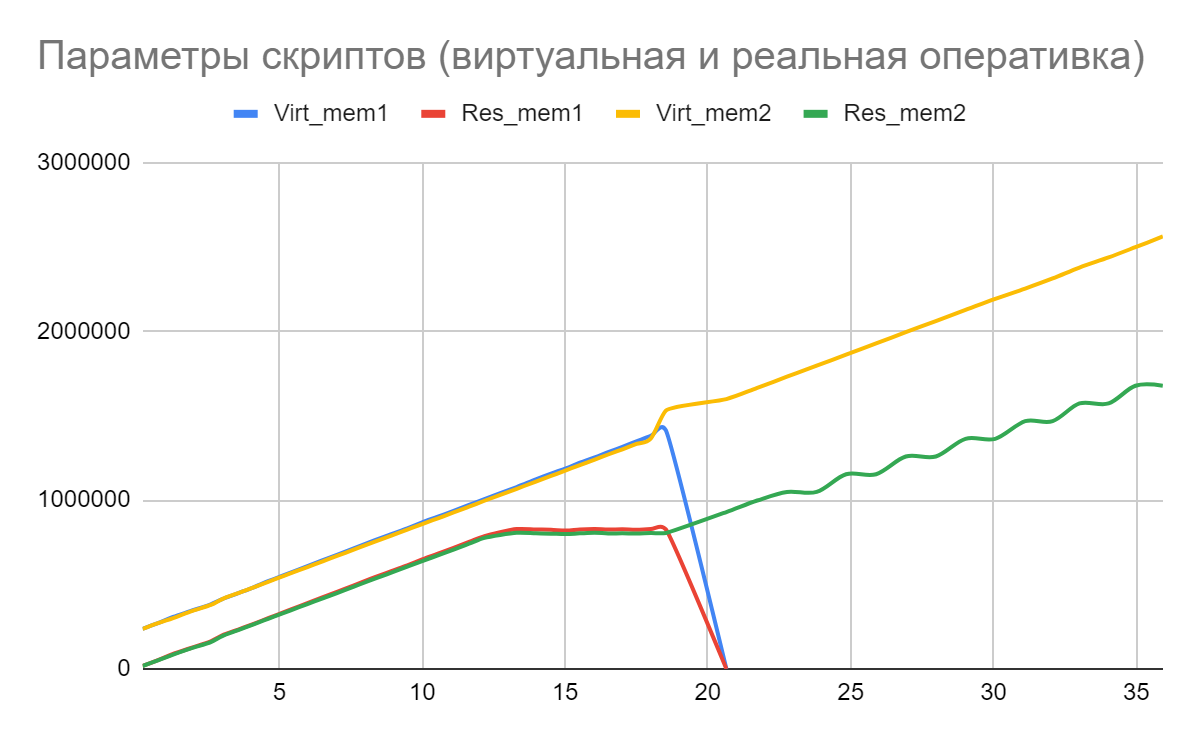
Графики:

График 4 (Эксперимент 1.2)

На этом графике четко видно 5 различных участков зависимости free Mem и free Swap от времени. Оперативная память падает – держится на константе 70МБ – высвобождается – снова падает до 70. Подкачка не тратится – резко начинает тратиться при нехватке оперативки – кончается и высвобождается – снова не тратится до проблем с оперативкой – падает до нуля.

График 5 (Эксперимент 1.2)

Все аналогично (в сумме на пике дают примерно 90). Зависимость на участках роста – линейная.

График 6 (Эксперимент 1.2)

Все обратно зависимостям free Mem и free Swap. Virt\_mem1 и Virt\_mem2 линейно растут сначала за счет оперативки (процессам выделяются страницы памяти) затем за счет подкачки до критического момента – затем один процесс убивают и оставшийся постепенно забирает себе его память опять же сначала оперативку – затем подкачку.

**Выводы**: результаты экспериментов полностью соответствуют модели со страничной организацией памяти. На отдельных участках все зависимости линейны, страницы памяти и подкачка выделяются и распределяются относительно равномерно (подкачка чуть медленнее). Сначала всегда тратится оперативная память – затем подкачка. Есть четкие логические точки перехода. Не трудно заметить, что размер массива (Y) у первого убитого процесса на 2 этапе эксперимента ровно в два раза меньше (X) убитого вторым (или убитого на 1 этапе), что опять же доказывает равномерность распределения ресурсов и их полное (до константы) итоговое распределение.

**Эксперимент №2)**

Запуск на параметрах N = 3000000 и K = 10 действительно проходит без аварийных остановок, что логично т. к. 30000000 / 10 = 3000000 (даже в худшем случае здесь все хорошо).

Запуск же на параметрах N = 3000000 и K = 30 приводит к нескольким (16 на моем устройстве) аварийным остановкам, что с первого взгляда логично, т. к. 3000000 \* 30 сильно больше 30000000 и в пиковой точке неизбежно всем может не хватить памяти (с другой же стороны, нужно учитывать более сложную систему распределения памяти и главное то, что скрипты запускаются по условию не подряд, а раз в секунду, из-за чего первые скрипты вполне могут успеть завершиться до потенциальных аварийных остановок вызванных запущенными более поздними скриптами).

Попробуем определить значение N при котором не будет возникать аварийных остановок (для этого я воспользовался своеобразным бинарным поиском).

Итого, найденным ответом оказалось N =~ 2230000.

**Вывод:**

Для установления объяснения я провел еще один эксперимент (он не требовался в задании), запустив все скрипты одновременно (без задержки в секунду), и получил что при N1 = 1000000 аварийных остановок нету (30 N1= 30000000, а это немного меньше фактического значения массива при запуске одного скрипта, т. к. в логах мы фиксировали только кратные 100000), а при N2 = 1100000 вылеты иногда уже есть (т.к. потенциально N2 \* 30 может быть больше 30000000 + eps). Соответственно, отсюда очевидно доказательство основной причины столь большого различия между этими двумя запусками. Большую роль все-таки играет то, что скрипты запускаются не подряд, а каждую секунду. Такой же эффект вносит и не абсолютное равномерное распределение процессорного времени.