МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

Санкт-петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа №5

По дисциплине Операционные системы

Выполнил студент группы №M3212:
Дубов Данил Денисович

Проверил:
Дюкарева Вероника Максимовна

САНКТ-ПЕТЕРБУРГ
2020

Цель работы:

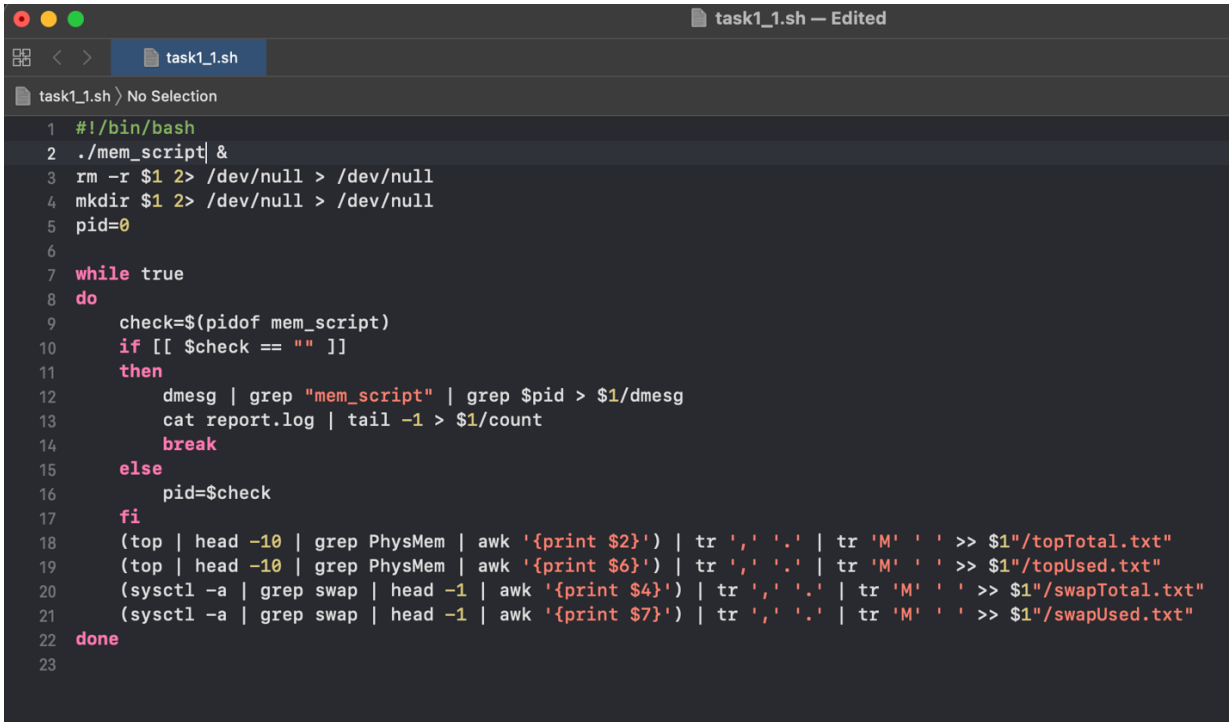
1. Рассмотреть использование утилиты top для мониторинга параметров памяти

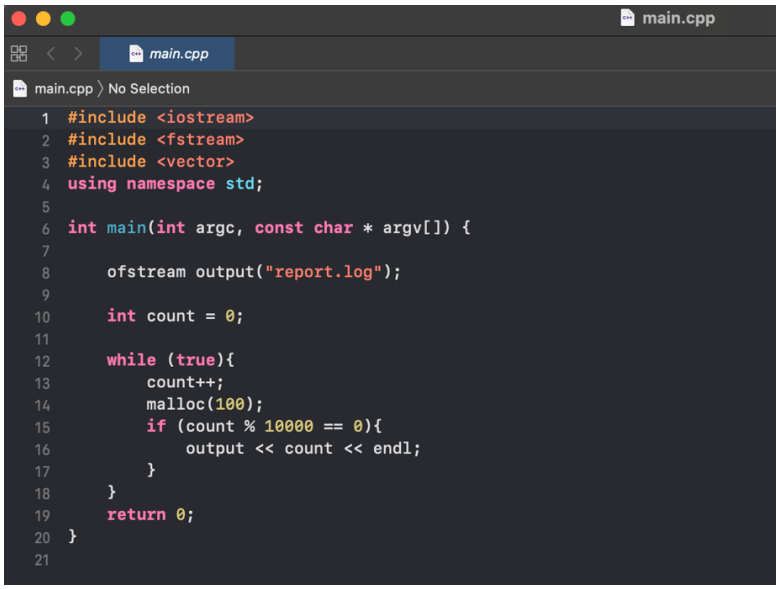
2. Рассмотреть использование имитационных экспериментов для анализа работы механизмов управления памятью

Сначала начал делать на mac

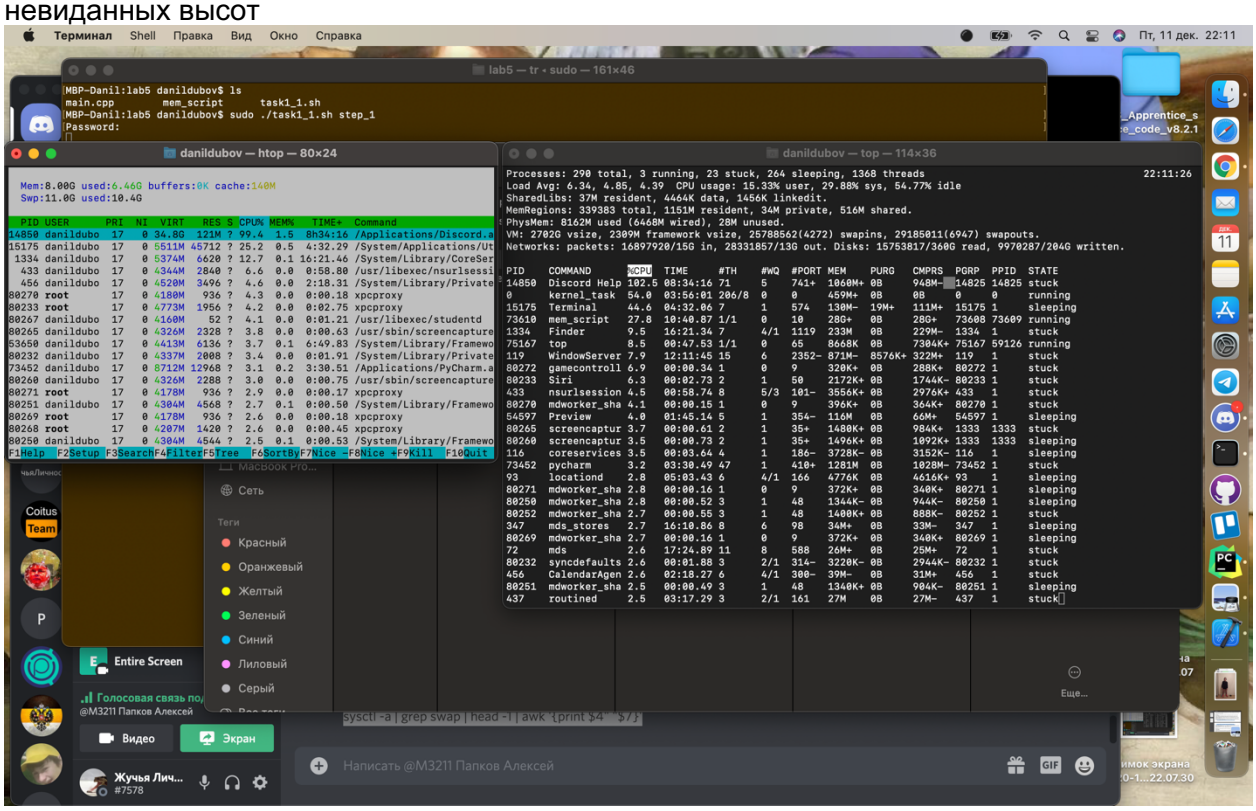
Эксперимент 1 шаг 1:

Был создан скрипт mem_script, который нагружает память, taskn_n.sh, чтобы анализировать данные из топ и sysctl, main.py скрикп, который из полученных данные строит графики.

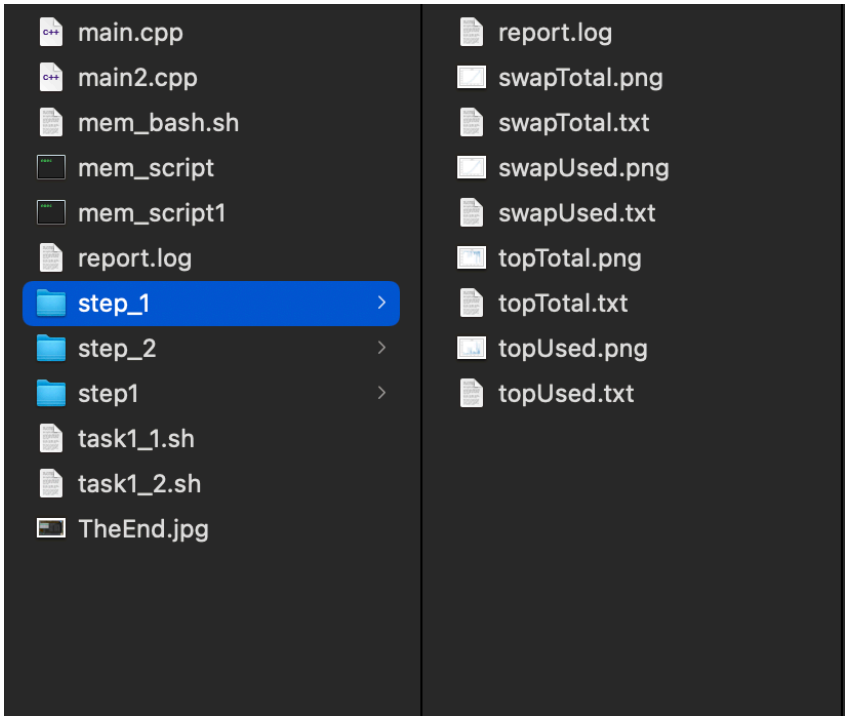




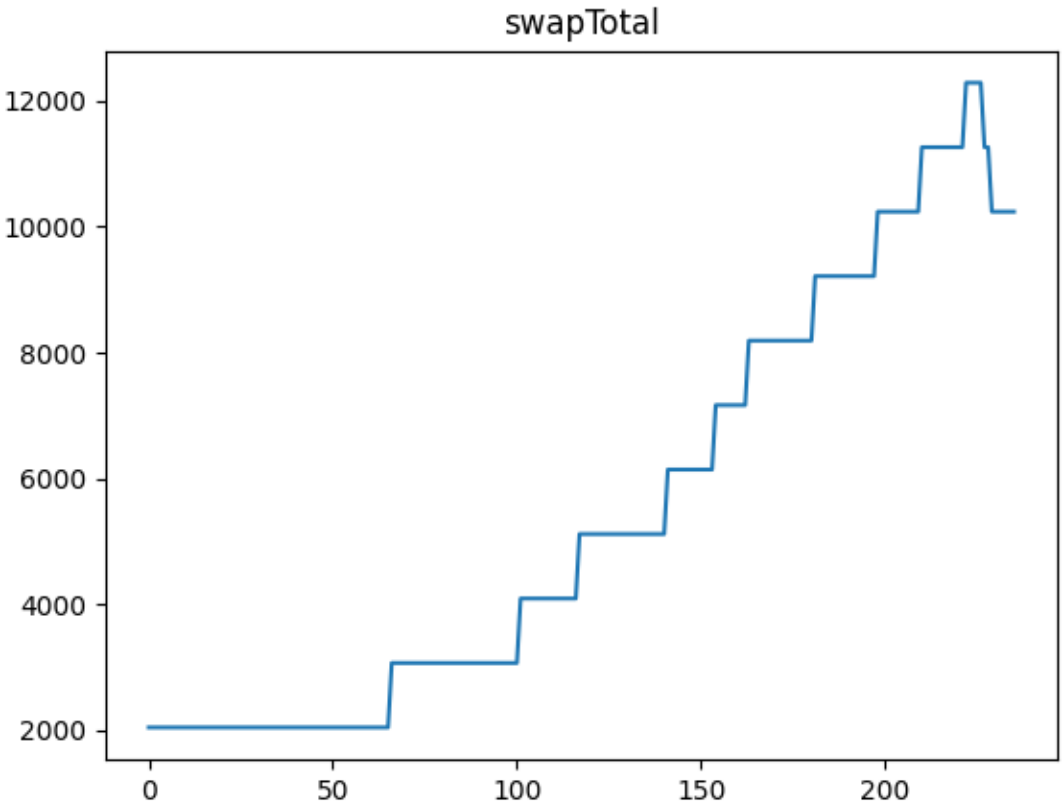
В результате работы скрипта task1_1.sh OOM-Killer не сработал и память загрузилась до невиданных высот

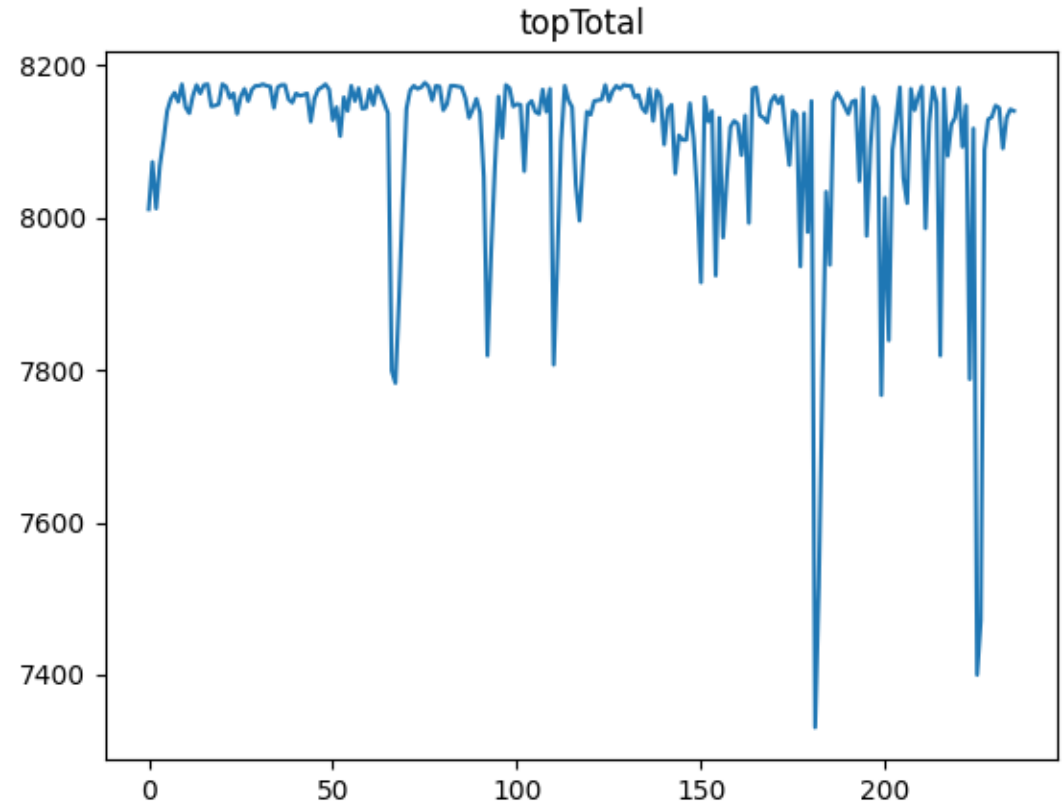


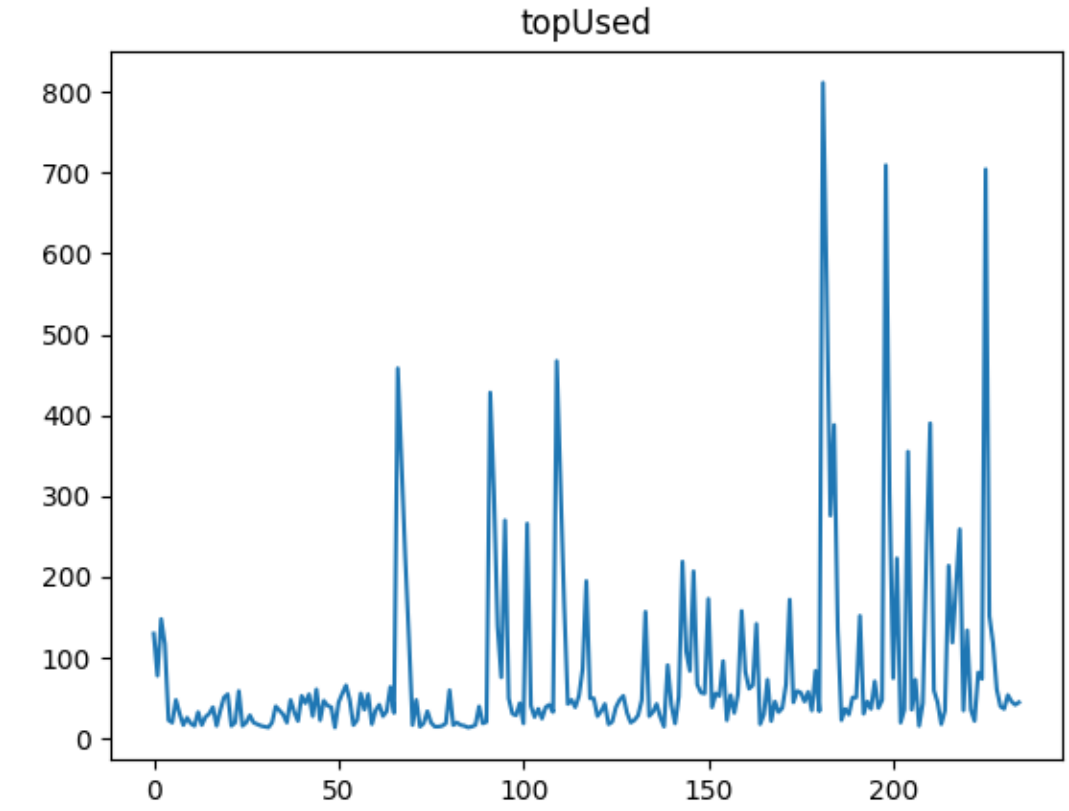
и была получена директория с файлами:

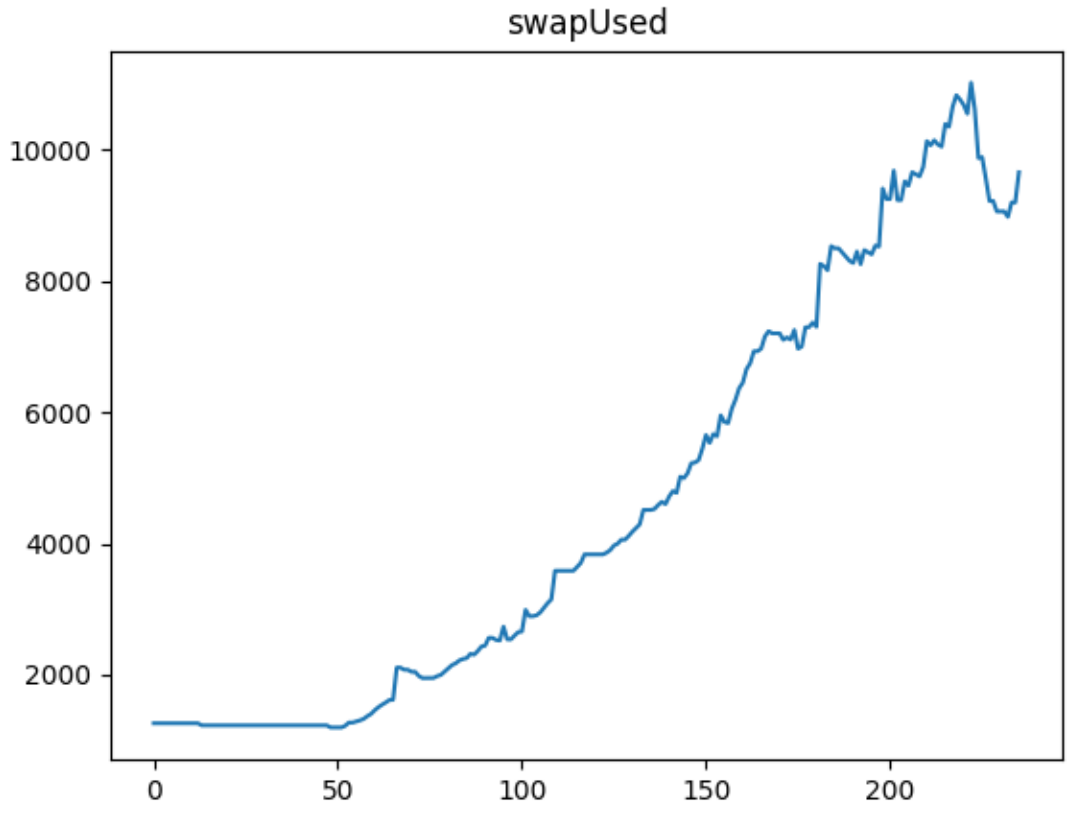


Графики:


swapTotal
12000
10000
8000
6000
4000
2000
0
50
100
150
200


topTotal
8200
8000
7800
7600
7400
0
50
100
150
200


topUsed
800
700
600
500
400
300
200
100
0
0
50
100
150
200


swapUsed
10000
8000
6000
4000
2000
0
50
100
150
200

Шаг 2 для эксперимента 1:

Был изменен скрипт task1_1.sh ---> task1_2.sh (запускаем два cpp скрипта)

task1_2.sh — Edited

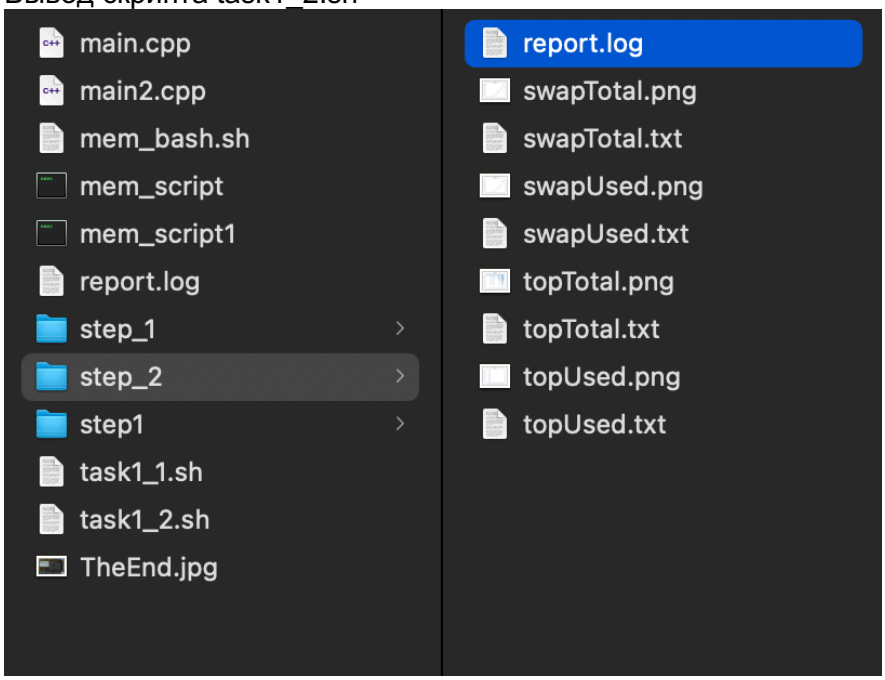
task1_2.sh

task1_2.sh 〉 No Selection

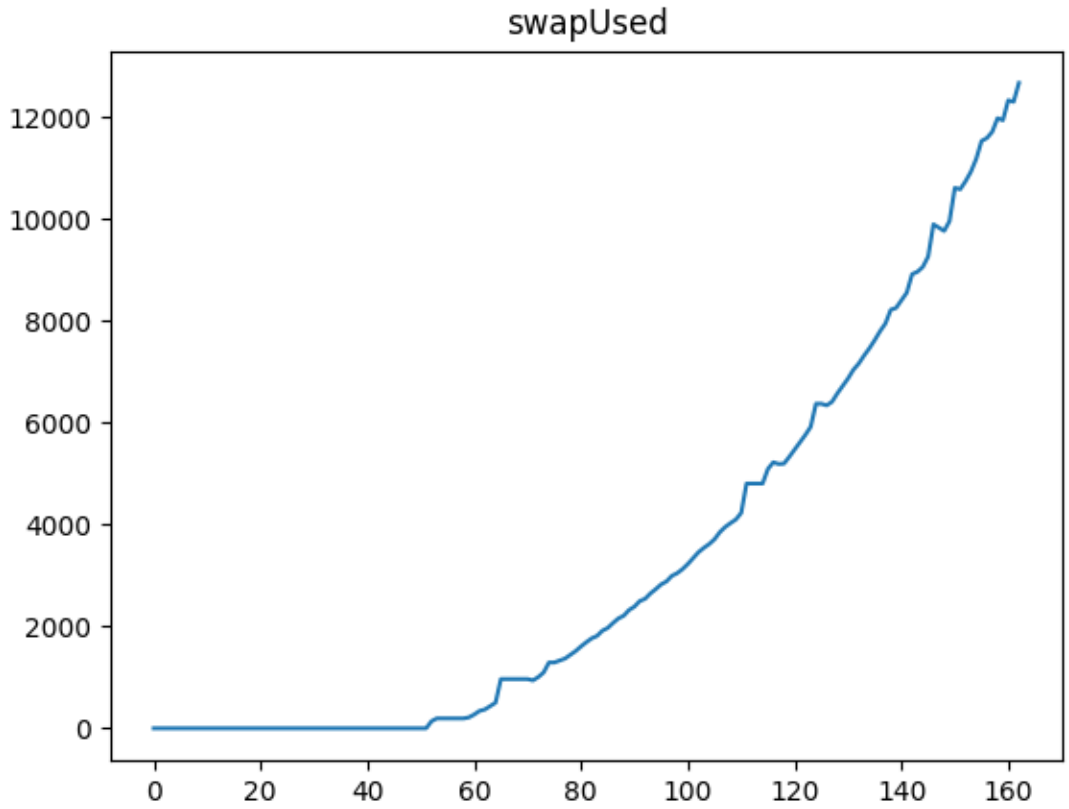
```
#!/bin/bash
./mem_script &
./mem_script1 &
rm -r $1 2> /dev/null > /dev/null
mkdir $1 2> /dev/null > /dev/null
pid1=0
pid2=0
while true
do
    check=$(pidof mem_script)

    if [[ $check != "" ]]
    then
        pid1=$check
    fi

    check=$(pidof mem_script1)

    if [[ $check != "" ]]
    then
            pid2=$check
        fi

    if [[ $(pidof mem_script) == $(pidof mem_script1) ]]
    then
        cat report.log | tail -1 >> $1/count
        cat report1.log | tail -1 >> $1/count
        dmesg | grep "mem_script" | grep $pid1 >> $1/dmesg
        dmesg | grep "mem_script1" | grep $pid2 >> $1/dmesg
        break
    fi
    (top | head -10 | grep PhysMem | awk '{print $2}') | tr ',' '.' | tr 'M' ' ' >> $1"/topTotal.txt"
    (top | head -10 | grep PhysMem | awk '{print $6}') | tr ',' '.' | tr 'M' ' ' >> $1"/topUsed.txt"
    (sysctl -a | grep swap | head -1 | awk '{print $4}') | tr ',' '.' | tr 'M' ' ' >> $1"/swapTotal.txt"
    (sysctl -a | grep swap | head -1 | awk '{print $7}') | tr ',' '.' | tr 'M' ' ' >> $1"/swapUsed.txt"
done
```

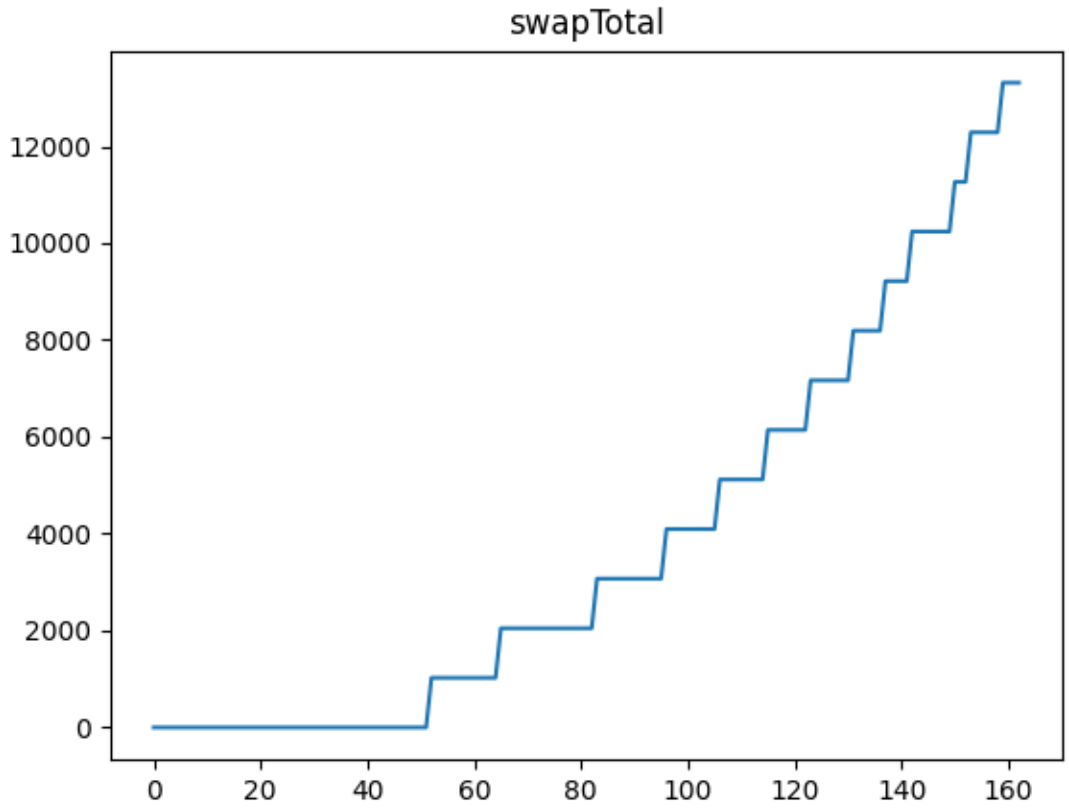
Mem_script был оставлен без изменений, результат. Работы скрипта task1_2.sh так же Меня очень сильно огорчил (OOM-Killer так же не заработал)

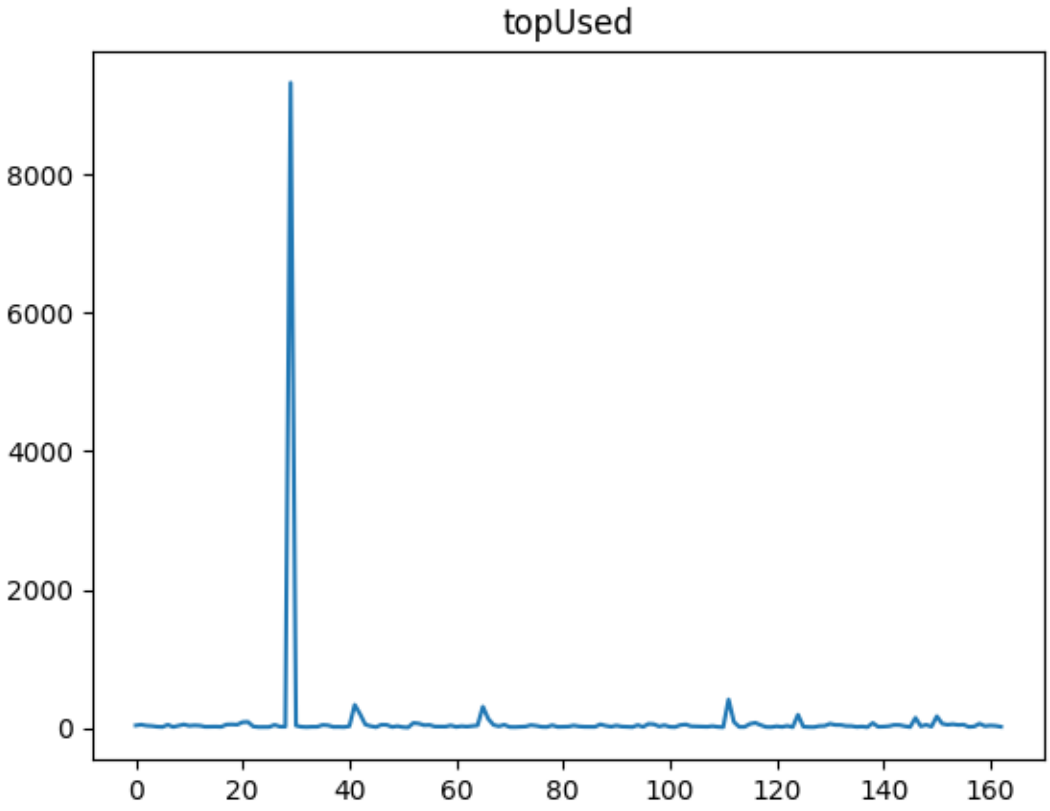
Вывод скрипта task1_2.sh

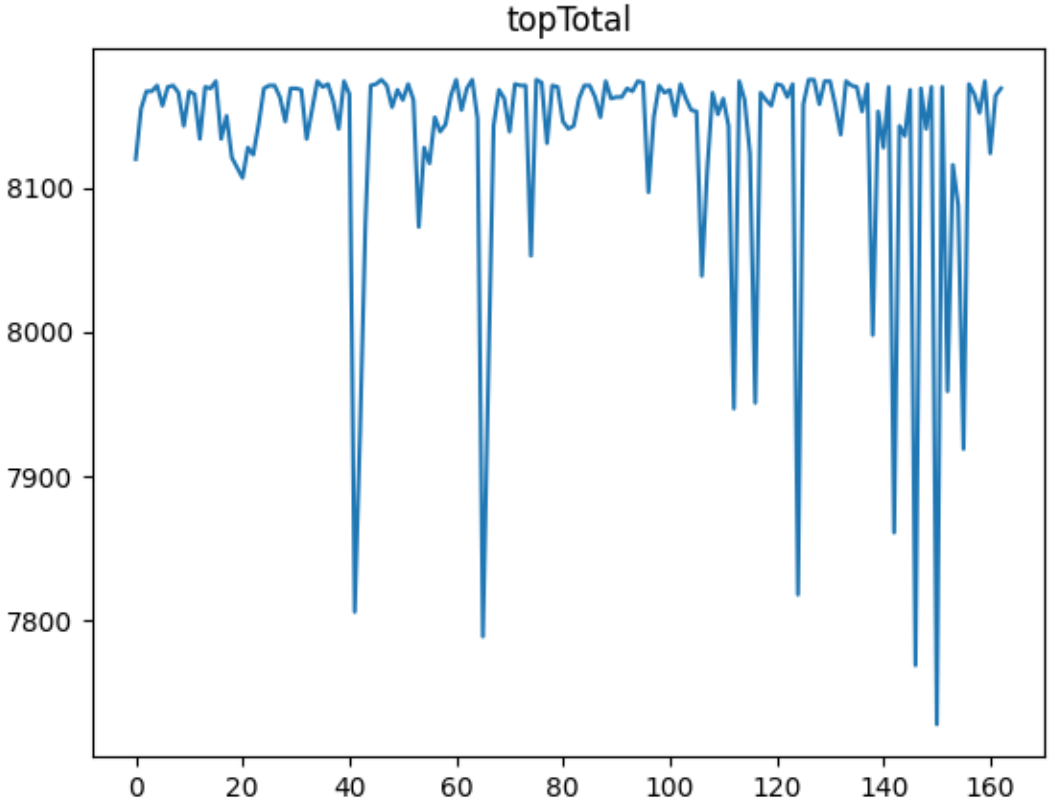


Графики:


swapUsed
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120
140
160


swapTotal
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120
140
160


topUsed
8000
6000
4000
2000
0
0
20
40
60
80
100
120
140
160


topTotal
8100
8000
7900
7800
0
20
40
60
80
100
120
140
160

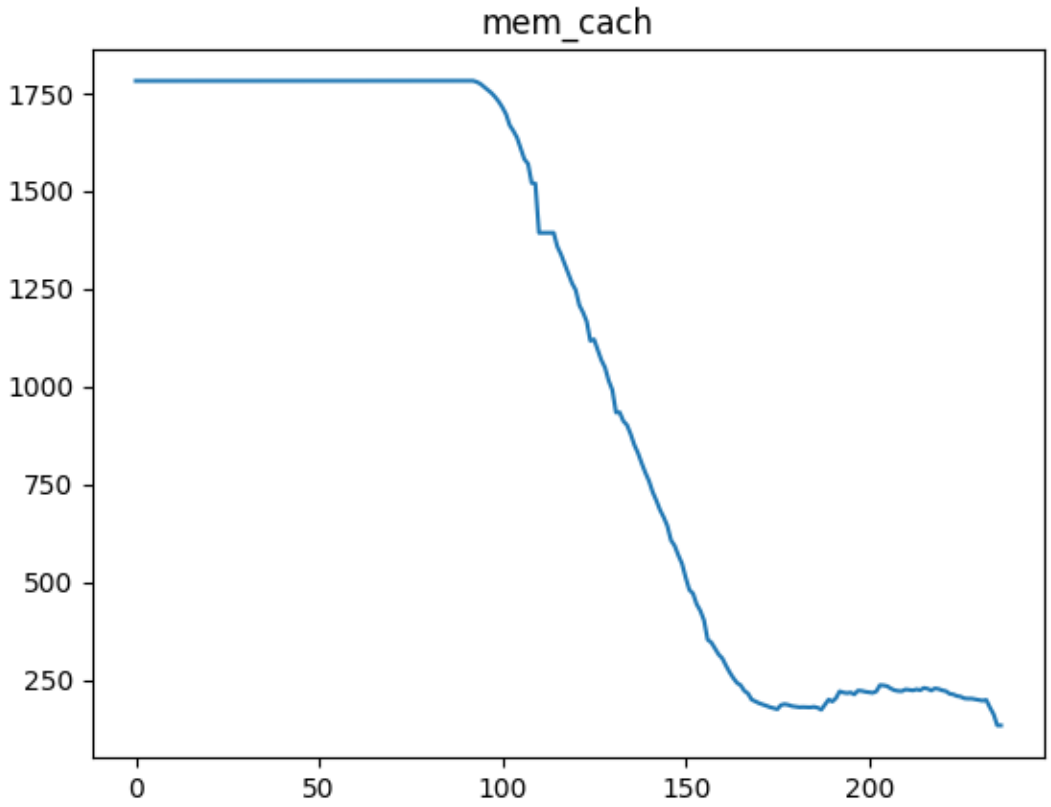
Вывод: в ходе выполнения данных экспериментов на macOS, не было замечено аварийных остановок процесса, в связи с тем, что файл подкачки изменяет свой размер. [В данном видеоролике видно что OOM-Killer на macOS работает очень странно]

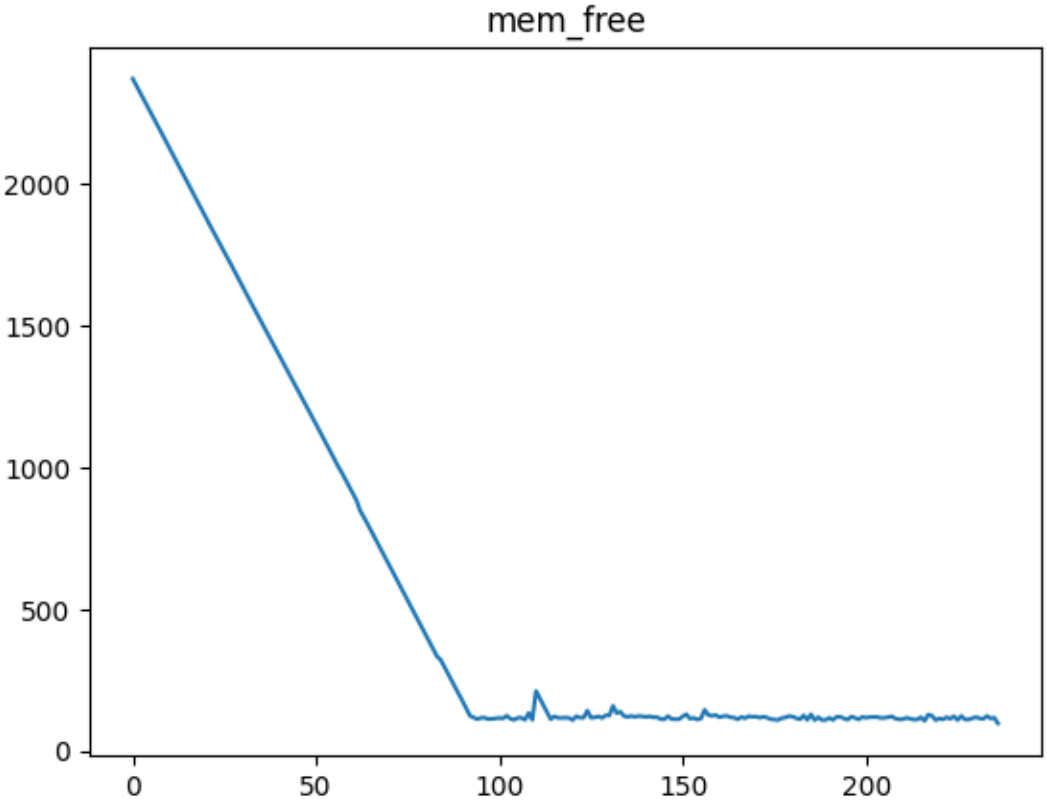
Подключился через SSH соединение к компьютеру с Linux. Выполнил те же самые операции.

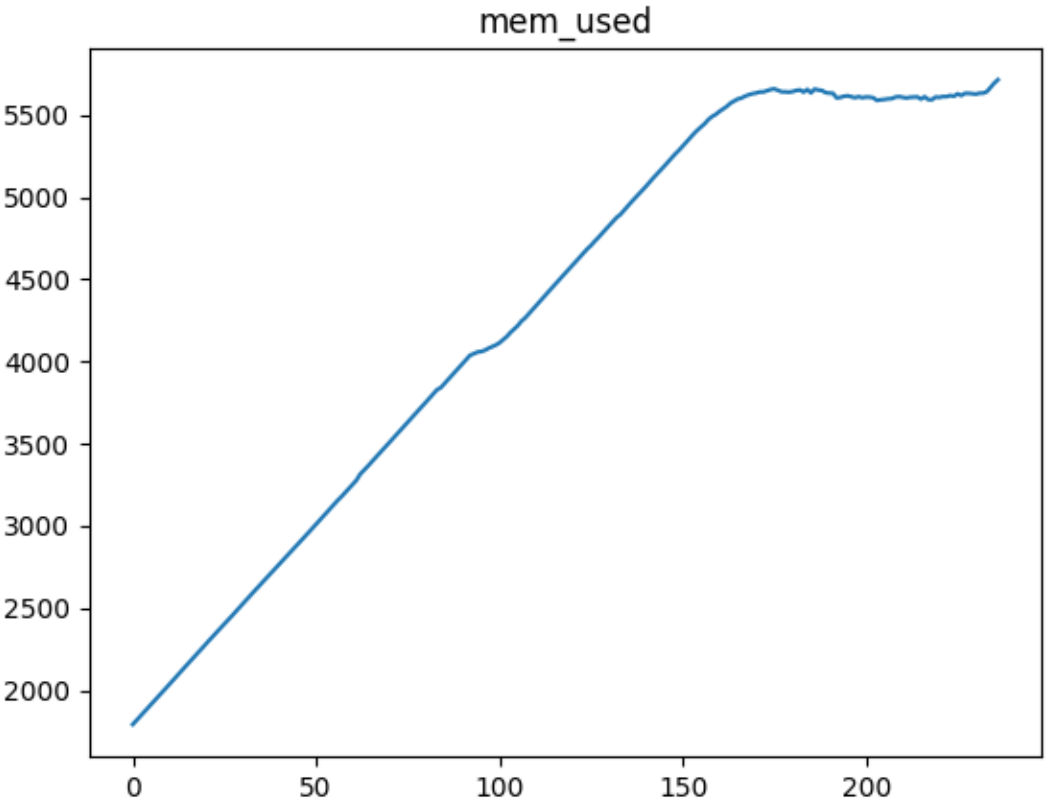
Изменил парсинг для MiB Swap и MiB Mem

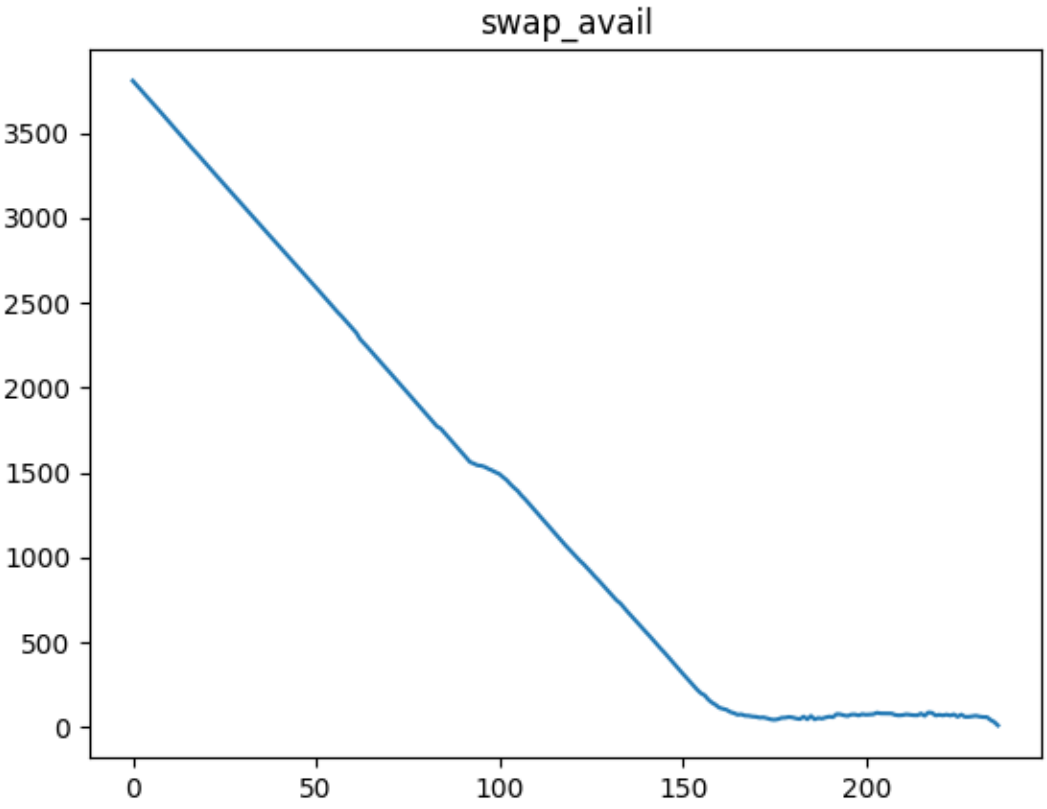
```
MiB Mem :   5945,5 total,   2012,8 free,   2253,2 used,   1679,5 buff/cache
MiB Swap:   2048,0 total,   2048,0 free,      0,0 used.   3272,3 avail Mem
```

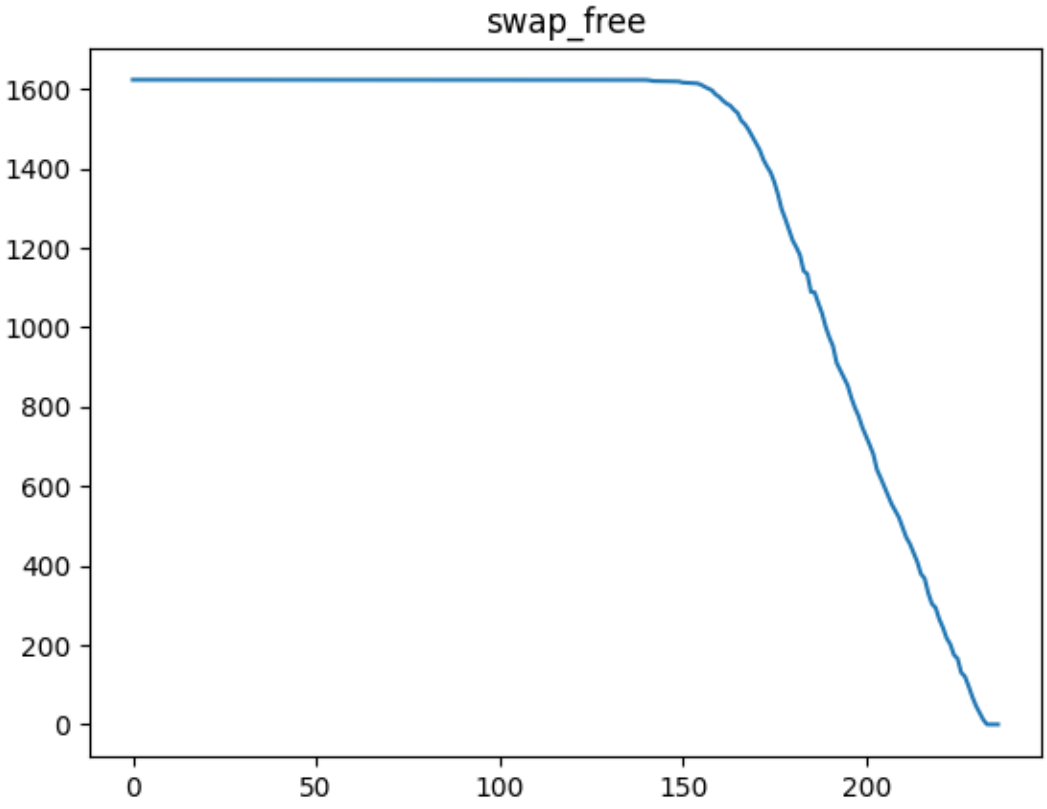
Эксперимент 1 шаг 1:

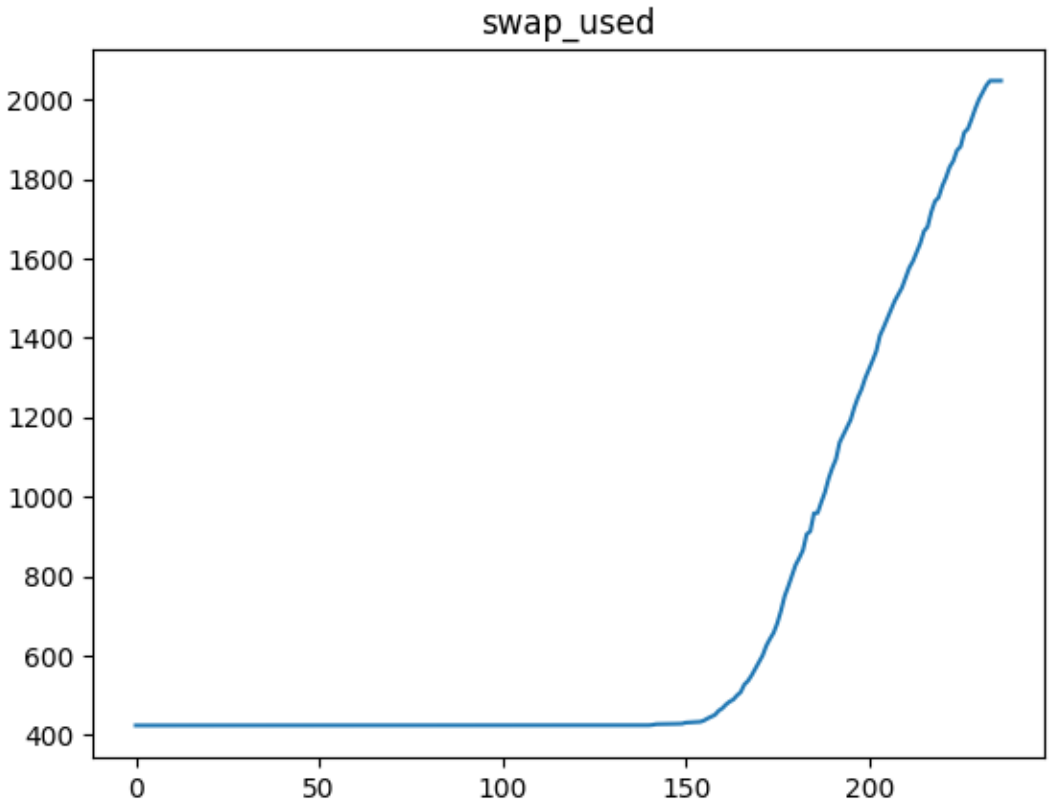



mem_free
2000
1500
1000
500
0
0
50
100
150
200


mem_used
5500
5000
4500
4000
3500
3000
2500
2000
0
50
100
150
200

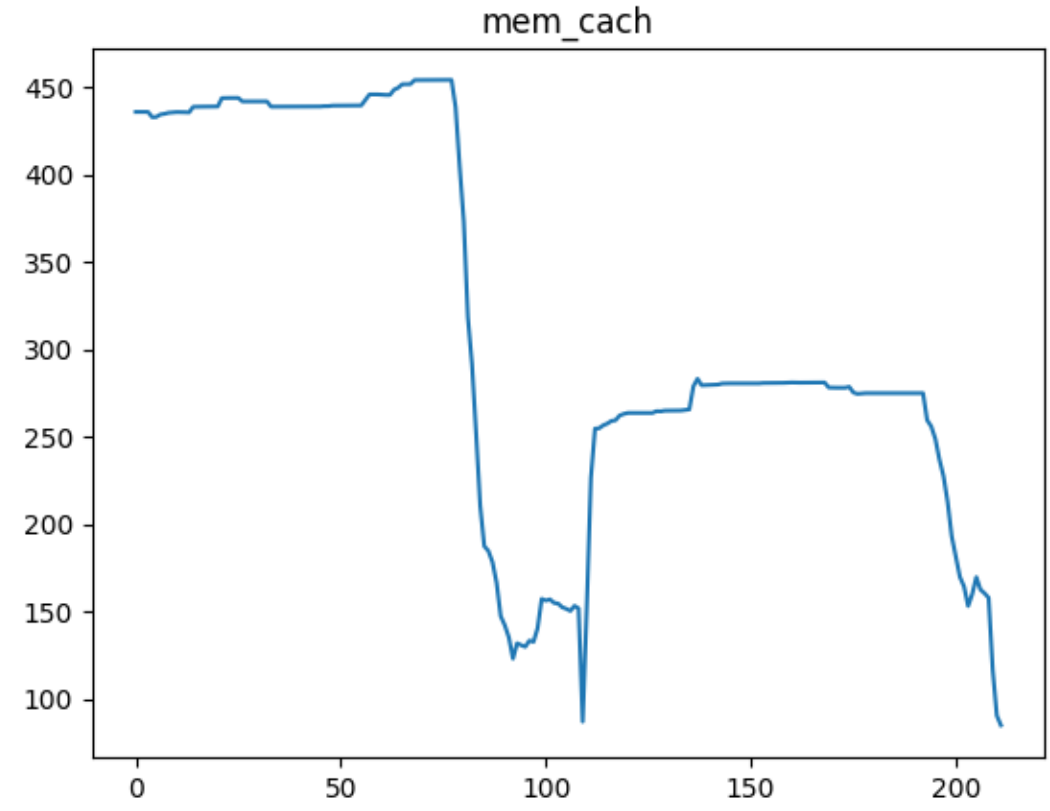

swap_avail
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200

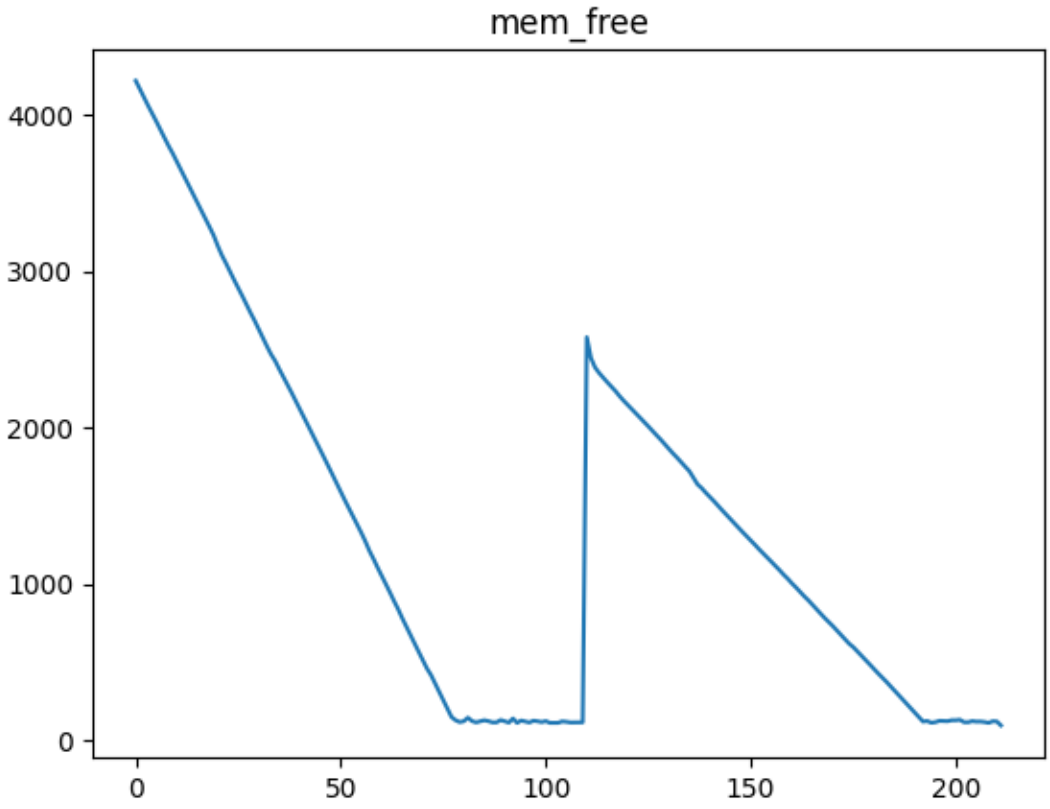

swap_free
1600
1400
1200
1000
800
600
400
200
0
0
50
100
150
200

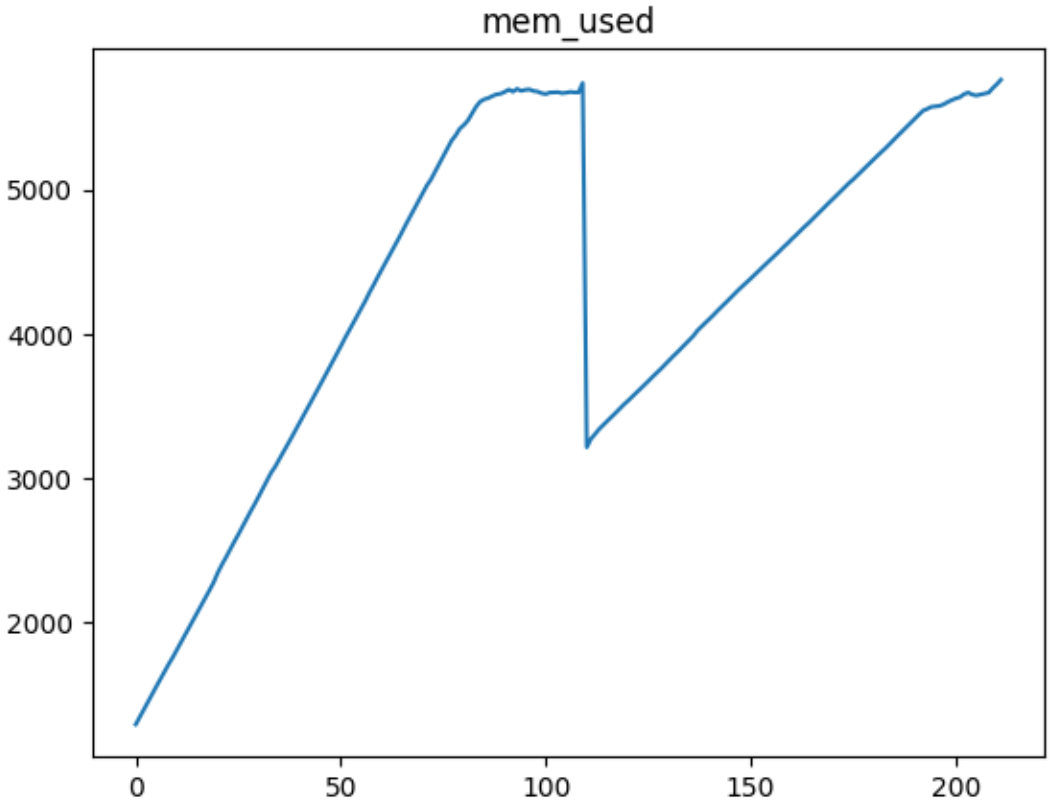


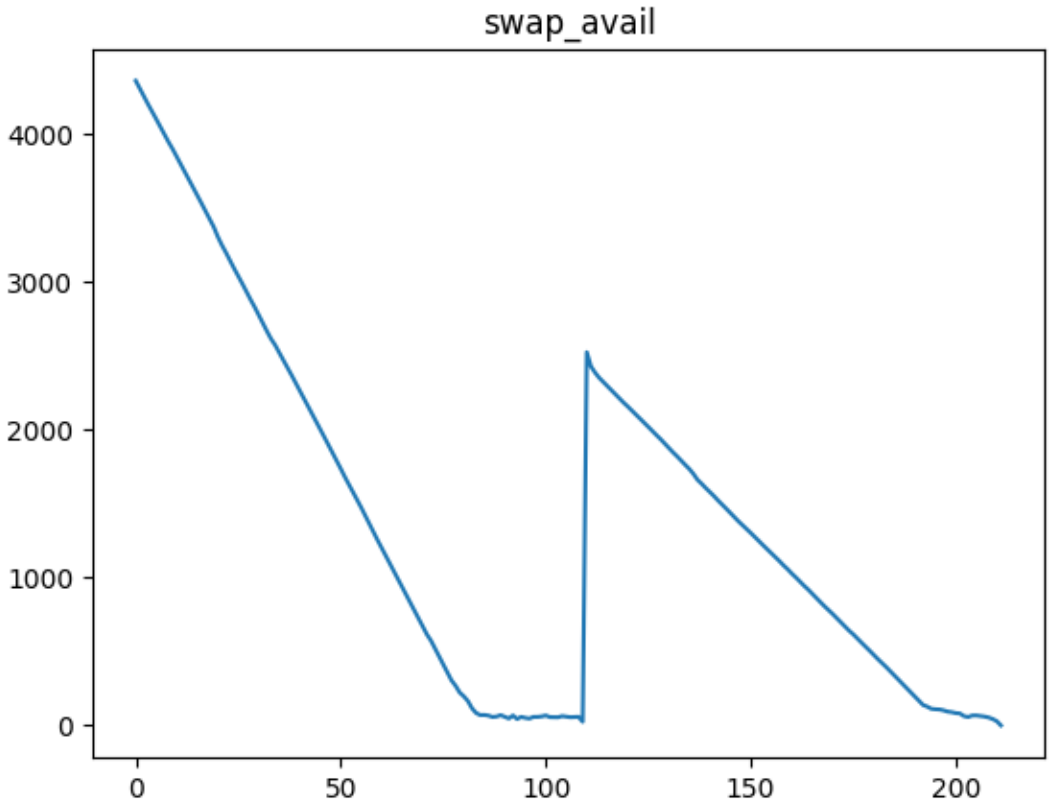
В результате работы скрипта OOM-Killer срабатывает и процесс умирает.

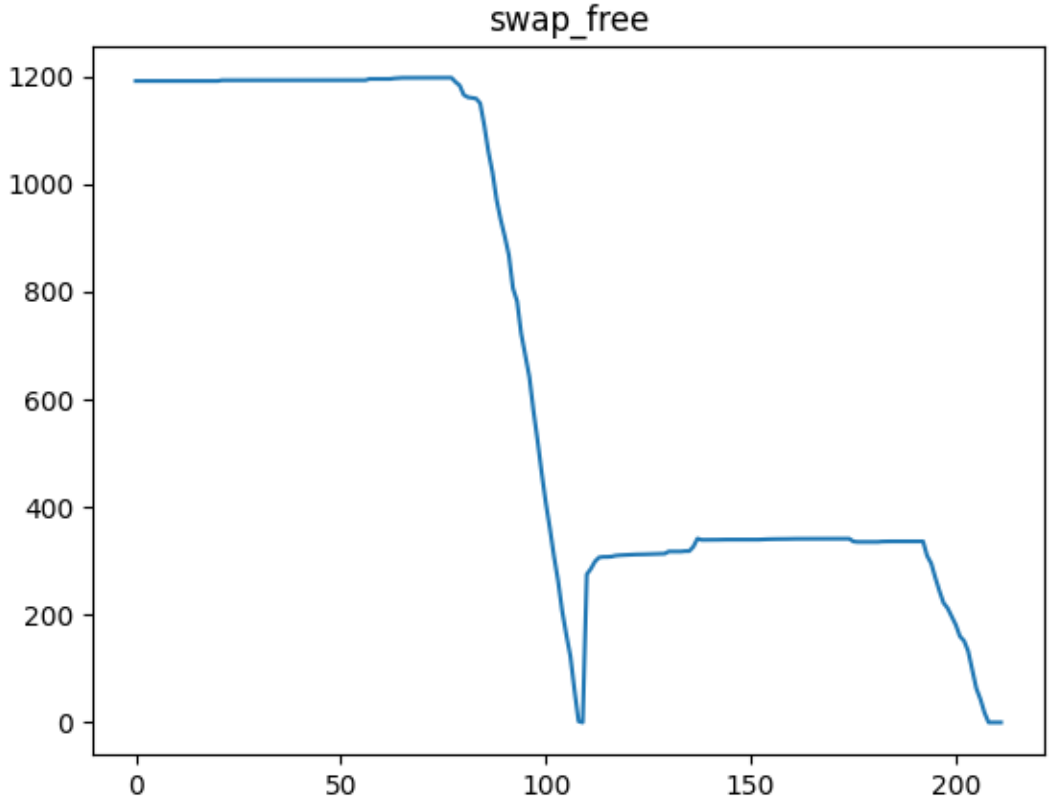
Эксперимент 1 Шаг 2:

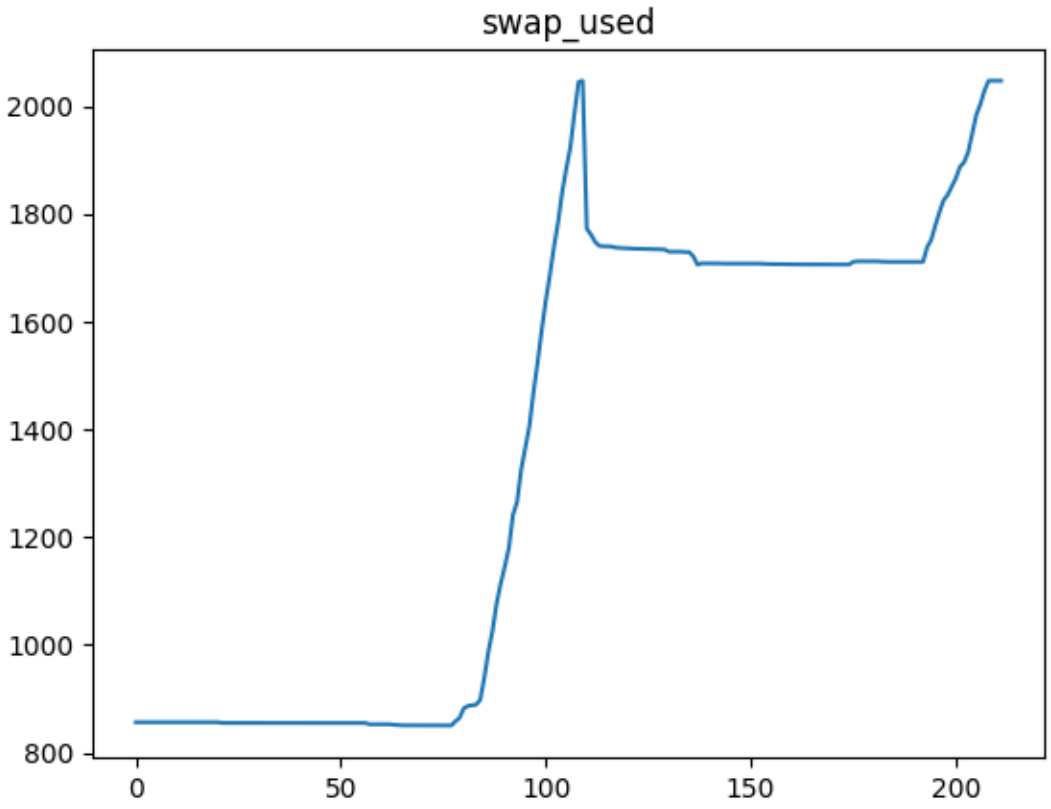



mem_free
4000
3000
2000
1000
0
0
50
100
150
200


mem_used
5000
4000
3000
2000
0
50
100
150
200


swap_avail
4000
3000
2000
1000
0
0
50
100
150
200

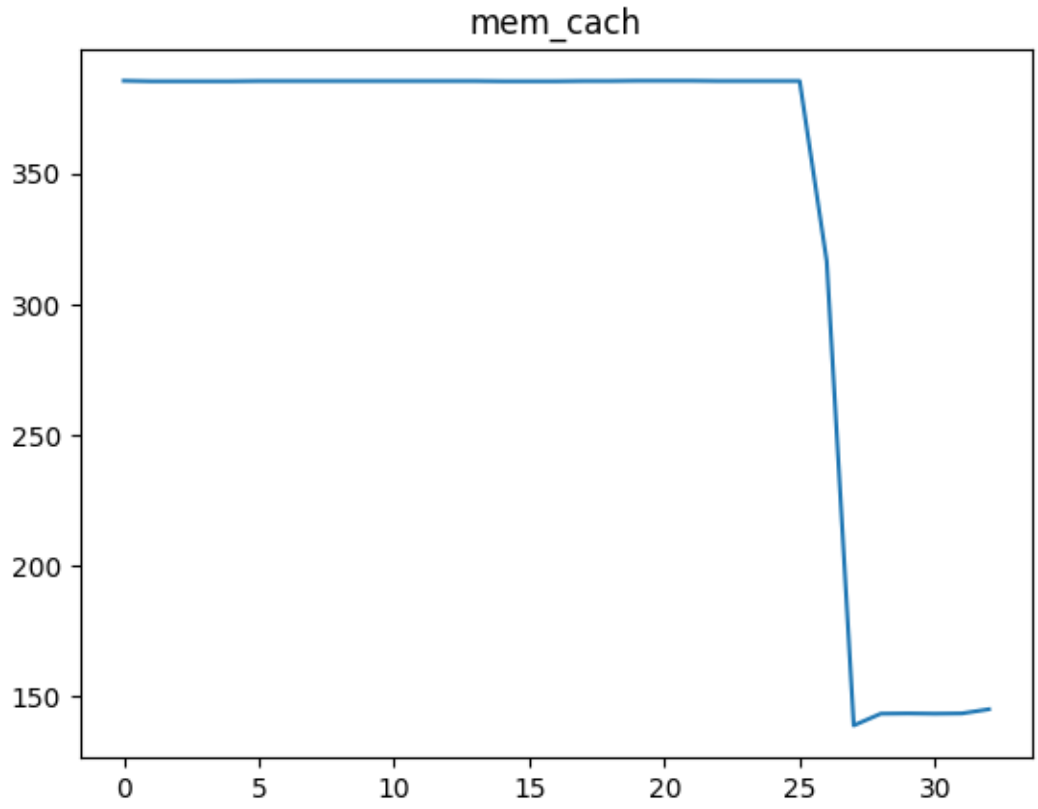

swap_free
1200
1000
800
600
400
200
0
0
50
100
150
200

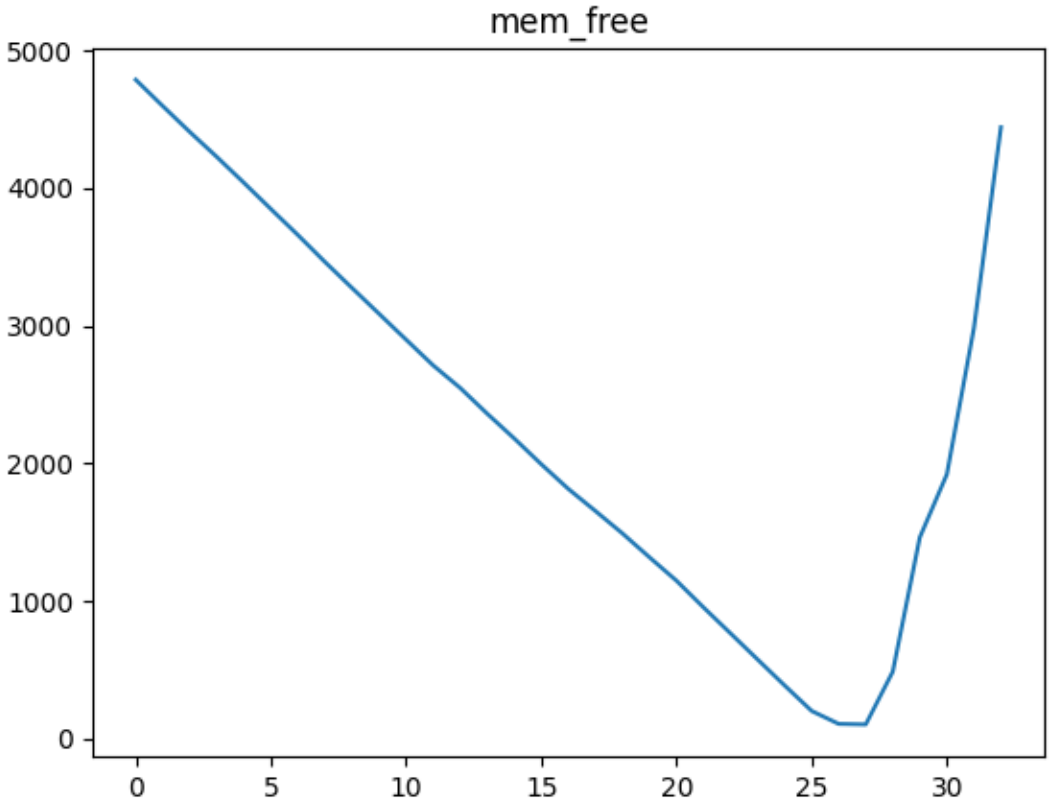


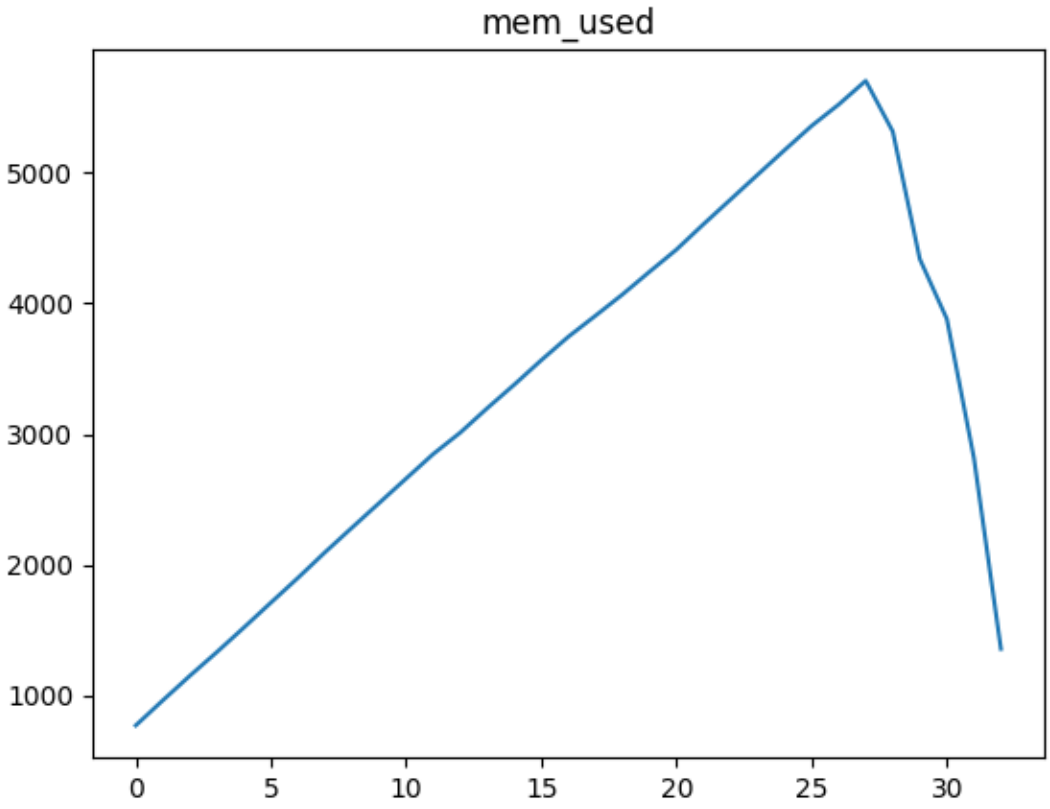
В результате работы скрипта сначала умирает один скрипт (освобождается память), второй скрипт работает еще некоторое кол-во времени и тоже умирает. OOM-Killer работает.

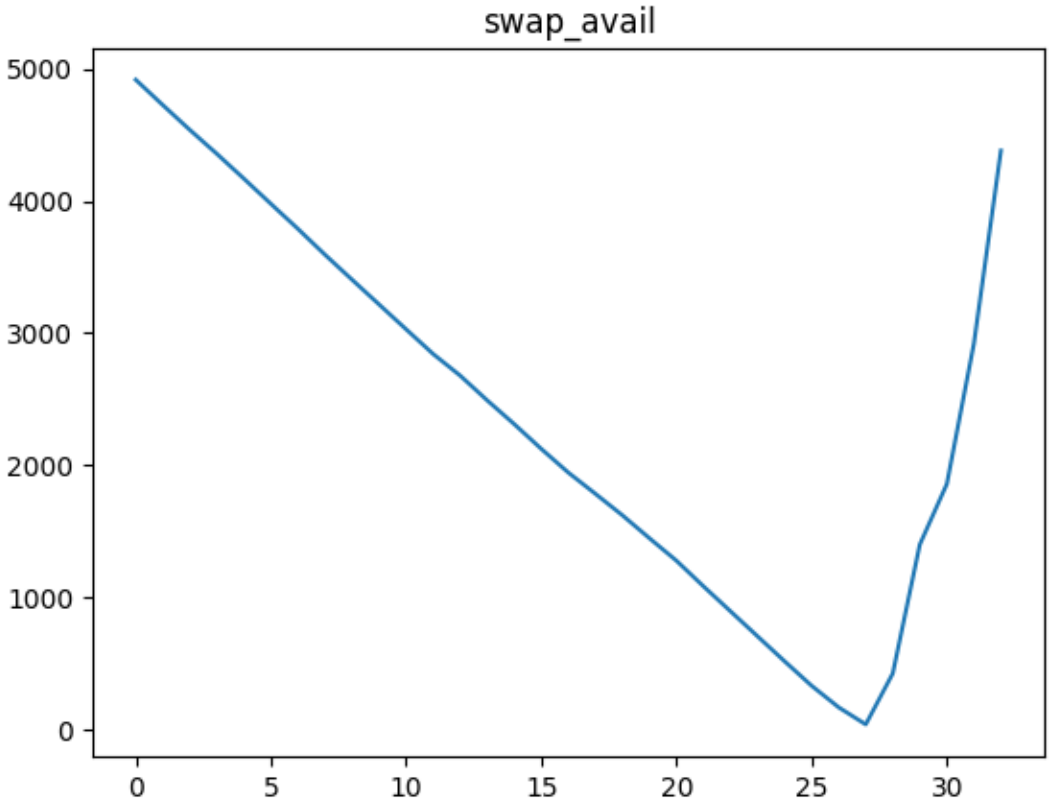
Эксперимент 2 шаг 1:

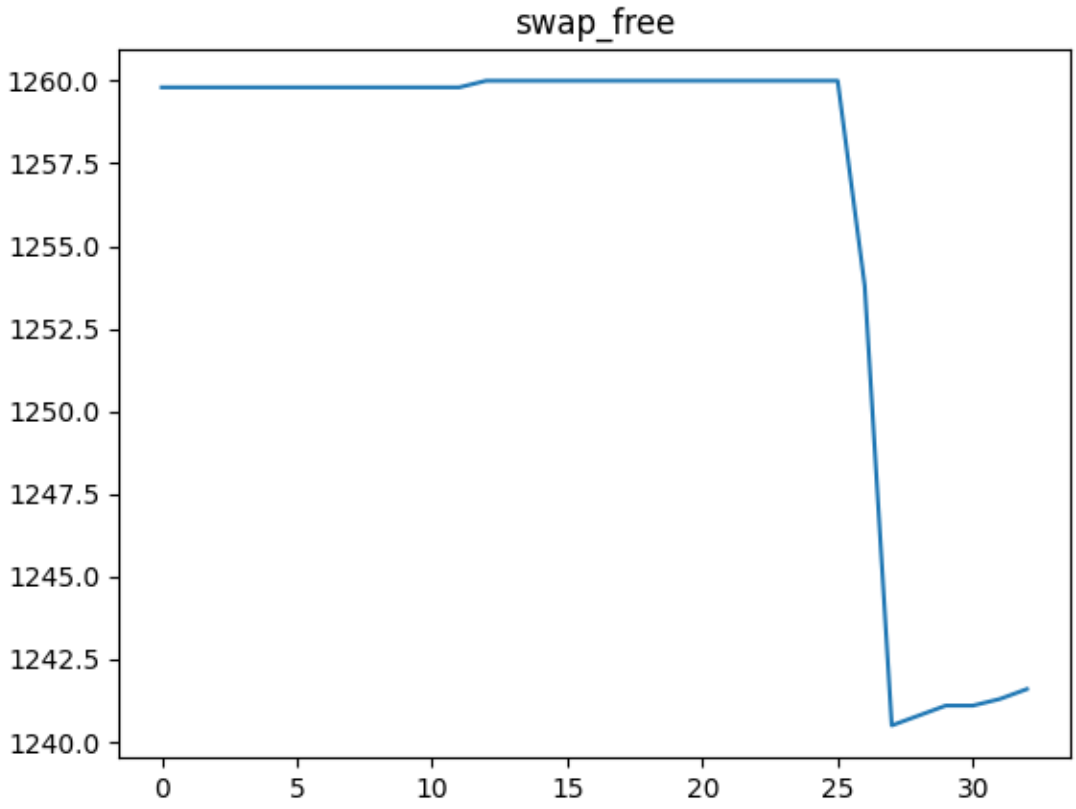
Запускаем 10 скриптов и ставим ограничение на N = 18810000 (Берем максимальное число N из прошлого эксперимента и делим на 10)

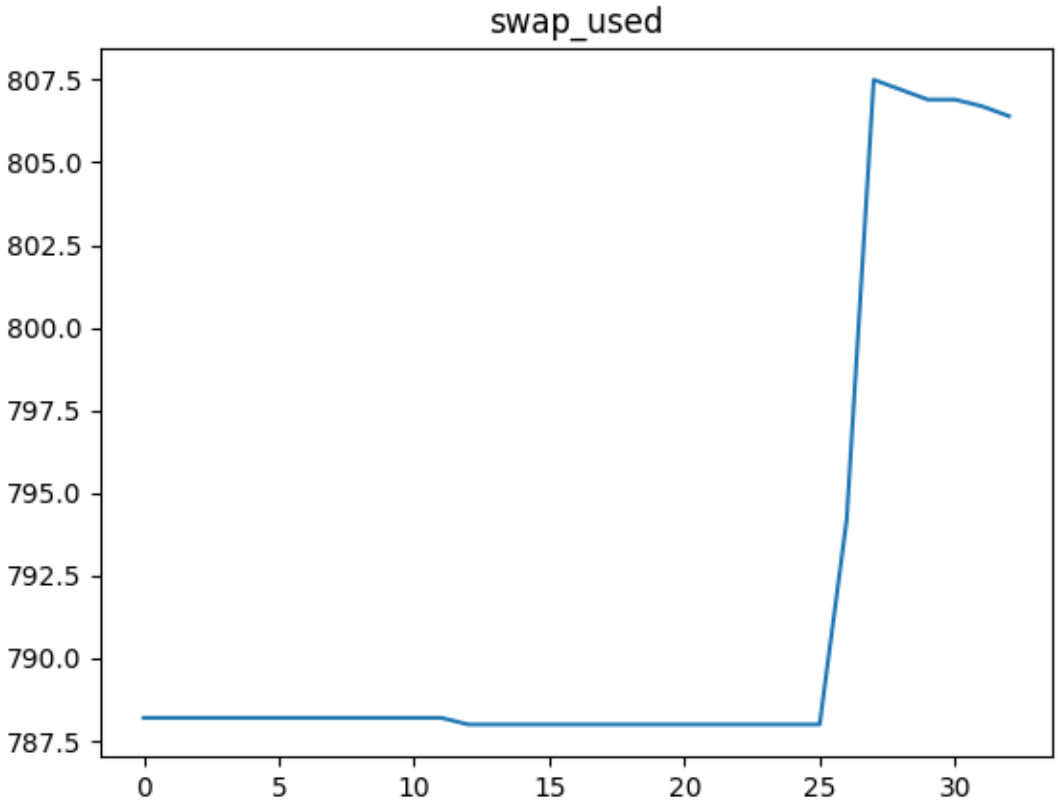

mem_cach
350
300
250
200
150
0
5
10
15
20
25
30


mem_free
5000
4000
3000
2000
1000
0
0
5
10
15
20
25
30


mem_used
5000
4000
3000
2000
1000
0
5
10
15
20
25
30


swap_avail
5000
4000
3000
2000
1000
0
5
10
15
20
25
30

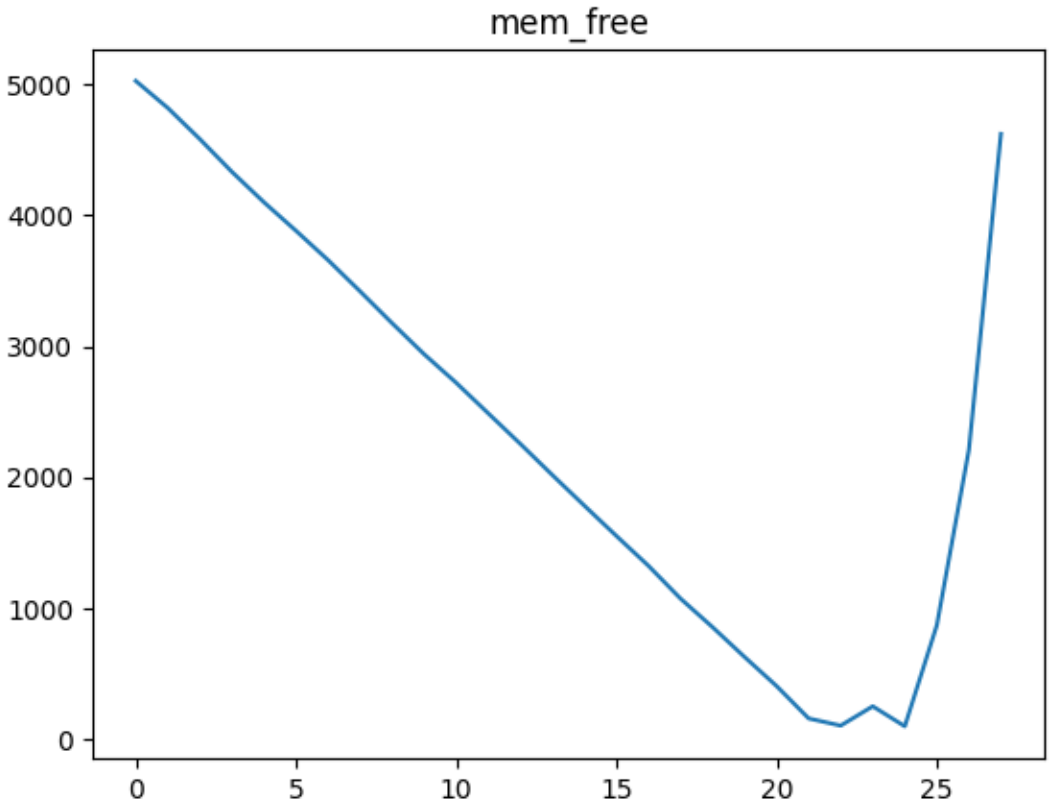


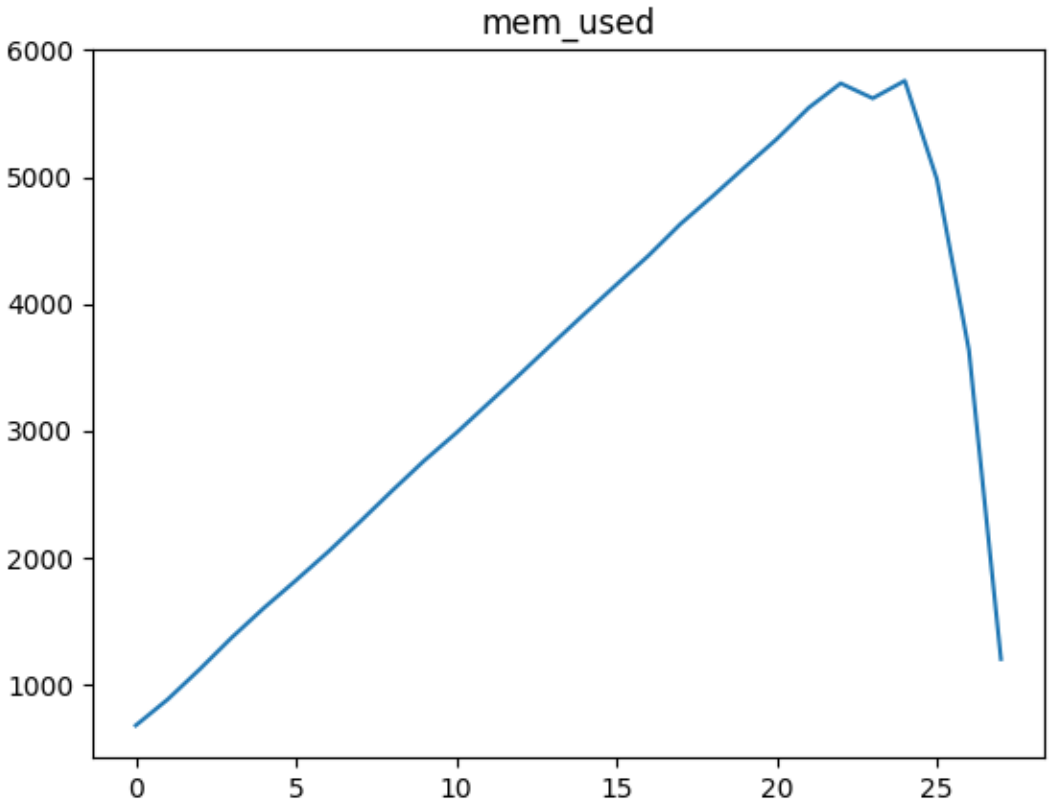


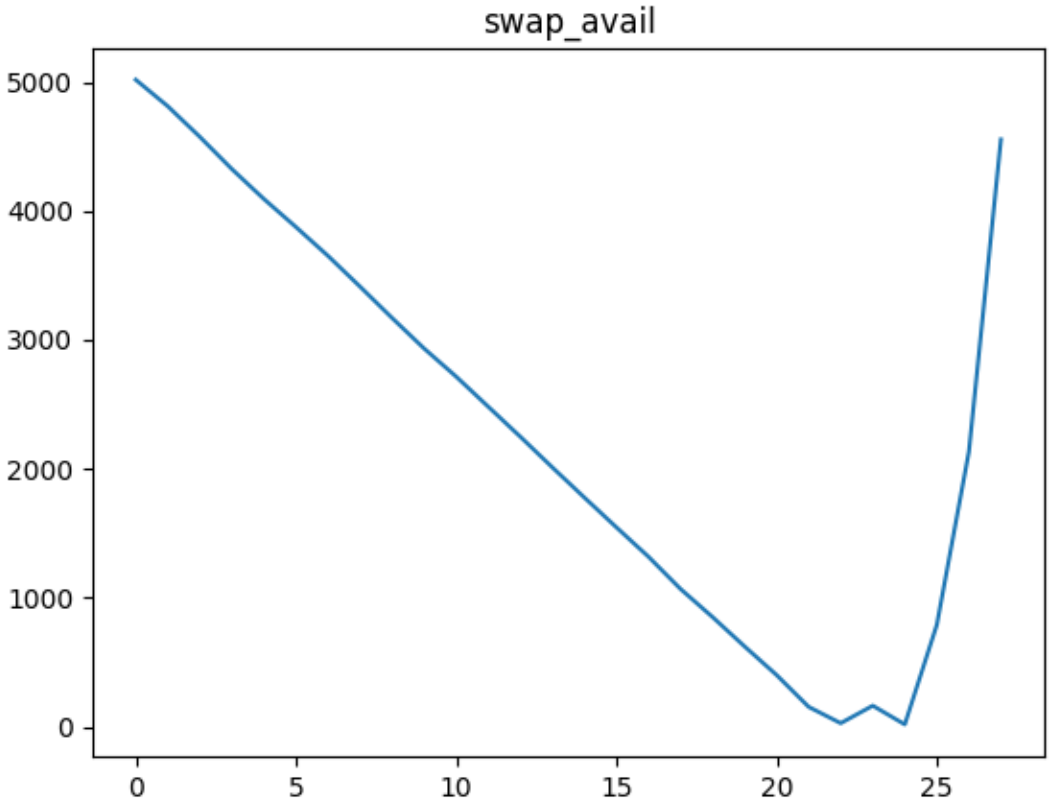
Ни один из скриптов не убился.

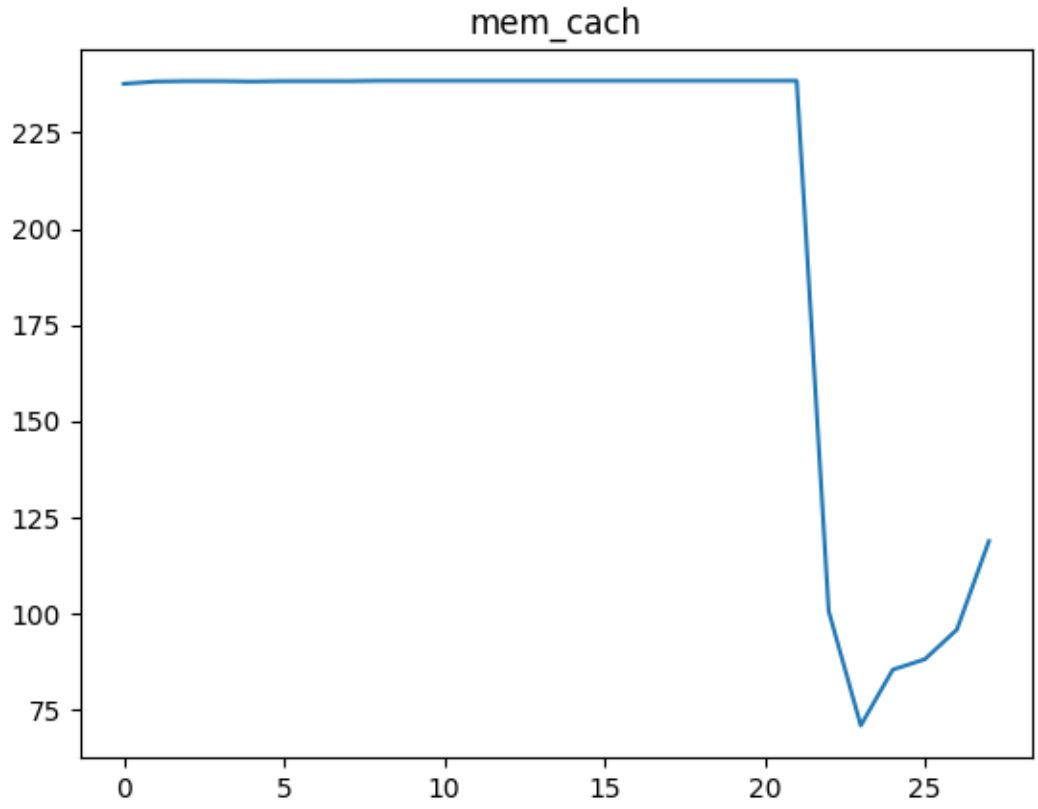
Эксперимент 2 шаг 2:

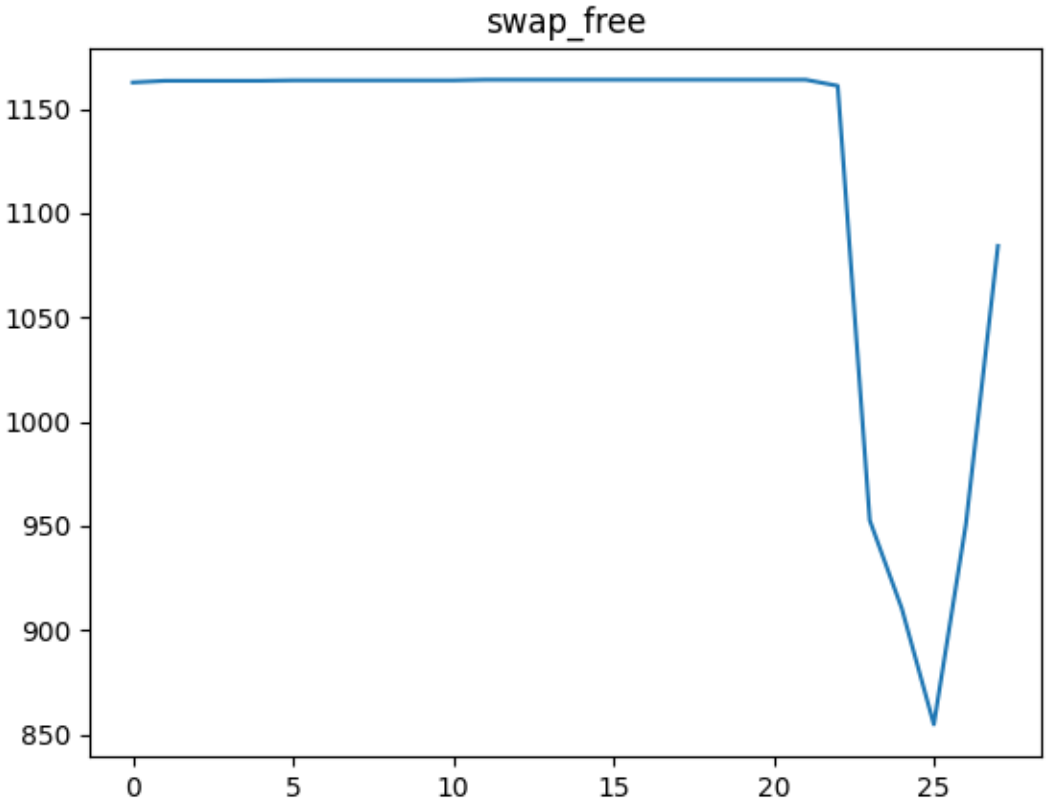
Запускаю 30 скриптов с N из 1 ого шага и вижу что 18/30 скриптов умирает.
Подбираю такое N = 7111330 (~ N / 27) в результате работы все скрипты выжили.

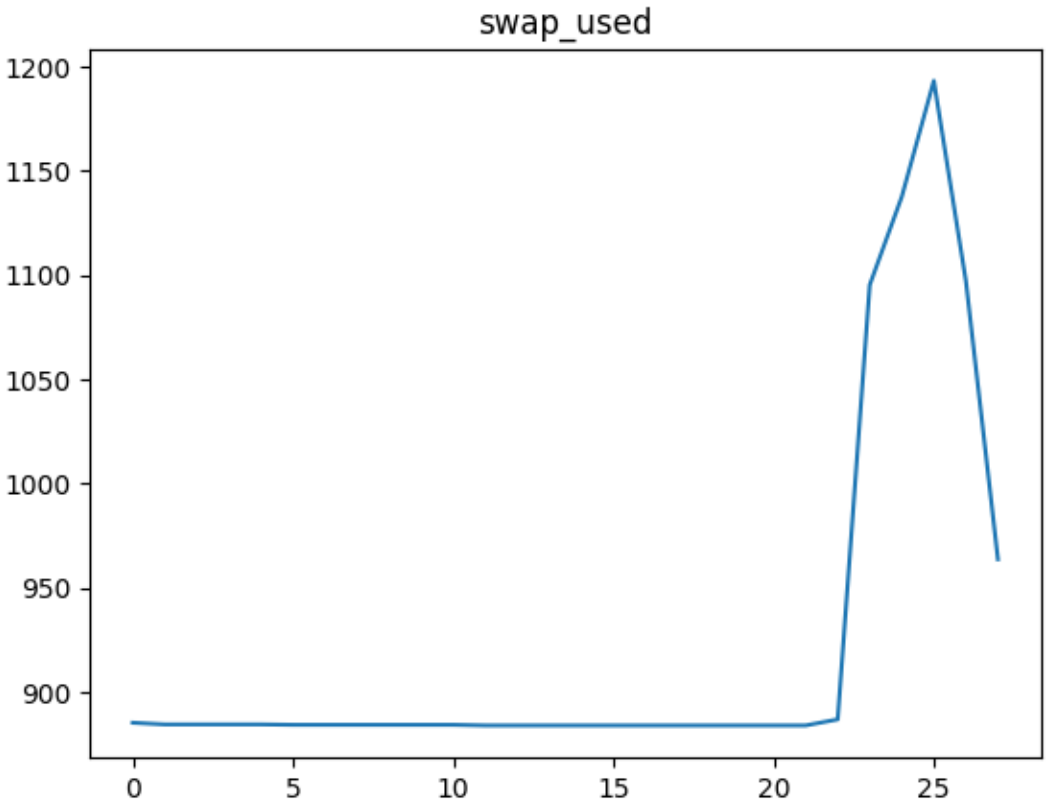



mem_used
6000
5000
4000
3000
2000
1000
0
5
10
15
20
25


swap_avail
5000
4000
3000
2000
1000
0
0
5
10
15
20
25


mem_cach
225
200
175
150
125
100
75
0
5
10
15
20
25


swap_free
1150
1100
1050
1000
950
900
850
0
5
10
15
20
25



Вывод: После запуска данных скриптов на Linux (Mint), убедились в корректности работы OOM-Killer и то, что после того как умирает 1 процесс, память освобождается и распределяется между другими процессами, именно поэтому все 10 и 30 процессов заканчивают свою работу корректно (OMM-Killer не срабатывает).