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»

“Исследование памяти”

Выполнил студент группы №M3211

Папков Алексей Дмитриевич

Проверил:

Дюкарева Вероника Максимовн

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

Эксперимент 1:

Был создан скрипт, который нагружает память. Для этого использовался язык C++, так как в нём есть удовлетворяющие инструменты для такой задачи.

Листинг 1. файл mem.cpp

```
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;

int main()
{
    ofstream output("report.log");

    int count = 0;

    while (true)
    {
      for (int i = 0; i < 100; i++)
            ;
        count++;
        malloc(1);

        if (count % 100000 == 0)
            output << count << endl;
    }

    return 0;
}
```

Как видно из кода программы, каждый 100000 шаг в файл report.log записывается количество выделенных байт.

Так же сделан bash-скрипт, который сохраняет данные вывода комманды top, после чего с помощью python строятся графики сохраненных данных.

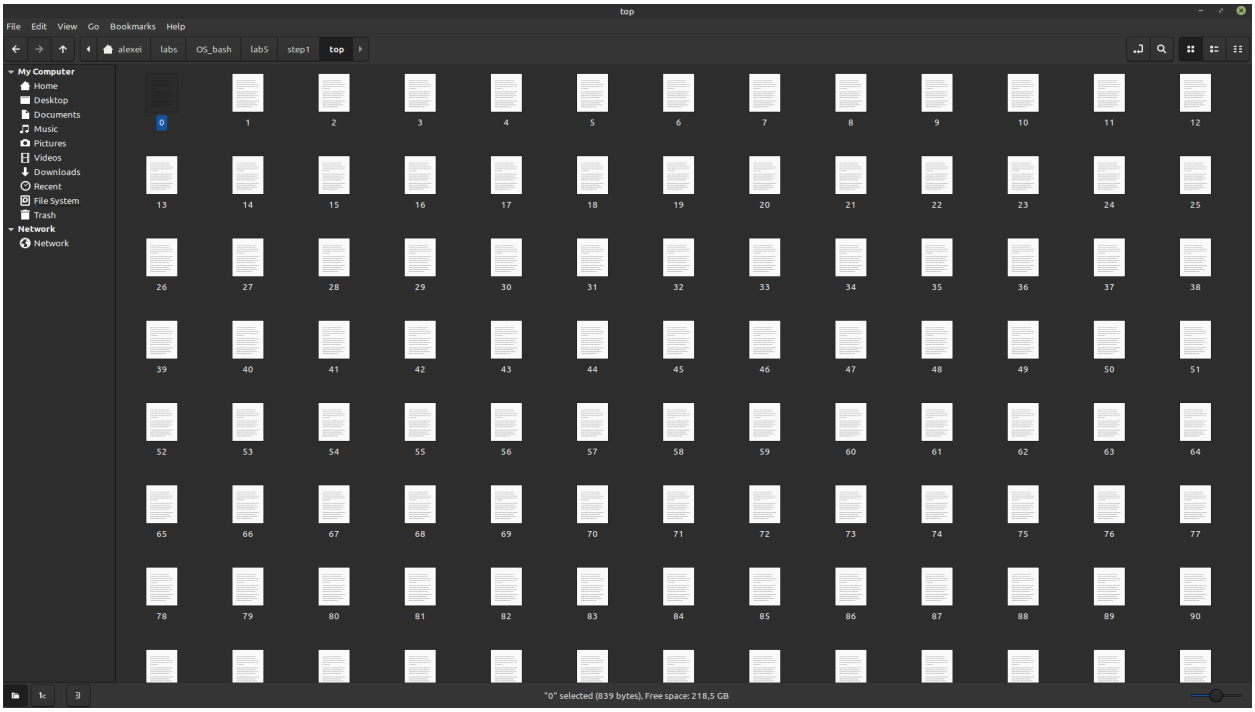
Листинг 2. файл analysis_step1.sh

```
#!/bin/bash

./mem_script &

rm -r $1 2> /dev/null > /dev/null
mkdir $1 2> /dev/null > /dev/null

mkdir $1/top

counter=0
pid=0
while true
do
      check=$(pidof mem_script)

      if [[ $check == "" ]]
      then
            dmesg | grep "mem_script" | grep $pid > $1/dmesg
            cat report.log | tail -1 > $1/count
            break
      else
            pid=$check
      fi

      top -b -n 1 | head -12 > $1/top/"$counter"
      counter=$((counter+1))
done
rm $1/top/$(ls $1/top | sort -n | tail -1)
for i in $(ls $1/top | sort -n)
do
      info_header_mem=$(cat $1/top/$i | head -5 | tail -2 | head -1)
      info_header_swap=$(cat $1/top/$i | head -5 | tail -2| tail -1)

      echo $(echo $info_header_mem | awk '{print $6}' | tr ',' '.') >>
$1/mem_free
      echo $(echo $info_header_mem | awk '{print $8}' | tr ',' '.') >>
$1/mem_used
      echo $(echo $info_header_mem | awk '{print $10}' | tr ',' '.') >>
$1/mem_cach

      echo $(echo $info_header_swap | awk '{print $5}' | tr ',' '.') >>
$1/swap_free
      echo $(echo $info_header_swap | awk '{print $7}' | tr ',' '.') >>
$1/swap_used
      echo $(echo $info_header_swap | awk '{print $9}' | tr ',' '.') >>
$1/swap_avail
done

echo $PWD"/"$1 | python3 graph1.py > /dev/null 2> /dev/null
```

Листинг 3. файл graph1.py

```
import os
import matplotlib.pyplot as plt
import numpy as np

def save(save_path, name='', fmt='png'):
    plt.savefig(save_path + '/{}.{}'.format(name, fmt), fmt='png')
def create_graph(_save_path, path, name1):
    f = open(path, 'r')
    text = f.read()
    splited = text.split("\n")
    filter = []
    for s in splited:
        if s != "":
            filter.append(s)

    yArray = np.array(list(map(float, filter)))
    xArray = np.array([x for x in range(0,len(yArray))])
    f.close()
    plt.title(name1)
    plt.plot(xArray, yArray)

    save(_save_path, name=name1, fmt='png')
    plt.figure().clear()
path = input()
create_graph(path, path + "/mem_cach", "mem_cach")
create_graph(path, path + "/mem_free", "mem_free")
create_graph(path, path + "/mem_used", "mem_used")

create_graph(path, path + "/swap_avail", "swap_avail")
create_graph(path, path + "/swap_free", "swap_free")
create_graph(path, path + "/swap_used", "swap_used")
```

analysis_step1.sh сохраняет вывод top до тех пор, пока mem_script аварийно не остановят.



После остановки скрипта в файл dmesg записывается вывод из системного журнала про аварийную остановку, так же сохраняется последнее значения из файла report.log, после запускается python для отображения графиков изменения величин.

Рассмотрим вывод скрипта:

count:

197400000

dmesg:

[5536.464578] [19192] 1000 19192 1544252 1300967 12406784   241844   0 mem_script

[5536.464586] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=/,mems_allowed=0,global_oom,task_memcg=/user.slice/user-1000.slice/user@1000.service,task=mem_script,pid=19192,uid=1000

[5536.464603] Out of memory: Killed process 19192 (mem_script) total-vm:6177008kB, anon-rss:5203864kB, file-rss:4kB, shmem-rss:0kB, UID:1000 pgtables:12116kB oom_score_adj:0

[5536.698747] oom_reaper: reaped process 19192 (mem_script), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB

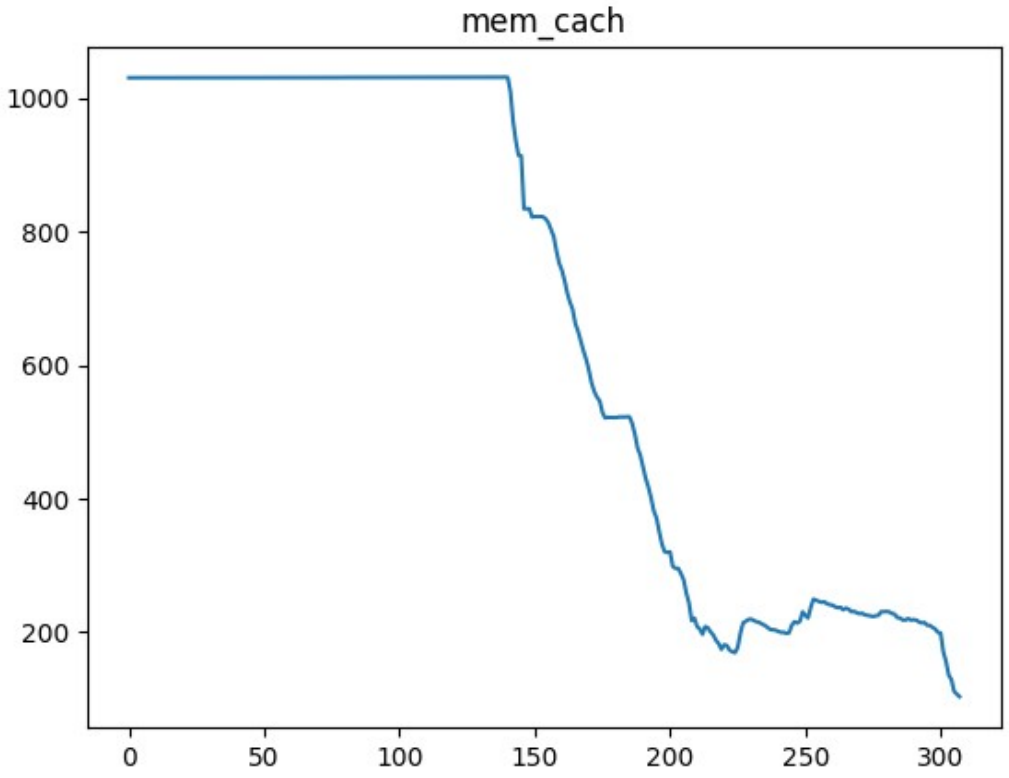


Рисунок 1. График величины buff/cache, 1-й этап эксперимента 1

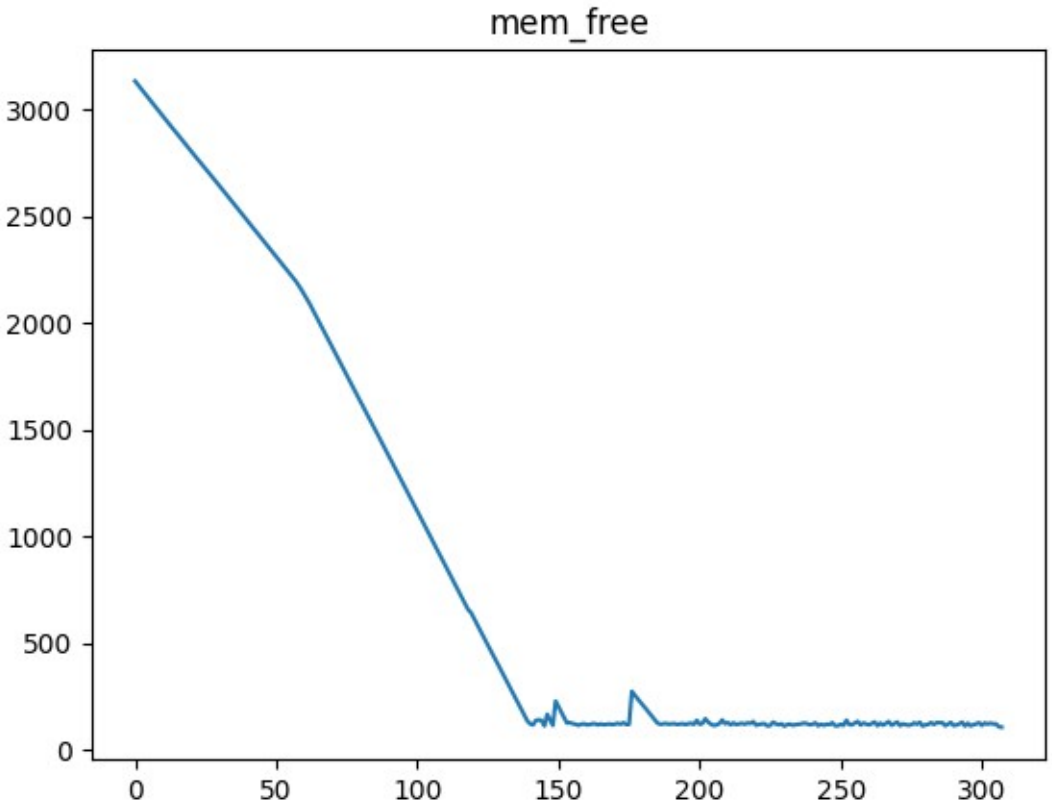


Рисунок 2. График величины mem free, 1-й этап эксперимента 1

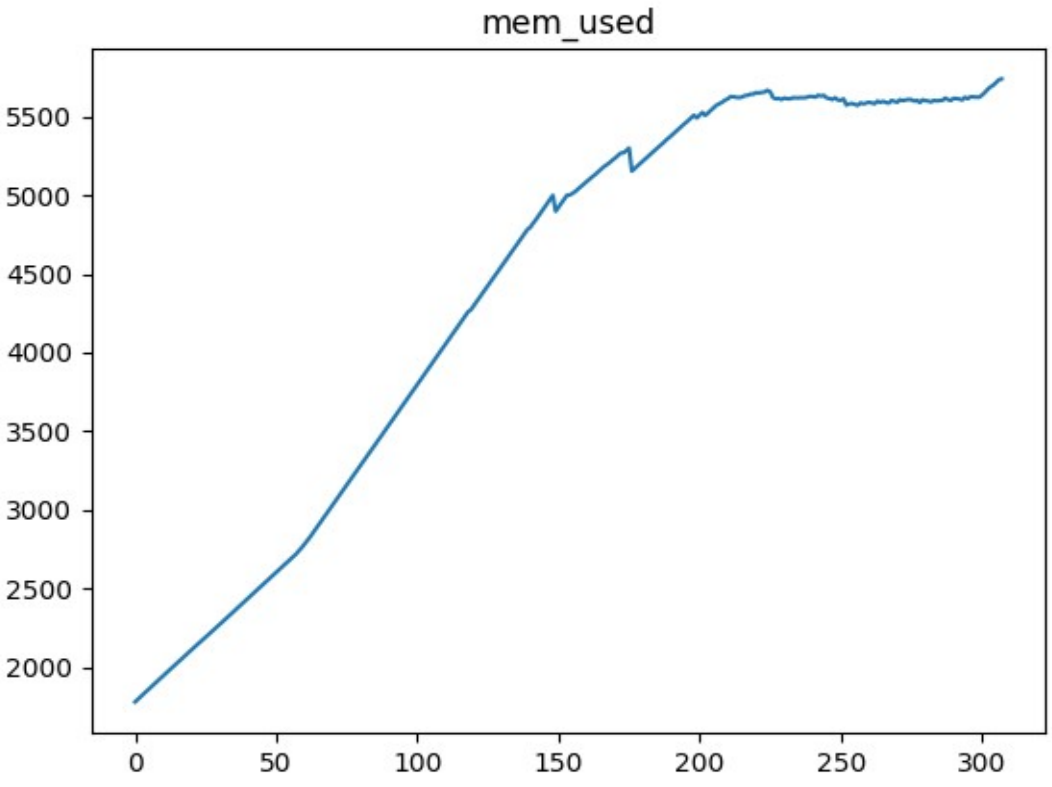


Рисунок 3. График величины mem used, 1-й этап эксперимента 1

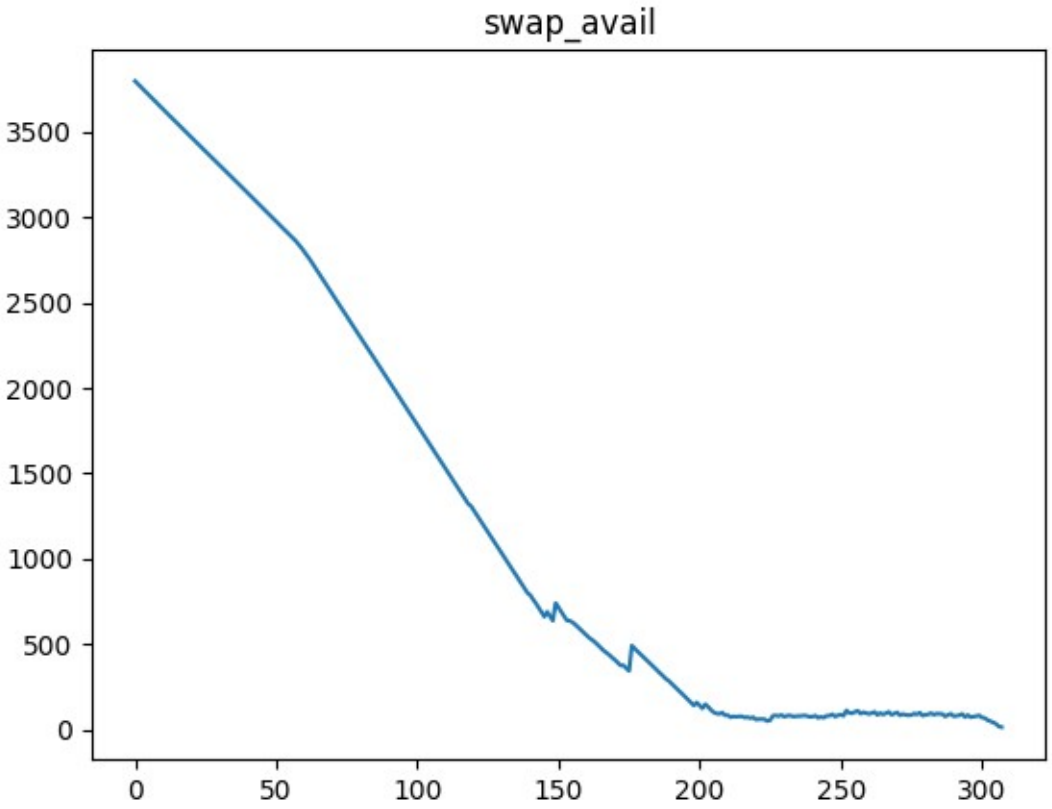


Рисунок 4. График величины avail Mem, 1-й этап эксперимента 1

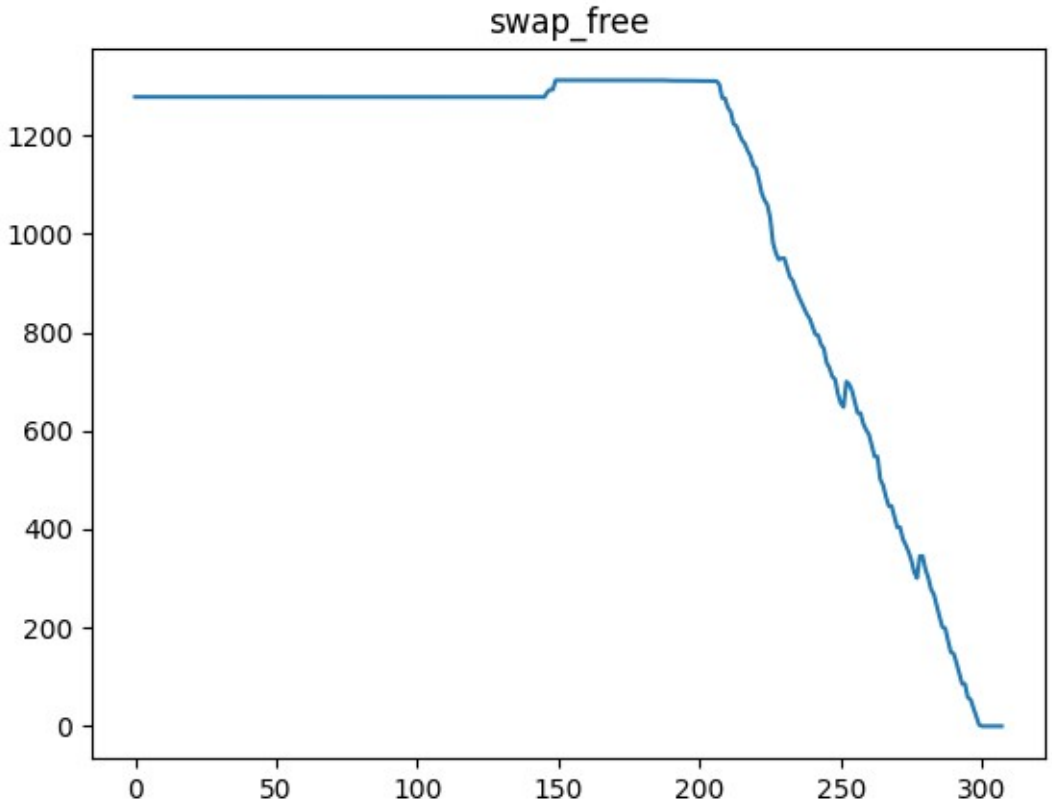


Рисунок 5. График величины swap free, 1-й этап эксперимента 1

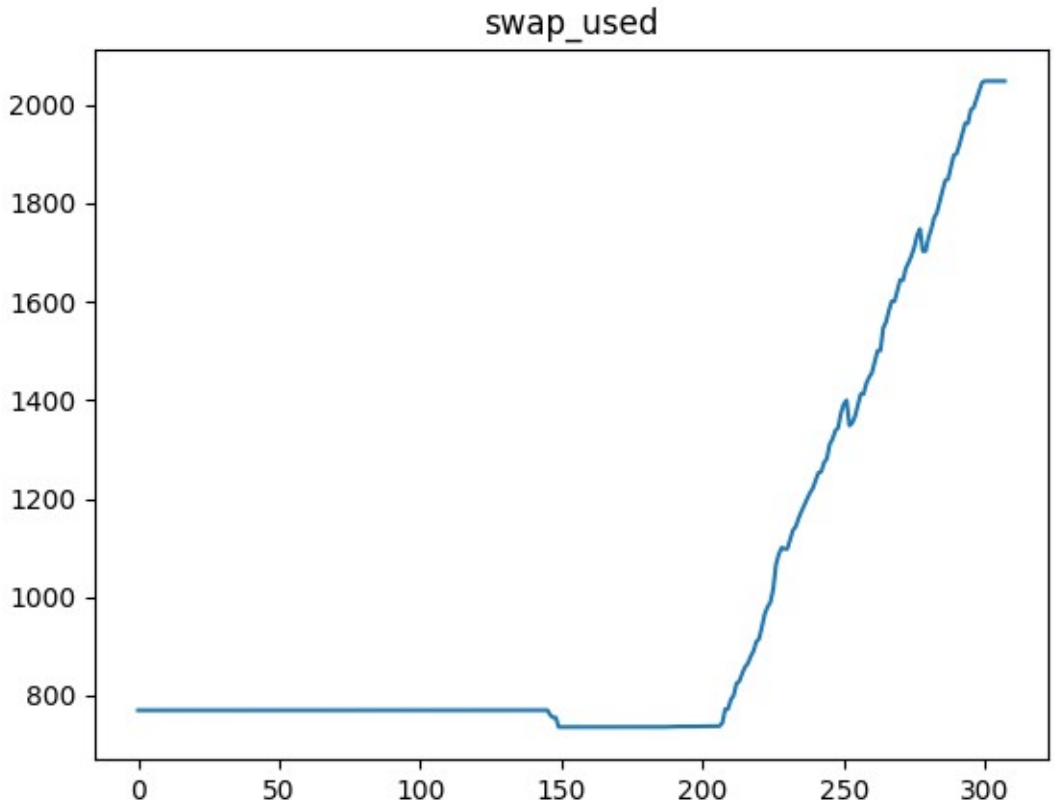


Рисунок 6. График величины swap used, 1-й этап эксперимента 1

Переходим к следующему этапу: одновременно будем запускать два экземпляра mem.cpp, но они будут записывать свои значения в разные файлы.

Листинг 4. файл analysis_step2.sh

```
#!/bin/bash

./mem_script &
./mem_script1 &

rm -r $1 2> /dev/null > /dev/null
mkdir $1 2> /dev/null > /dev/null

mkdir $1/top

counter=0
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 -b -n 1 | head -12 > $1/top/"$counter"
        counter=$((counter+1))
done
rm $1/top/$(ls $1/top | sort -n | tail -1)
for i in $(ls $1/top | sort -n)
do
        info_header_mem=$(cat $1/top/$i | head -5 | tail -2 | head -1)
        info_header_swap=$(cat $1/top/$i | head -5 | tail -2| tail -1)

        echo $(echo $info_header_mem | awk '{print $6}' | tr ',' '.') >> $1/mem_free
        echo $(echo $info_header_mem | awk '{print $8}' | tr ',' '.') >> $1/mem_used
        echo $(echo $info_header_mem | awk '{print $10}' | tr ',' '.') >> $1/mem_cach

        echo $(echo $info_header_swap | awk '{print $5}' | tr ',' '.') >> $1/swap_free
        echo $(echo $info_header_swap | awk '{print $7}' | tr ',' '.') >> $1/swap_used
        echo $(echo $info_header_swap | awk '{print $9}' | tr ',' '.') >> $1/swap_avail
done

echo $PWD"/"$1 | python3 graph1.py > /dev/null 2> /dev/null
```

Скрипт будет дальше сохранят вывод top, пока оба скрипта не будут аварийно остановлены

Рассмотрим вывод скрипта:

count:

99100000

197700000

dmesg:

[6408.317780] [31192] 1000 31192   776375   661789 6262784   113153 0 mem_script

[6408.317800] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=/,mems_allowed=0,global_oom,task_memcg=/user.slice/user-1000.slice/user@1000.service,task=mem_script,pid=31192,uid=1000

[6408.317815] Out of memory: Killed process 31192 (mem_script) total-vm:3105500kB, anon-rss:2647152kB, file-rss:4kB, shmem-rss:0kB, UID:1000 pgtables:6116kB oom_score_adj:0

[6408.485136] oom_reaper: reaped process 31192 (mem_script), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB

[6408.317428] CPU: 2 PID: 31193 Comm: mem_script1 Not tainted 5.4.0-54-generic #60-Ubuntu

[6408.317782] [31193] 1000 31193 768851 654123 6201344 113280 0 mem_script1

[6435.479269] [31193] 1000 31193 1546232 1321293 12431360 223473 0 mem_script1

[6435.479273] oom-kill:constraint=CONSTRAINT_NONE,nodemask=(null),cpuset=/,mems_allowed=0,global_oom,task_memcg=/user.slice/user-1000.slice/user@1000.service,task=mem_script1,pid=31193,uid=1000

[6435.479288] Out of memory: Killed process 31193 (mem_script1) total-vm:6184928kB, anon-rss:5285172kB, file-rss:0kB, shmem-rss:0kB, UID:1000 pgtables:12140kB oom_score_adj:0

[6435.696820] oom_reaper: reaped process 31193 (mem_script1), now anon-rss:0kB, file-rss:0kB, shmem-rss:0kB

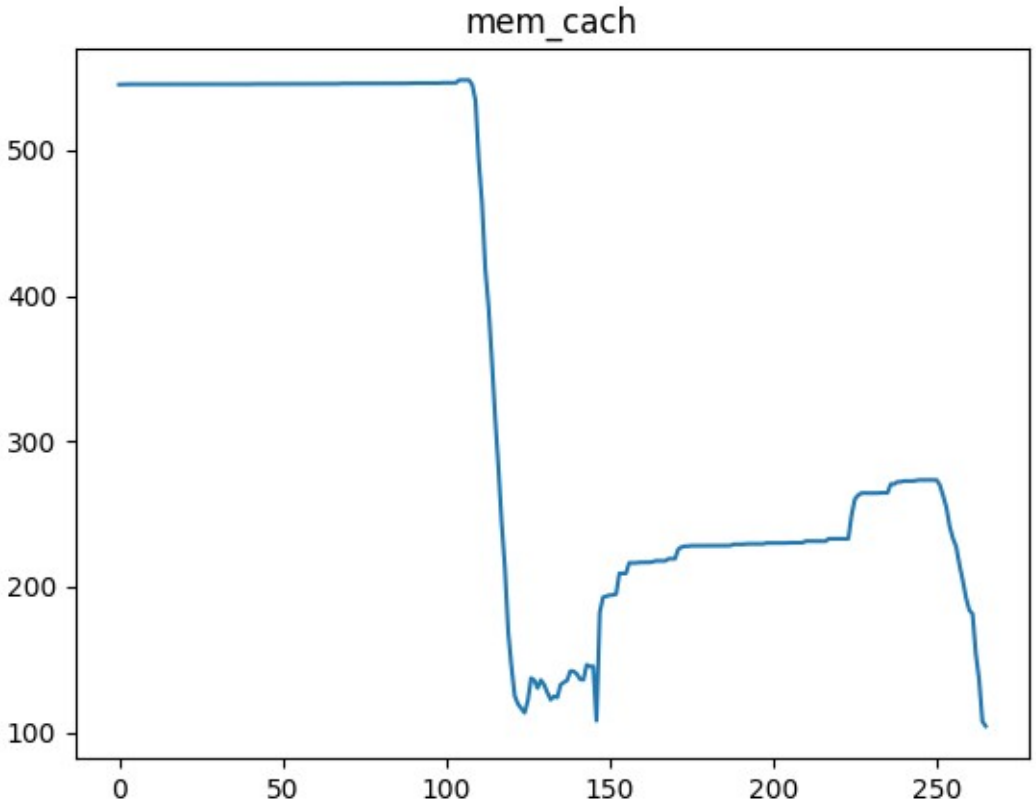


Рисунок 7. График величины buff/cache, 2-й этап эксперимента 1

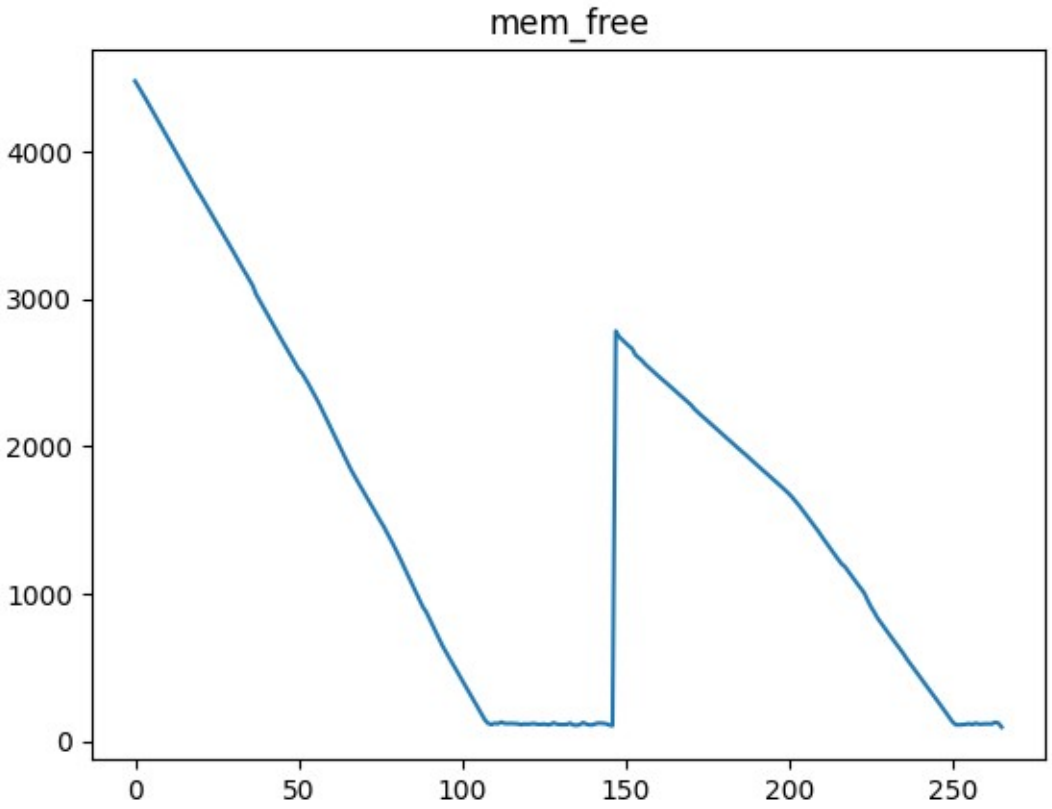


Рисунок 8. График величины mem free, 2-й этап эксперимента 1

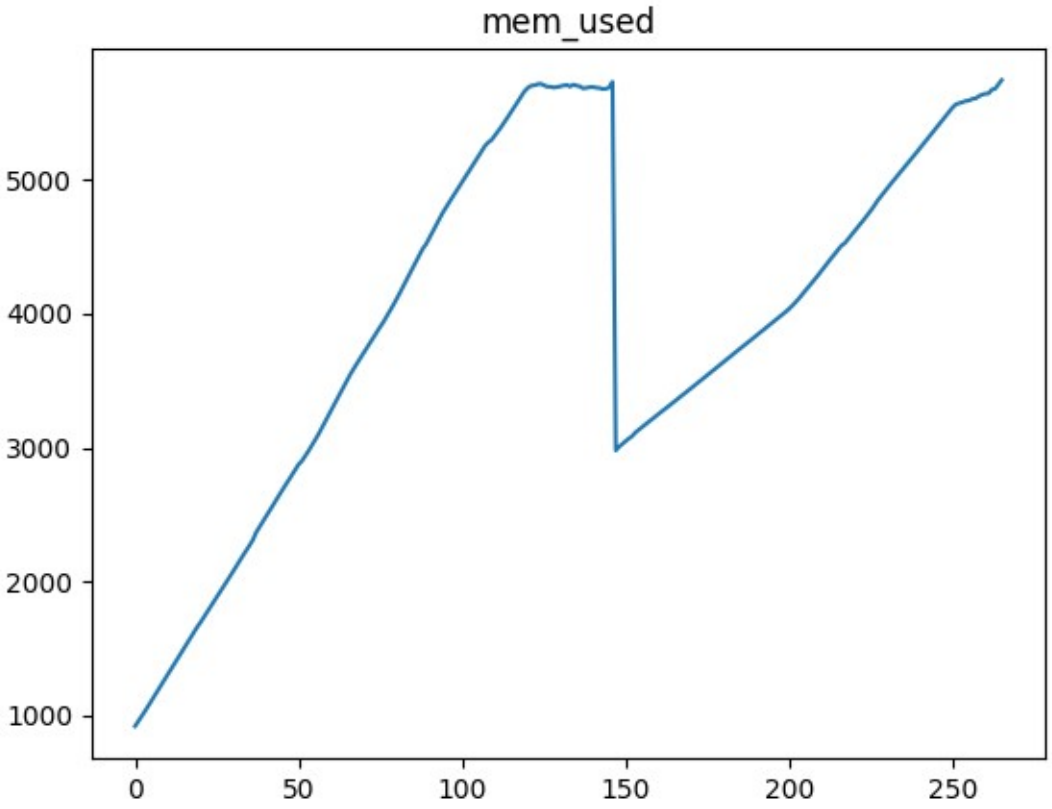


Рисунок 9. График величины mem used, 2-й этап эксперимента 1

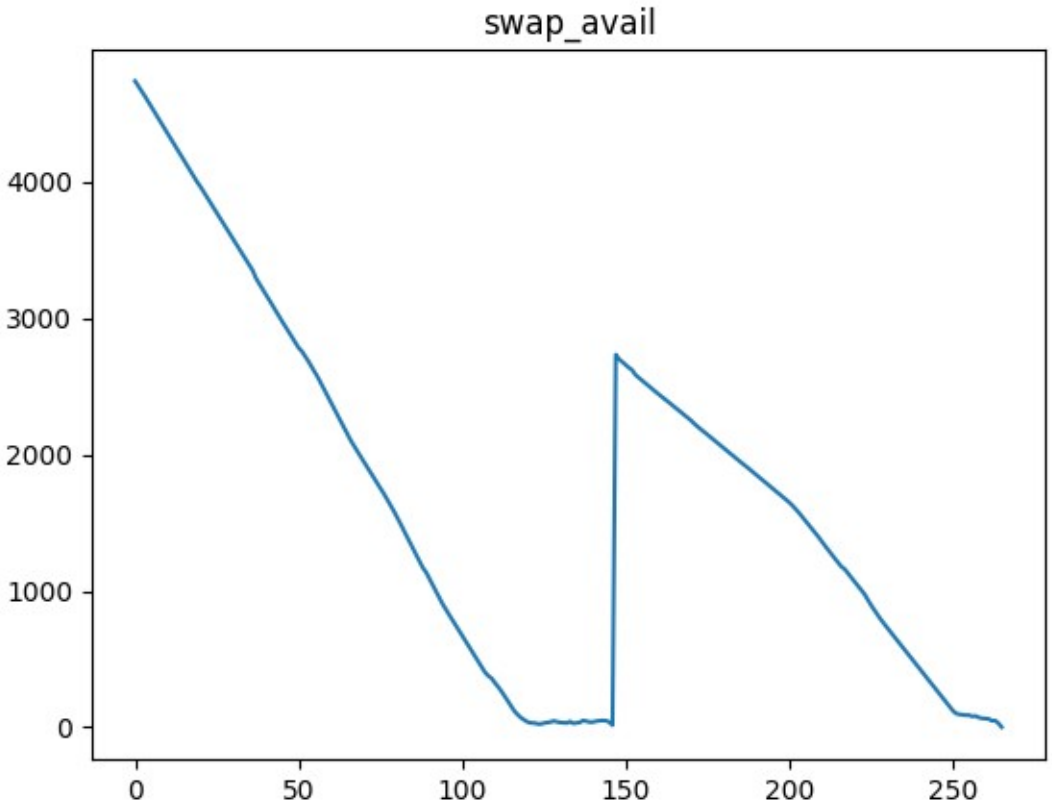


Рисунок 10. График величины avail Mem, 2-й этап эксперимента 1

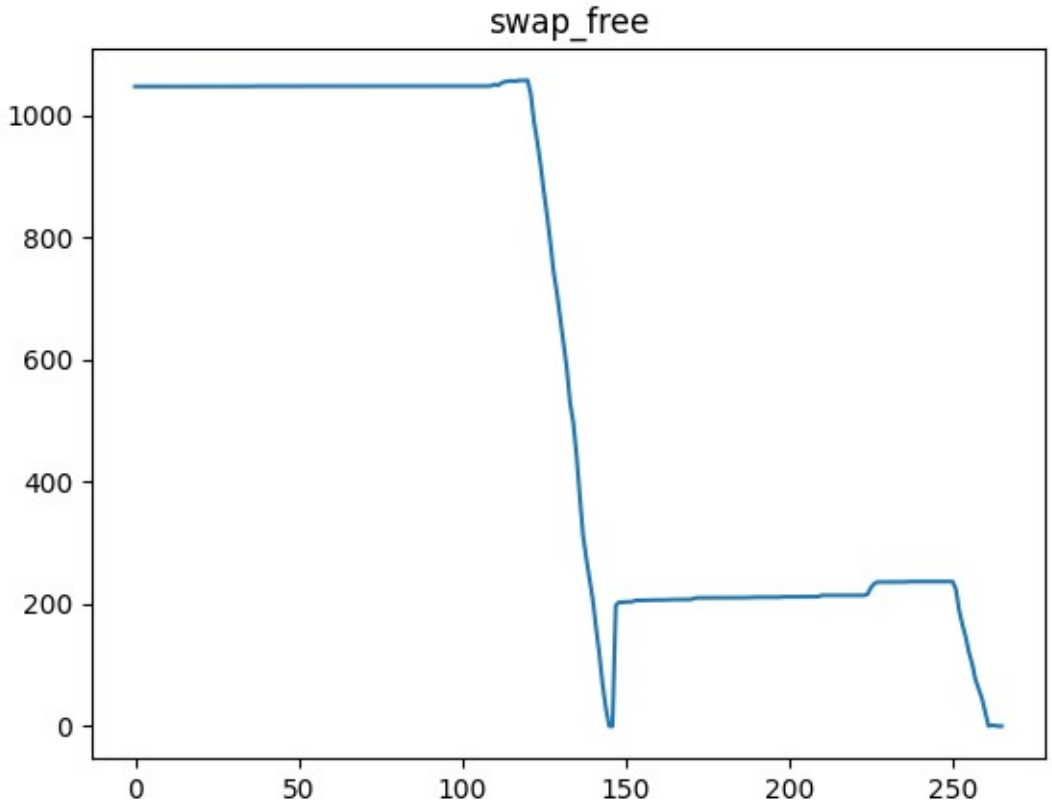


Рисунок 11. График величины swap free, 2-й этап эксперимента 1

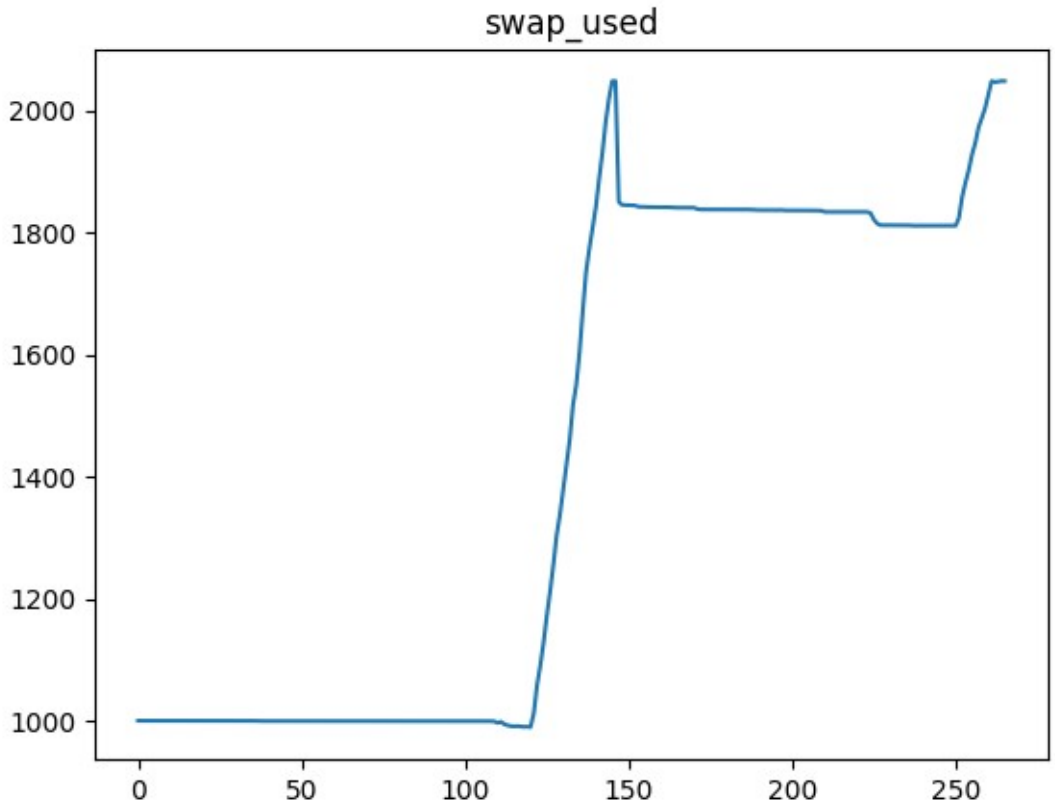


Рисунок 12. График величины swap used, 2-й этап эксперимента 1

Вывод: в данном этапе мы исследовали в каких объемах выделяется память под процессы, можно заметить, что процессы аварийно выключаются, это происходит из-за OOM-killer (Out of Memory killer). Данный инструмент убивает процессы, когда ОС не может выделить новые страницы для процессов. Т.к. OOM-killer считает наш процесс mem_script «плохой», то он его и убивает, это логично, так как наш процесс никак не взаимодействует с памятью, которую на него выделяют. Во втором этапе на графиках видно, что углы выделения памяти, после убийства 1-го процесса уменьшаются, это происходит т. к. до этого сразу 2 процесса просят память, после лишь один, из-за этого в report.log файлах совершенно разные числа. Так же из теоретических знаний можно сказать, что после полного заполнения ОЗУ, ОС начинает выделять память из файла подкачки (что должно привести к уменьшению угла), на графиках выше это сложно заметить, т. к. измерения происходили на SSD, которое в разы быстрее чем HDD.

Эксперимент 2:

Перепишем скрипт, который нагружает память так, чтобы он выделял N байтов, после чего сам заканчивался

Листинг 5. файл newmem.cpp

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    int N;
    cin >> N;
    int count = 0;

    for (int j = 0; j < N; j++)
    {
      for (int i = 0; i < 100; i++)
            ;
        malloc(1);
    }

    return 0;
}
```

Возьмем N из 1-го этапа 1-го эксперимента меньше в 10 раз и запустим 10 таких программ.

Листинг 6. analysis2.sh

```
#!/bin/bash

rm -r $1 2> /dev/null > /dev/null
mkdir $1 2> /dev/null > /dev/null

mkdir $1/top

for (( i=0; i<10; i++ ))
do
      echo "19740000" | ./new_mem_script &
done

counter=0
pid=0
while true
do
      check=$(pidof new_mem_script | wc | awk '{print $2}')

    if [[ $check == "0" ]]
    then
        break
    fi


      top -b -n 1 | head -12 > $1/top/"$counter"
      counter=$((counter+1))
done
```

```
rm $1/top/$(ls $1/top | sort -n | tail -1)
for i in $(ls $1/top | sort -n)
do
      info_header_mem=$(cat $1/top/$i | head -5 | tail -2 | head -1)
      info_header_swap=$(cat $1/top/$i | head -5 | tail -2| tail -1)

      echo $(echo $info_header_mem | awk '{print $6}' | tr ',' '.') >>
$1/mem_free
      echo $(echo $info_header_mem | awk '{print $8}' | tr ',' '.') >>
$1/mem_used
      echo $(echo $info_header_mem | awk '{print $10}' | tr ',' '.') >>
$1/mem_cach

      echo $(echo $info_header_swap | awk '{print $5}' | tr ',' '.') >>
$1/swap_free
      echo $(echo $info_header_swap | awk '{print $7}' | tr ',' '.') >>
$1/swap_used
      echo $(echo $info_header_swap | awk '{print $9}' | tr ',' '.') >>
$1/swap_avail
done

echo $PWD"/"$1 | python3 graph1.py > /dev/null 2> /dev/null
```

Проверив dmesg видно, что ни один из скриптов аварийно не выключился.

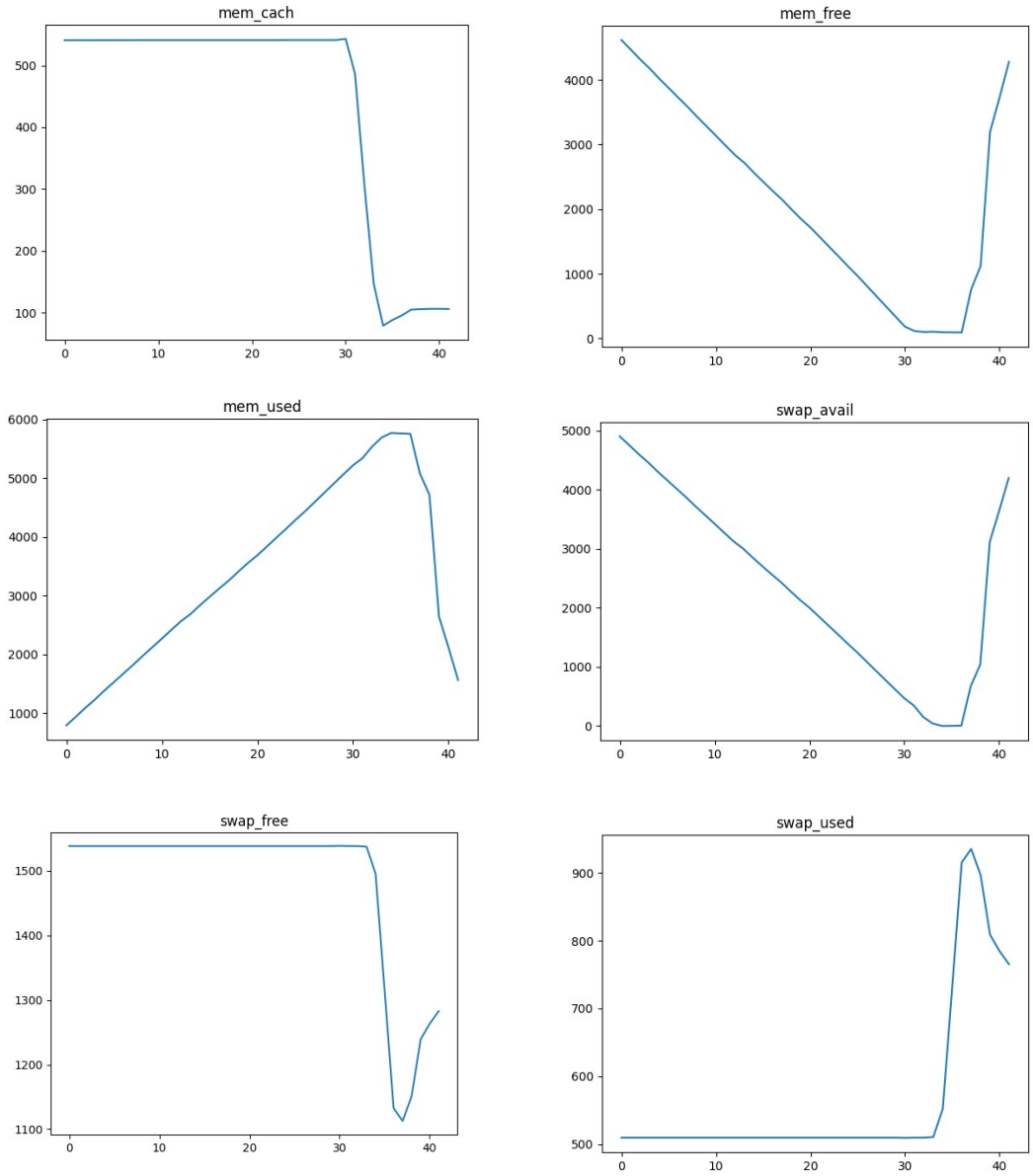


Рисунок 13. демонстрация величин, эксперимент 2

Запустим теперь 30 таких программ и посмотрим на результат

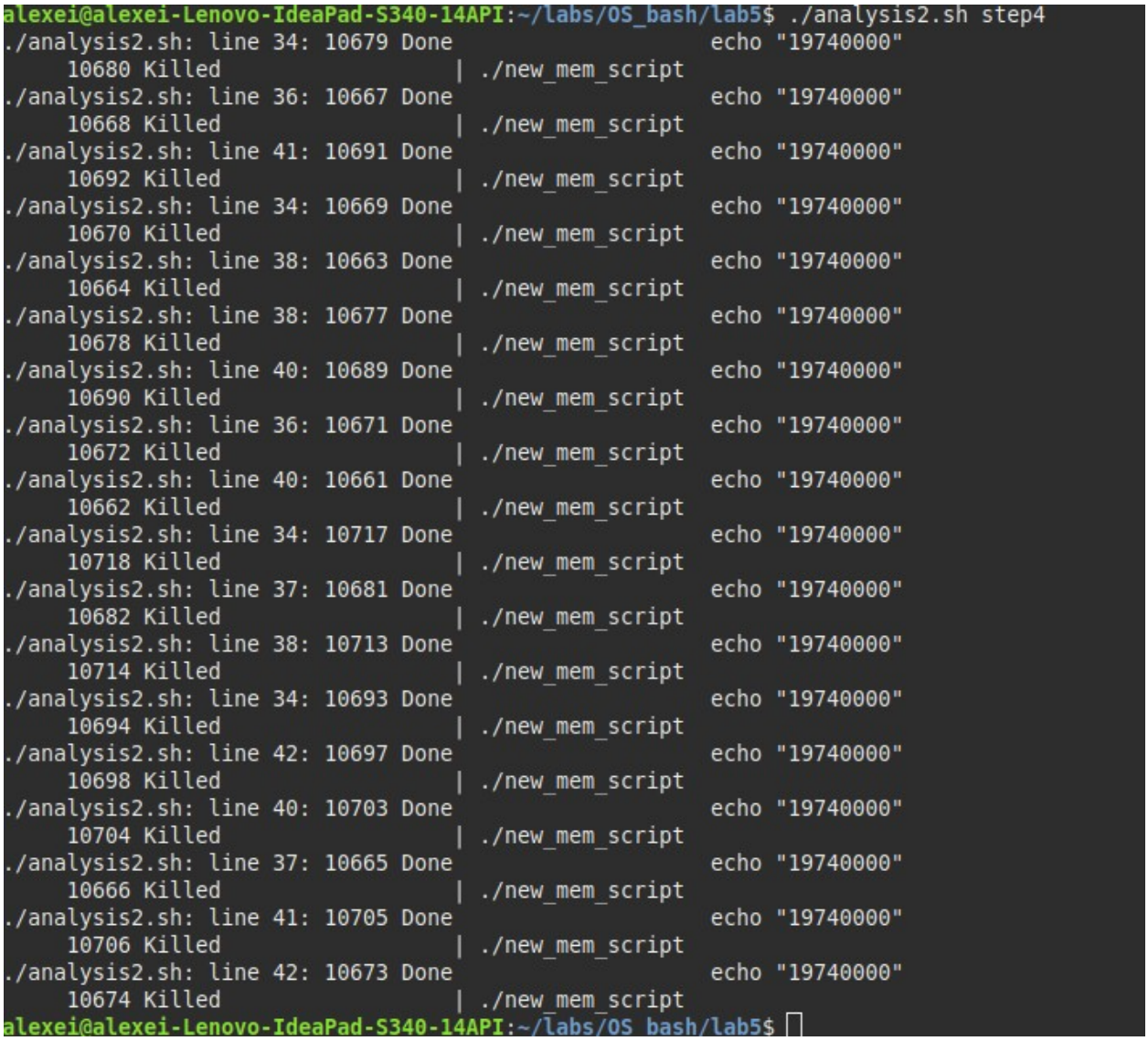


Рисунок 14. демонстрация терминала, эксперимент 2

Видим, что 18 программ было убито OOM-killer-ом, это произошло по причинам, которые возникали при выполнении 2-го этапа 1-го эксперимента.

Даже уменьшив объем выделяемой памяти, но увеличив количество таких программ, мы спровоцировали работу OOM-killer, он убил именно наши процессы, так как они никак не использовали страницы, которые для них выделяет ОС.

Далее требуется подобрать такое максимальное N, для того, чтобы не вызвался OOM-killer.

В ходе подбора N оказалось 8421340

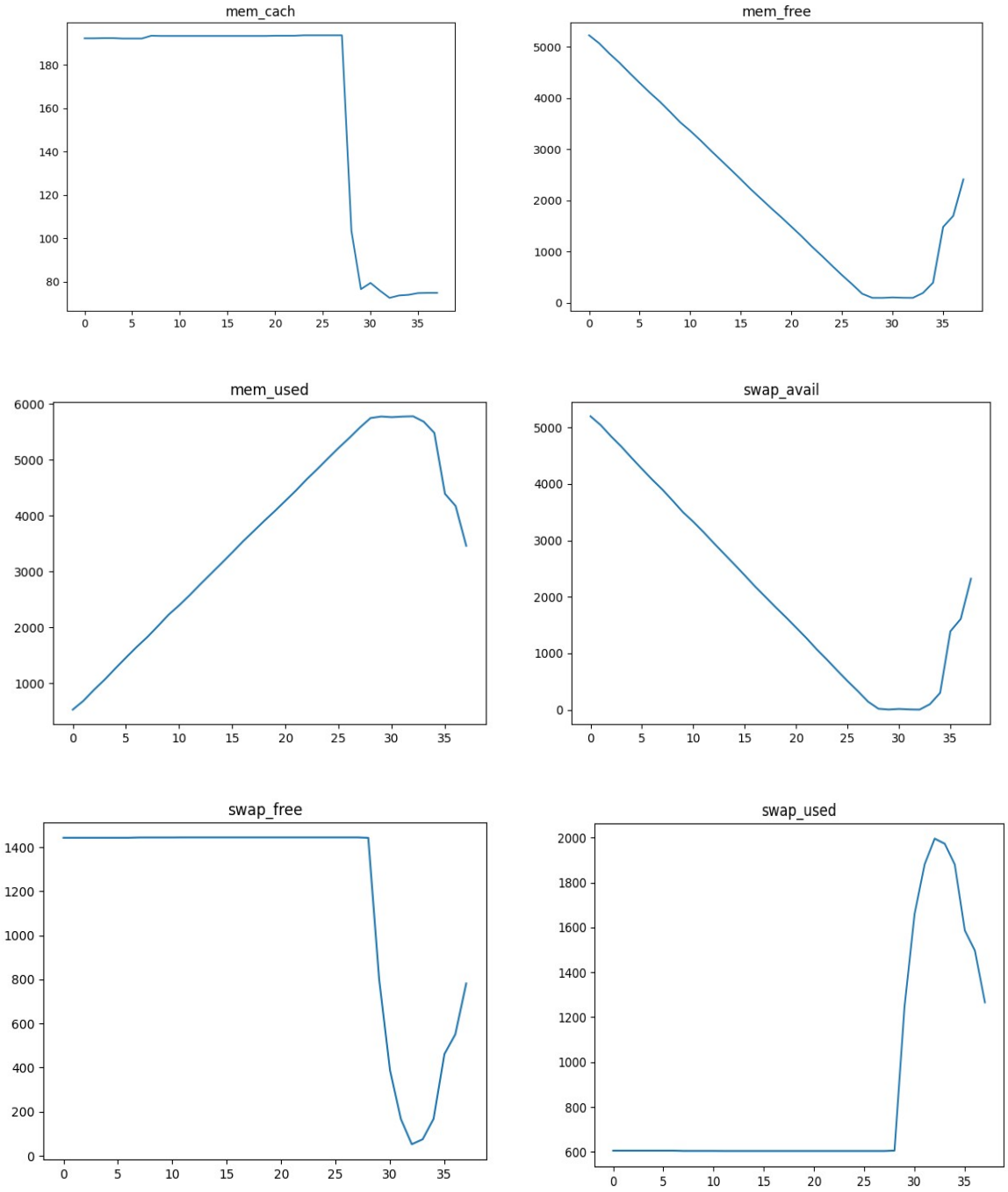


Рисунок 15. демонстрация величин, эксперимент 2

Вывод: в данном эксперименте были исследованы механизмы выделения памяти, тот факт, что для корректной работы системы Nnew > N*3, говорит нам о том, что эти механизмы корректно работают. Я считаю, что такое поведение было связано с тем, что после очистки 1-го процесса всем остальным достались его страницы и т. д. Пока все процессы не отработают.