УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Операционные системы»

Лабораторная работа №1

Студент

Бобрусь А.В.

P33091

Преподаватель

Саржевский И.А.

Санкт-Петербург, 2023 г.

Оглавление

Описание задания

Цель работы

Знакомство с системными инструментами анализа производительности и поведения программ для проведения нагрузочного тестирования операционной системы.

Задание

Построить графики (подходящие по заданию.):

- Потребления программой CPU;
- Нагрузки, генерируемой программой на подсистему ввода-вывода;
- Нагрузки, генерируемой программой на сетевую подсистему;
- Другие графики, необходимые для демонстрации работы.

Ход работы

Параметры процессора

```
root@2078049-gm08834:~/os_lab1# lscpu | awk 'NR==5 || NR==11 || NR==19'
CPU(s):                               1
Thread(s) per core:                   1
L1d cache:                            32 KiB (1 instance)
```

Вариант

Теперь необходимо нагрузить систему, используя утилиту stress-ng с параметрами, соответствующими варианту:

cpu: [rand, int32float];
cache: [l1cache-ssize, cache-ways];
io: [ioport, io-uring];
memory: [fork-vm, memrate];
network: [dccp, netdev];
pipe: [pipeherd, pipe-ops];
sched: [resched, sched-period]

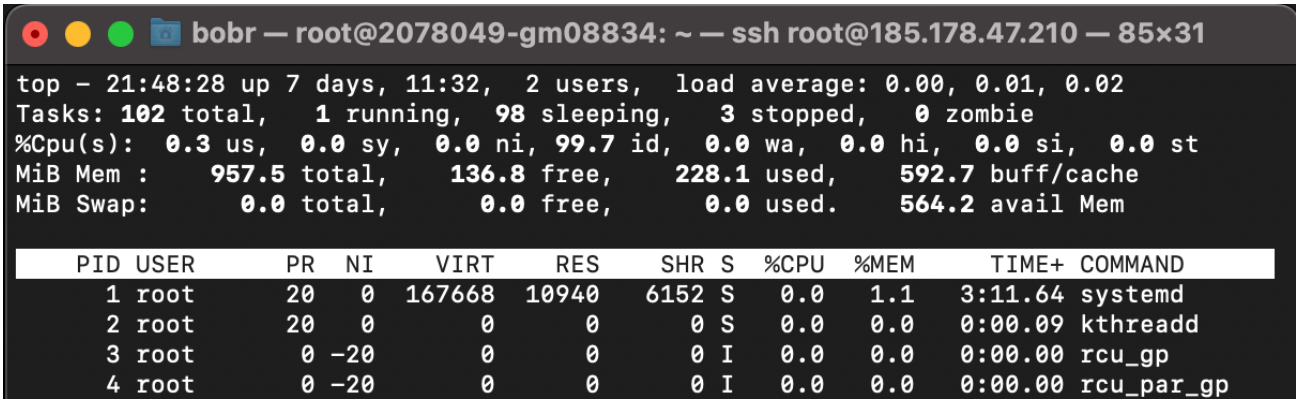
Код программы

Программа используется для автоматического запуска всех утилит, парсинга отслеживаемых значений и построения графиков: ссылки пока нет

CPU

Узнаем, при каком количестве процессов можно добиться максимальной производительности процессора.

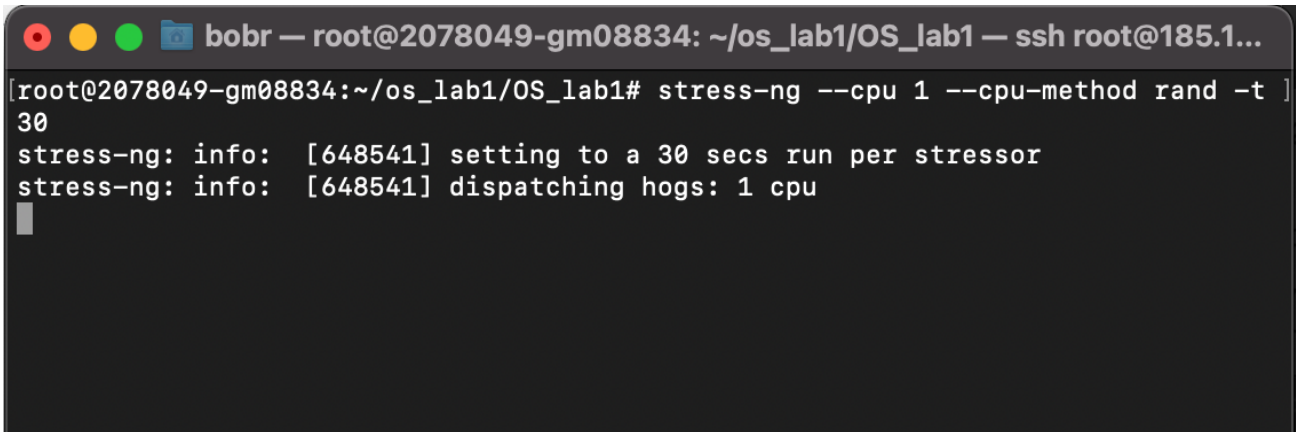
Для начала просмотрим потребление процессора при отсутствии нагрузки с помощью утилиты top:



Видим, что %CPU равен 0, значит получится сделать объективный анализ.

1) cpu-method rand

Теперь нагрузим процессор одним процессом и посмотрим, как изменилось потребление:

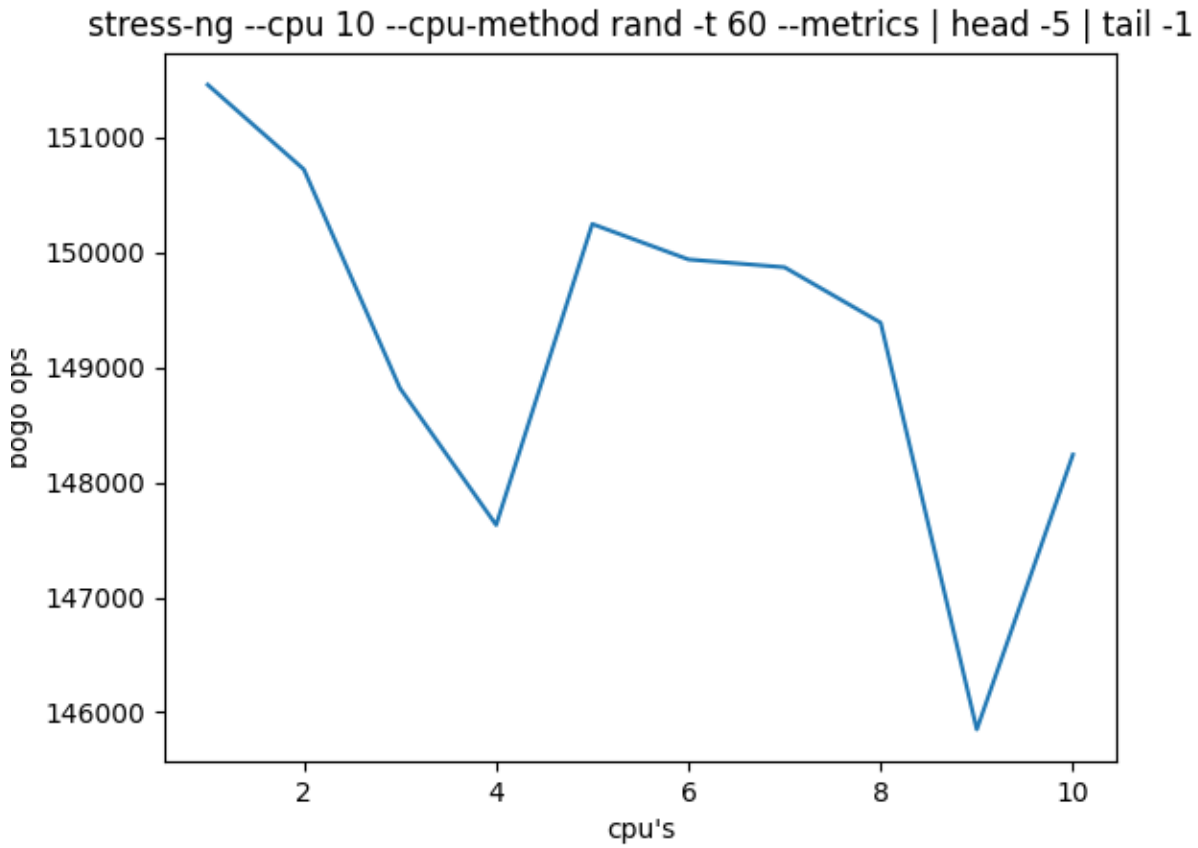


```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 79×33
top - 21:58:23 up 7 days, 11:42,  2 users,  load average: 0.41, 0.18, 0.08
Tasks: 106 total,   2 running, 101 sleeping,   3 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 s
MiB Mem :    957.5 total,    134.3 free,    228.4 used,    594.8 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    563.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 647415 root      20   0   61068   1268     32 R  99.7   0.1   0:19.84 stress+
 647410 root      20   0   10480   4060   3356 R   0.3   0.4   0:00.04 top
      1 root      20   0  167668  10940   6152 S   0.0   1.1   3:11.80 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.09 kthrea+
```

Видим, что %CPU стремиться к 100%, значит один процесс полностью загружает процессор и большее количество указывать нет смысла.

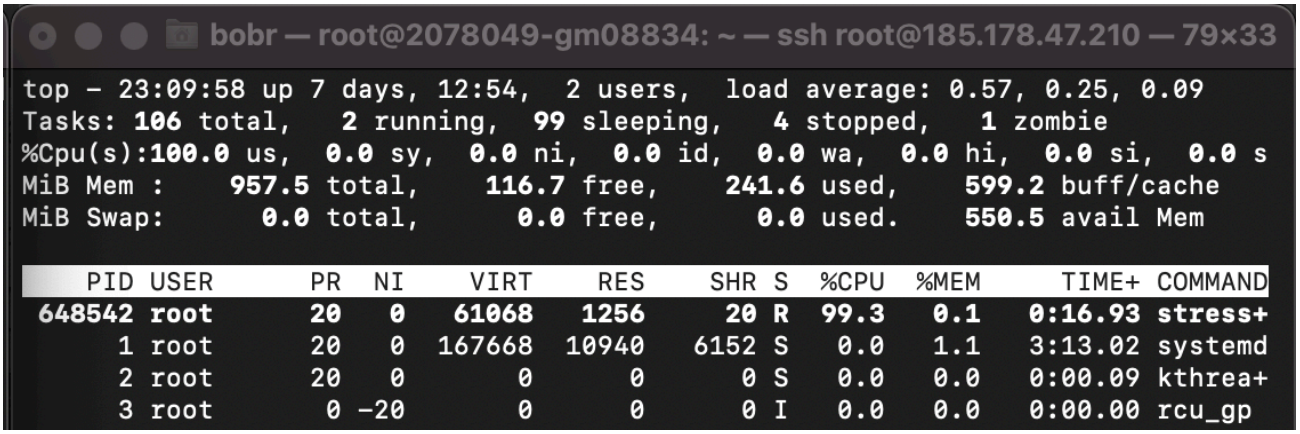
Убедимся в этом, построив график bogo ops в зависимости от количества процессов:



Из графика видно, что пик действительно достигается при 1 запущенном процессе.

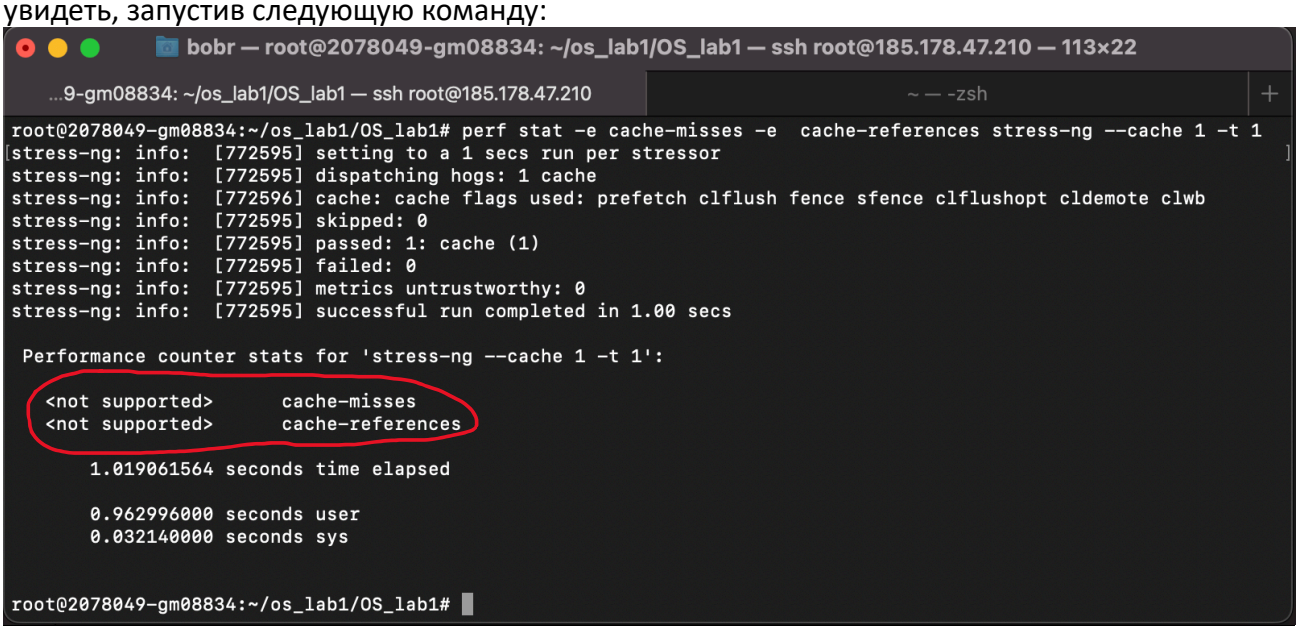
2) cpu-method int32float

Можно сделать вывод, что максимальной производительности получится достичь при количестве процессов равном произведению ядер процессора на их потоки (на тестируемом железе у процессора 1 ядро, 1 поток, в этом можно убедиться тут). Выявленное на предыдущем шаге количество процессов является оптимальным и для всех остальных методов. Запустим также утилиту top при работающем одном процессе с методом int32float и увидим аналогичную ситуацию – загрузка ядра максимальная. При увеличении числа процессов начнется просадка производительности из-за переключения контекстов.



Cache

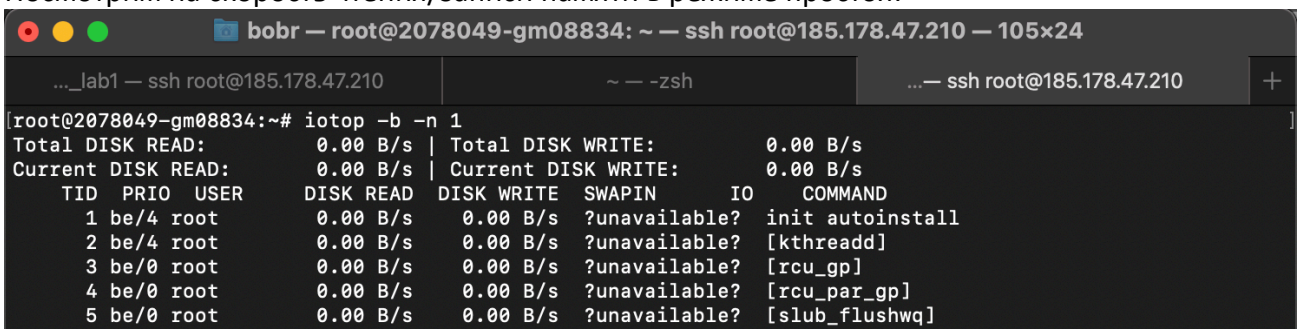
Для отслеживания динамики поведения кеша объективными характеристиками являются число промахов кеша и количество обращений к нему. Однако процессор, на котором запускается тестирование, не поддерживает отображение данных счетчиков. Это можно увидеть, запустив следующую команду:



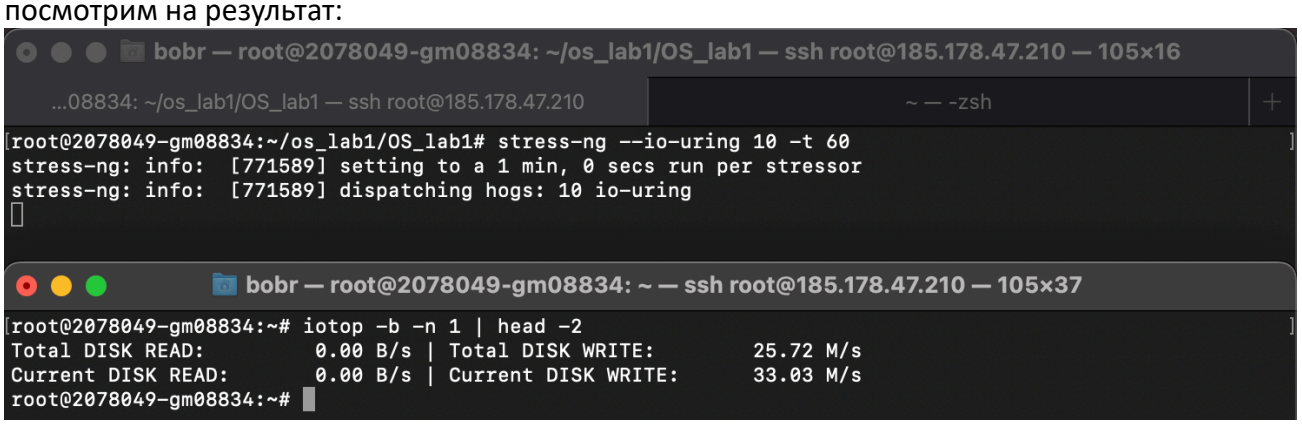
IO

1) io-uring

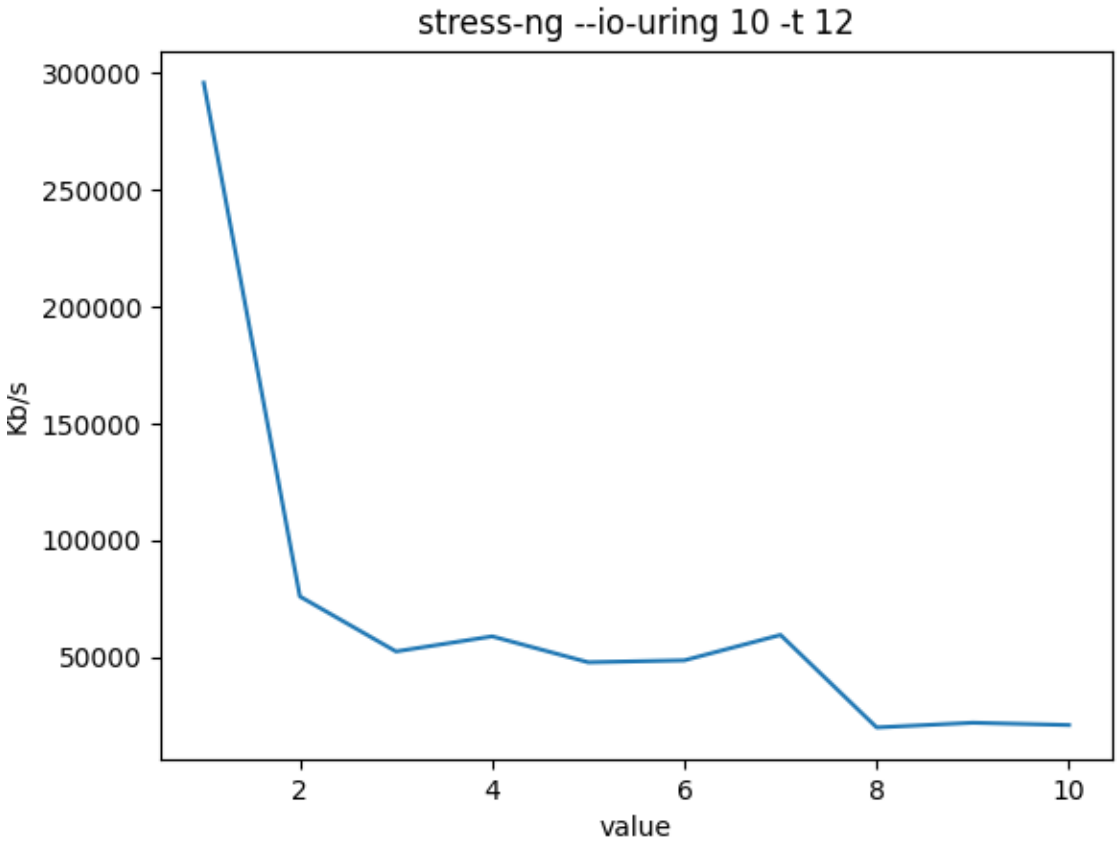
Посмотрим на скорость чтения/записи памяти в режиме простоя:



Видим, что в данный момент не происходит обращение к памяти. Теперь запустим тест и посмотрим на результат:



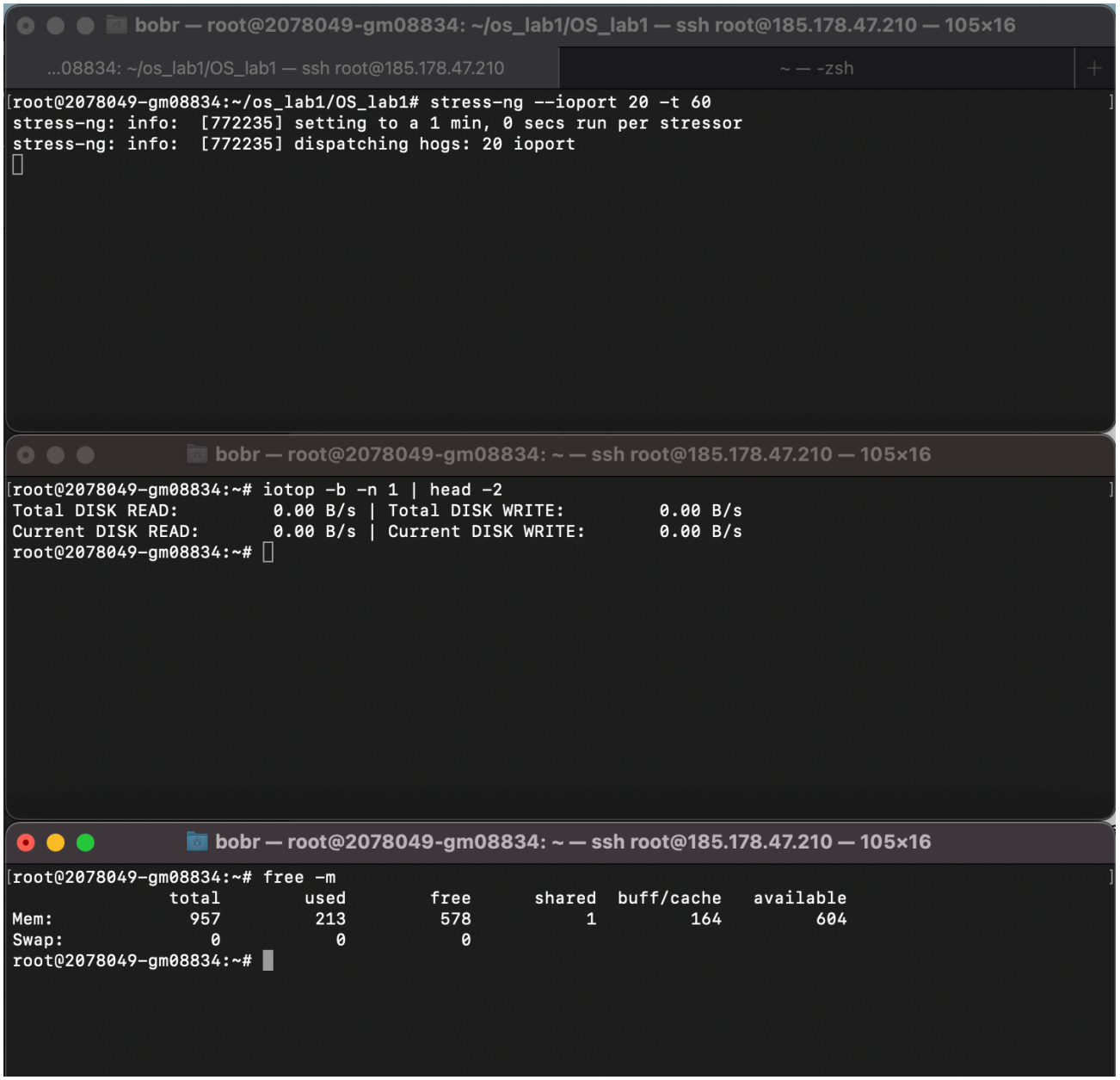
Видим, что тест производит операции записи. Поэтому построим график зависимости скорости записи диска относительно количества процессов io-uring:



Исходя из полученных результатов делаем вывод, что оптимальное количество процессов – 1.

2) Ioport

Запустим stress-ng с параметром –ioport 20 и посмотрим влияние на память:

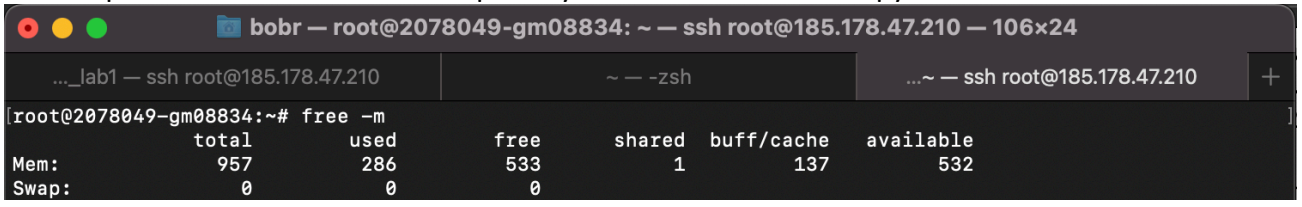


Никакого воздействия на память не происходит. Другими словами, мы получим константный ответ при запуске --ioport с любым значением.

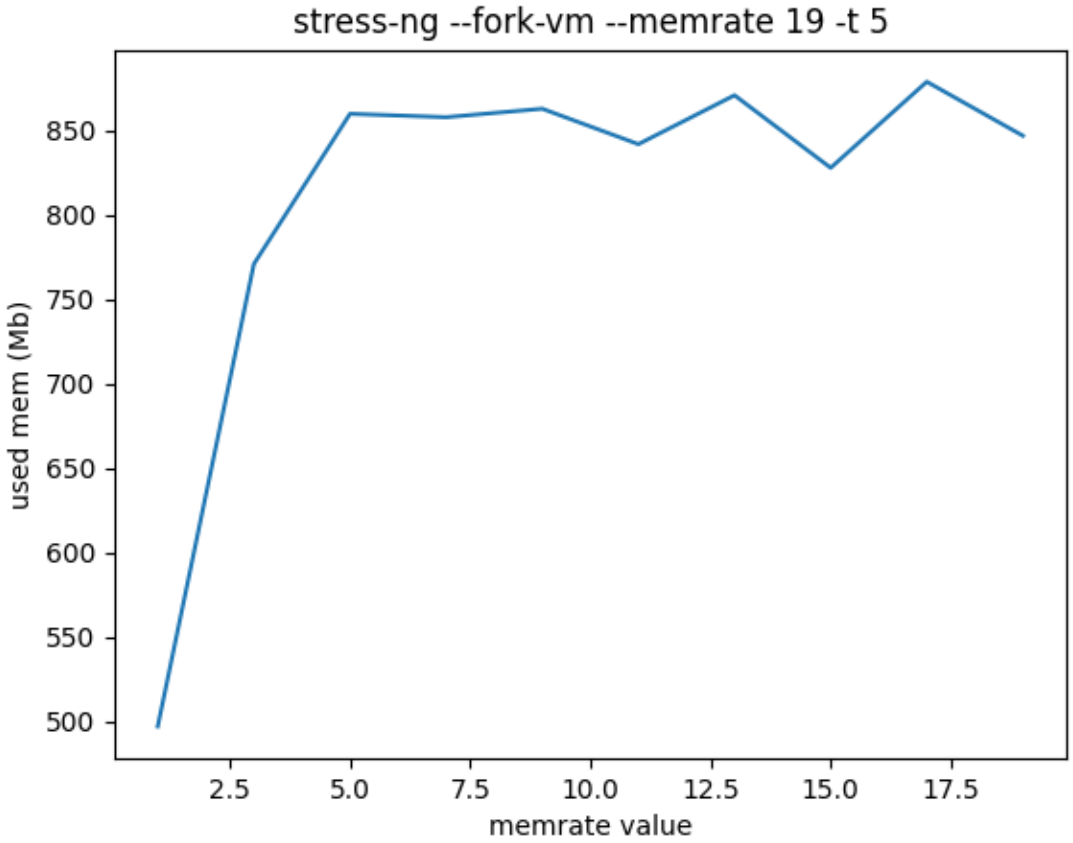
Memory

1) memrate

Посмотрим на состояние памяти при отсутствии какой-либо нагрузки:



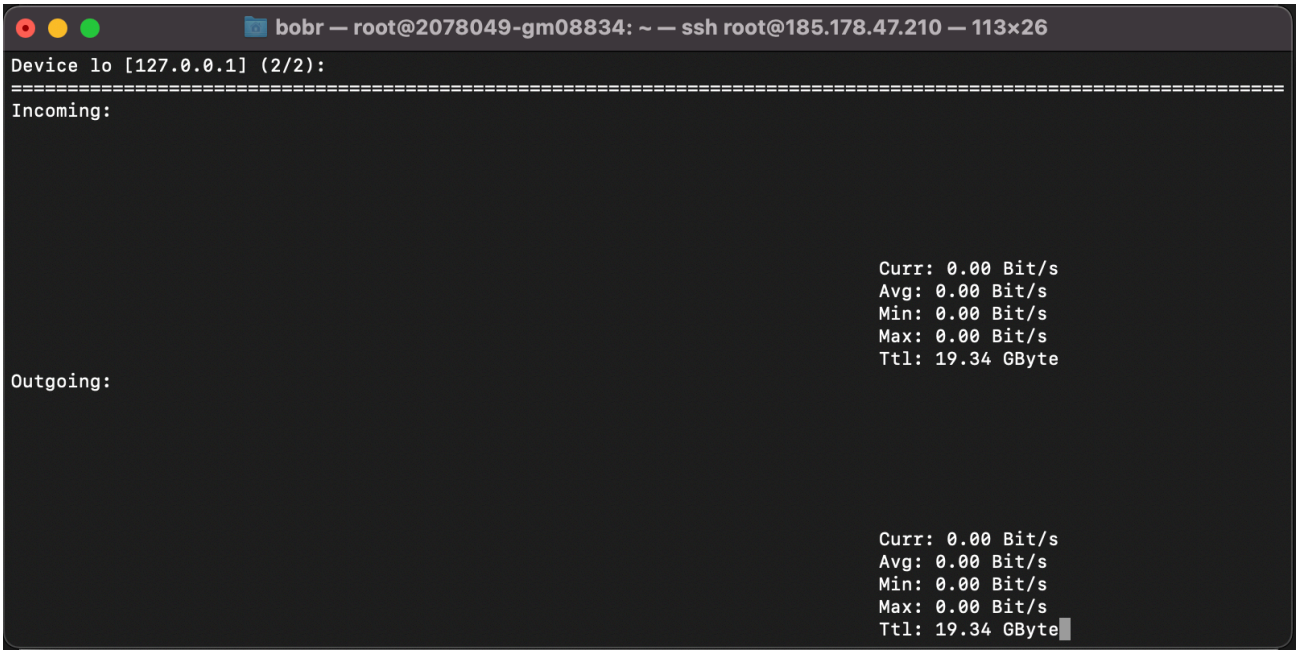
Видно, что использованной памяти 286Мб. Будем отталкиваться от этого значения: все, что выше него – потребление наших тестов. Построим график зависимости использованной памяти от количества процессов memrate:



Можем сделать вывод, что оптимальное число процессов memrate равно 5, затем память начинает заканчиваться и система начинает зависать.

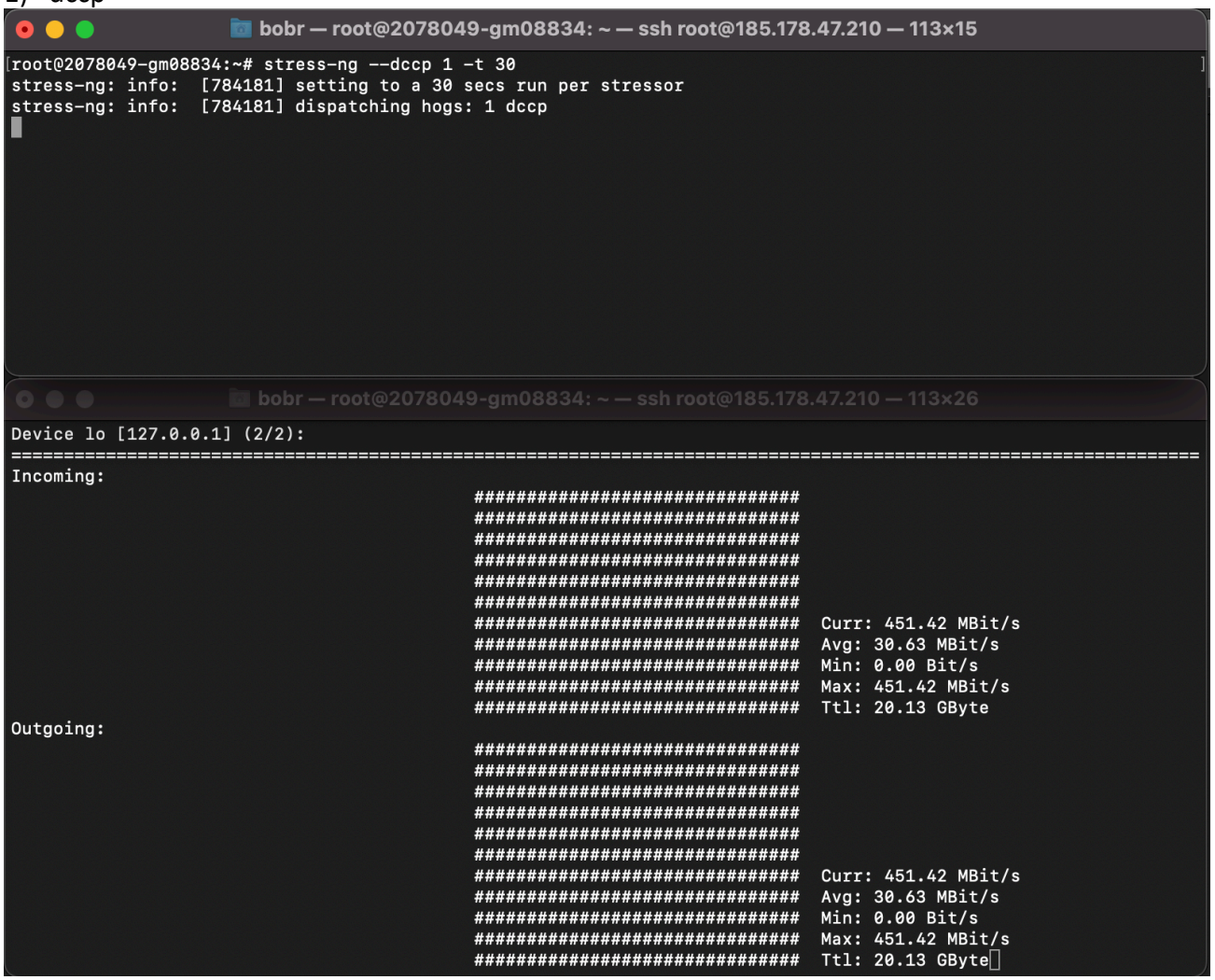
Network

Запустим утилиту nload в состоянии отсутствия искусственной нагрузки на сеть и посмотрим на производительность сетевого канала:

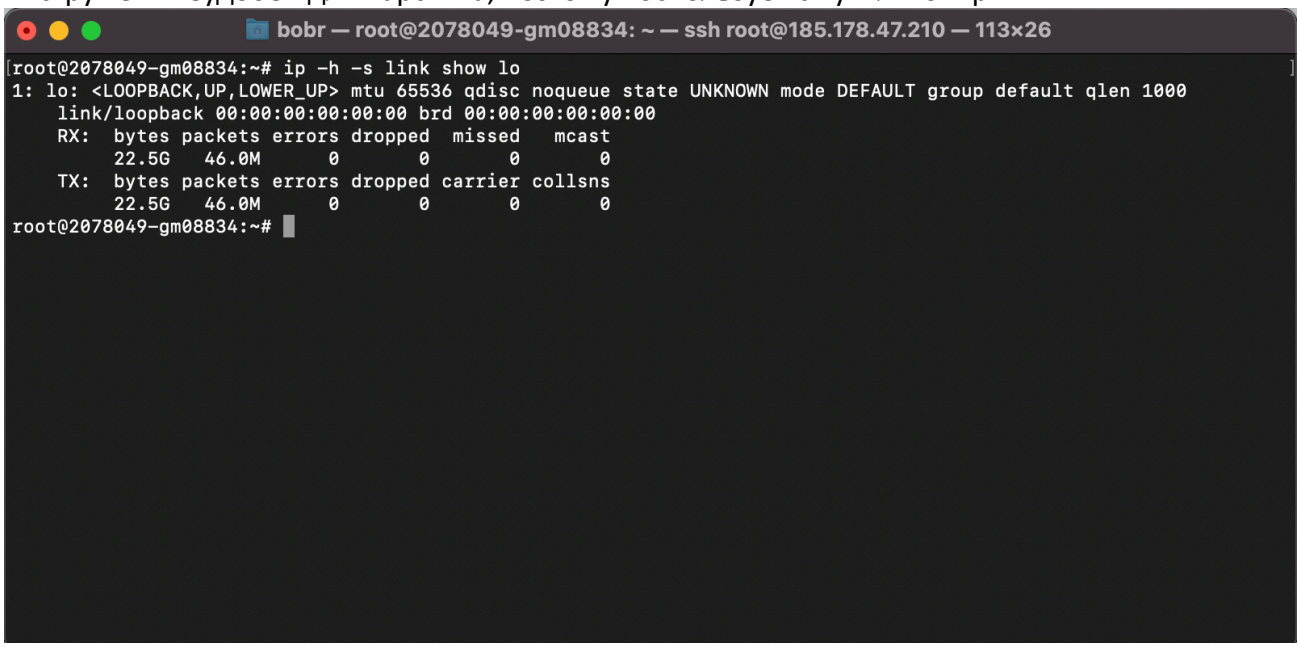


Все интересующие нас показатели – нулевые, это подтверждает отсутствие нагрузки. Теперь посмотрим, как измениться результат, если запустить тесты:

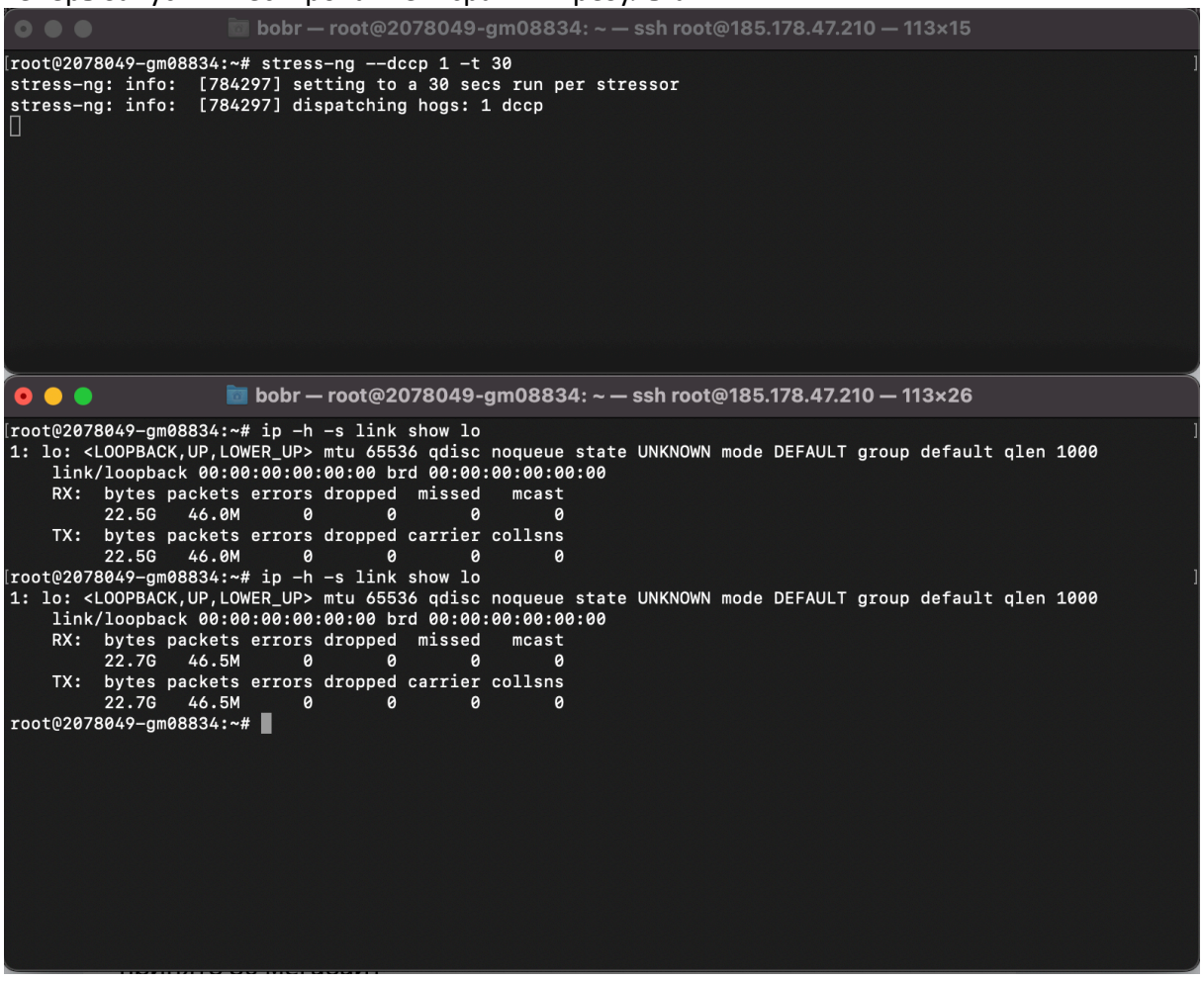
1) dccp



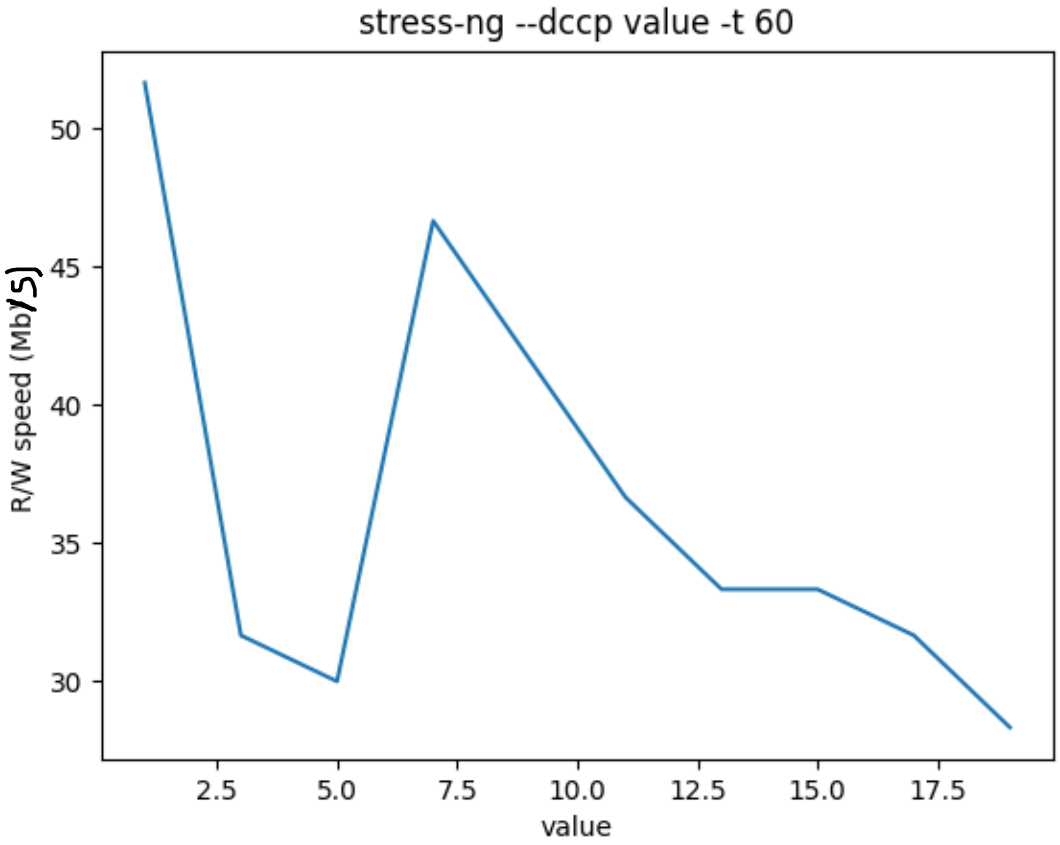
Видим результат: передача и прием сетевого канала работают с одинаковой скоростью, поэтому можем отслеживать только одну характеристику. Однако, интерактивный инструмент неудобен для парсинга, поэтому воспользуемся утилитой ip:



Нет скорости передачи, зато она отображает общее число переданной информации. Теперь запустим тестирование и сравним результат:



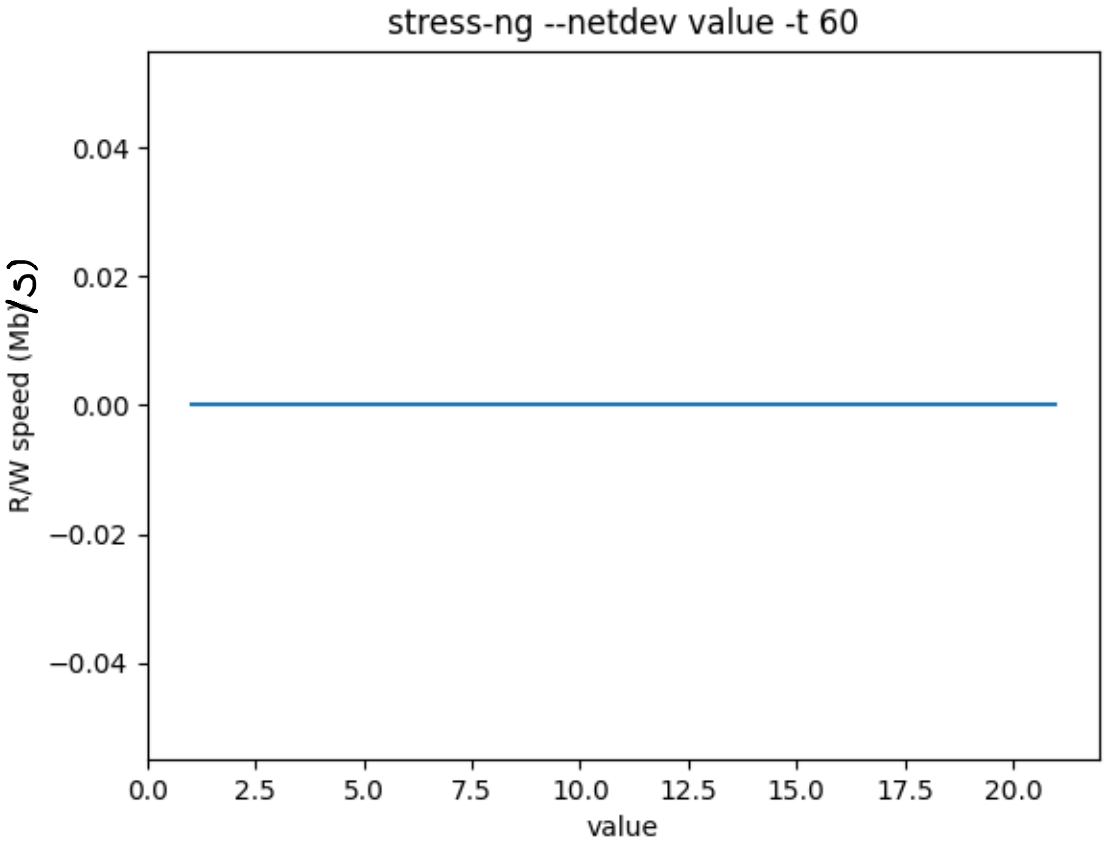
Результаты изменились. Объективной метрикой нагрузки на сеть будет скорость передачи, ее можно рассчитать по формуле: (кол-во байт в конце тестирования – количество байт в начале тестирования) / время тестирования. Строим график:



Общая тенденция скорости чтения/записи идет на снижение, следовательно оптимальное число процессов dccp равно единице.

2) netdev

Строим аналогичный график:



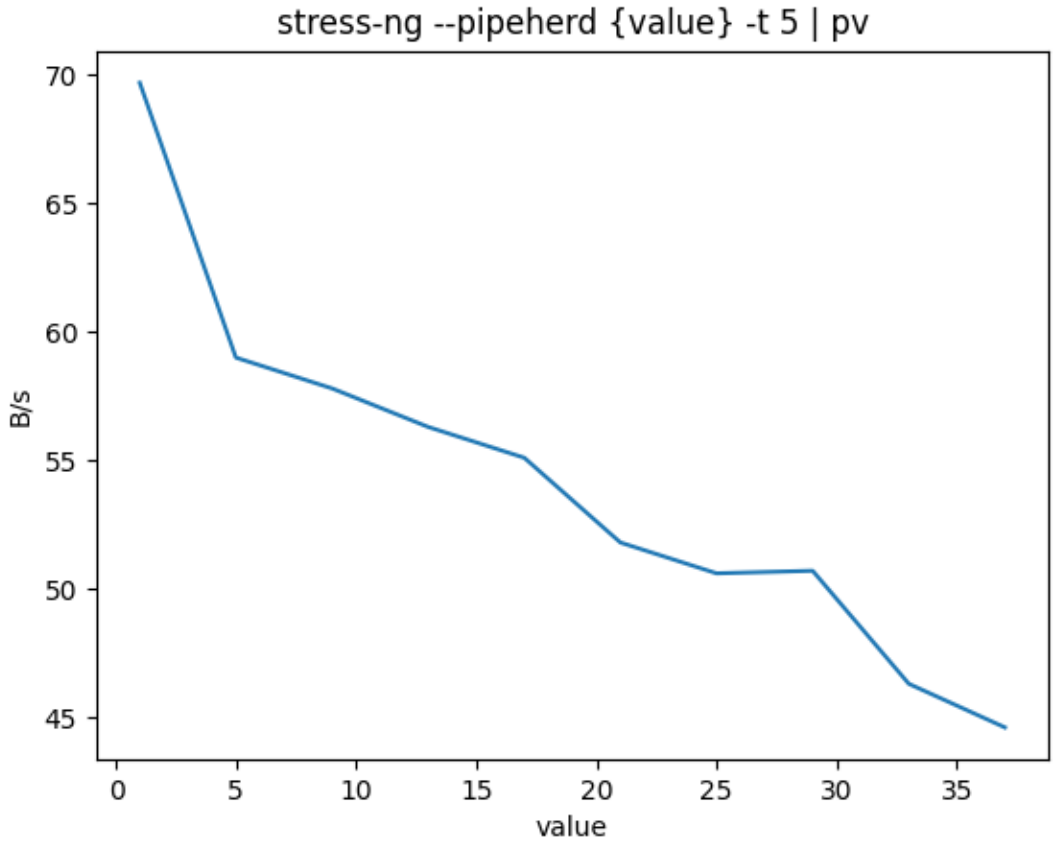
Посмотрим на загрузку процессора при 1 процессе netdev с помощью htop:

Загрузка процессора максимальная, значит оптимальное значение равно 1.

Pipe

1) pipeherd

Используя утилиту pv с параметром –pipeherd в интервале [1; 40] проанализируем скорость pipeline и построим график:



На графике видна четкая тенденция на убывание, делаем вывод, что оптимальное число процессов равно 1.

2) pipe-ops

Посмотрим как влияет значение pipe-ops на скорость pipeline (возьмем 10 чисел с равным интервалом на промежутке от 1 до 100):

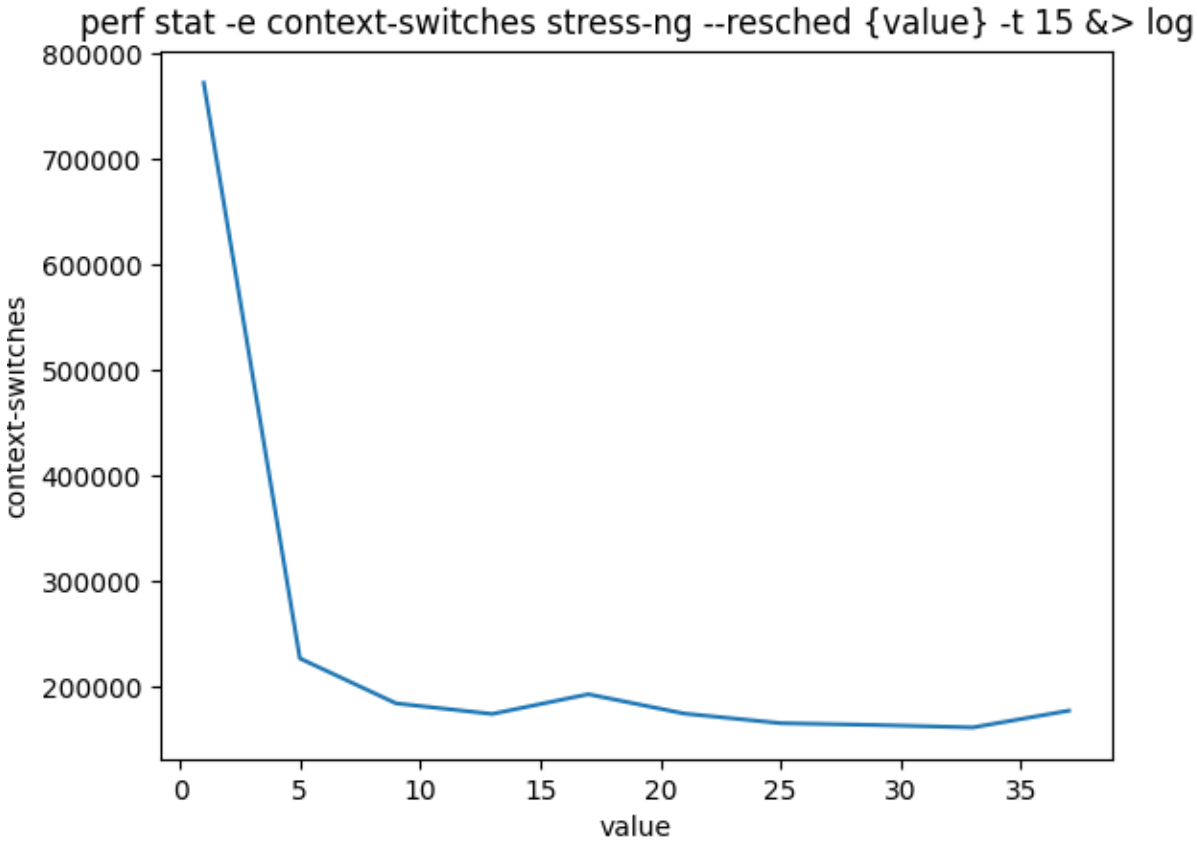
```
Enter parameter name from list: cpu, memory, io, network, pipe
[>>>pipe
[Enter time per test >>>10
[Enter min range value >>>1
[Enter max range value >>>100
[Enter needed pipe method from list: ['pipeherd', 'pipe-ops'] >>>pipe-ops
command: stress-ng --pipe-ops 1 -t 10 | pv
 118 B 0:00:00 [16.7KiB/s] [  <=>
command: stress-ng --pipe-ops 11 -t 10 | pv
 118 B 0:00:00 [13.5KiB/s] [  <=>
command: stress-ng --pipe-ops 21 -t 10 | pv
 118 B 0:00:00 [9.72KiB/s] [  <=>
command: stress-ng --pipe-ops 31 -t 10 | pv
 118 B 0:00:00 [13.7KiB/s] [  <=>
command: stress-ng --pipe-ops 41 -t 10 | pv
 118 B 0:00:00 [10.8KiB/s] [  <=>
command: stress-ng --pipe-ops 51 -t 10 | pv
 118 B 0:00:00 [13.6KiB/s] [  <=>
command: stress-ng --pipe-ops 61 -t 10 | pv
 118 B 0:00:00 [9.92KiB/s] [  <=>
command: stress-ng --pipe-ops 71 -t 10 | pv
 118 B 0:00:00 [9.56KiB/s] [  <=>
command: stress-ng --pipe-ops 81 -t 10 | pv
 118 B 0:00:00 [9.68KiB/s] [  <=>
command: stress-ng --pipe-ops 91 -t 10 | pv
 118 B 0:00:00 [20.4KiB/s] [  <=>
```

Видим, что скорость константная. Суть заключается в том, что параметр pipe-ops лишь указывает на максимальное количество операций pipeline, а не нагружает pipeline.

Sheduler

1) resched

Построим график зависимости переключения контекстов от значения resched (возьмем 10 чисел с равным интервалом на промежутке [1; 40]):



Видим практически экспоненциальное убывание, из которого можно понять следующее: чем меньше значение resched, тем больше переключений контекста, т.е. оптимальное значение равно 1.

2) sched-period

Данный параметр лишь ограничивает время, которое будет выполняться работа scheduler.

```
root@2078049-gm08834:~/os_lab1/OS_lab1# stress-ng --help | grep sched-period
      --sched-period N        set period for SCHED_DEADLINE to N nanosecs
```

Однако, в официальном github stress-ng указано, что данный функционал был удален.

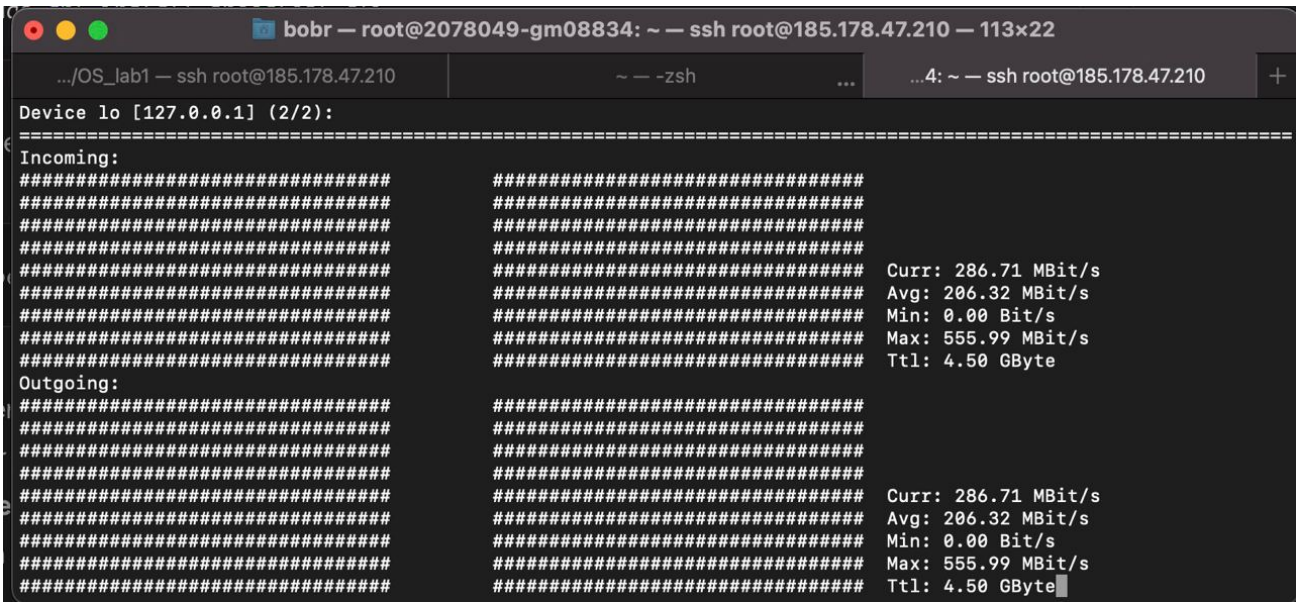
core-sched.c	core-sched: remove unused function stress_set_deadline_sched	3 months ago
core-sched.h	core-sched: remove unused function stress_set_deadline_sched	3 months ago

Вывод

В ходе выполнения данной работы я научился выявлять точки экстремума производительности ПО (stress-ng в данном случае имитировал работу программы) на конкретном железе. Это очень важный навык, который помогает рассчитать нагрузку на сервер при разработке программного обеспечения.

Бонус

Если при помощи stress-ng подать правильную рассчитанную нагрузку на сеть, то в утилите nload можно нарисовать флаг Швеции:


bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 113×22
.../OS_lab1 — ssh root@185.178.47.210
~ — -zsh
...4: ~ — ssh root@185.178.47.210
Device lo [127.0.0.1] (2/2):
Incoming:
Curr: 286.71 MBit/s
Avg: 206.32 MBit/s
Min: 0.00 Bit/s
Max: 555.99 MBit/s
Ttl: 4.50 GByte
Outgoing:
Curr: 286.71 MBit/s
Avg: 206.32 MBit/s
Min: 0.00 Bit/s
Max: 555.99 MBit/s
Ttl: 4.50 GByte