

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.....	3
Цель работы.....	3
Задание	3
Ход работы	3
Параметры процессора	3
Вариант	3
Код программы	3
CPU	4
Cache	6
IO	7
Memory	9
Network	10
Pipe.....	14
Sheduler.....	16
Вывод	17
Бонус.....	17

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 посмотрим потребление процессора при отсутствии нагрузки с помощью утилиты top:

```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 85x31
top - 21:48:28 up 7 days, 11:32, 2 users, load average: 0.00, 0.01, 0.02
Tasks: 102 total, 1 running, 98 sleeping, 3 stopped, 0 zombie
%Cpu(s): 0.3 us, 0.0 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 957.5 total, 136.8 free, 228.1 used, 592.7 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 564.2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	167668	10940	6152	S	0.0	1.1	3:11.64	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.09	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp

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

1) cpu-method rand

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

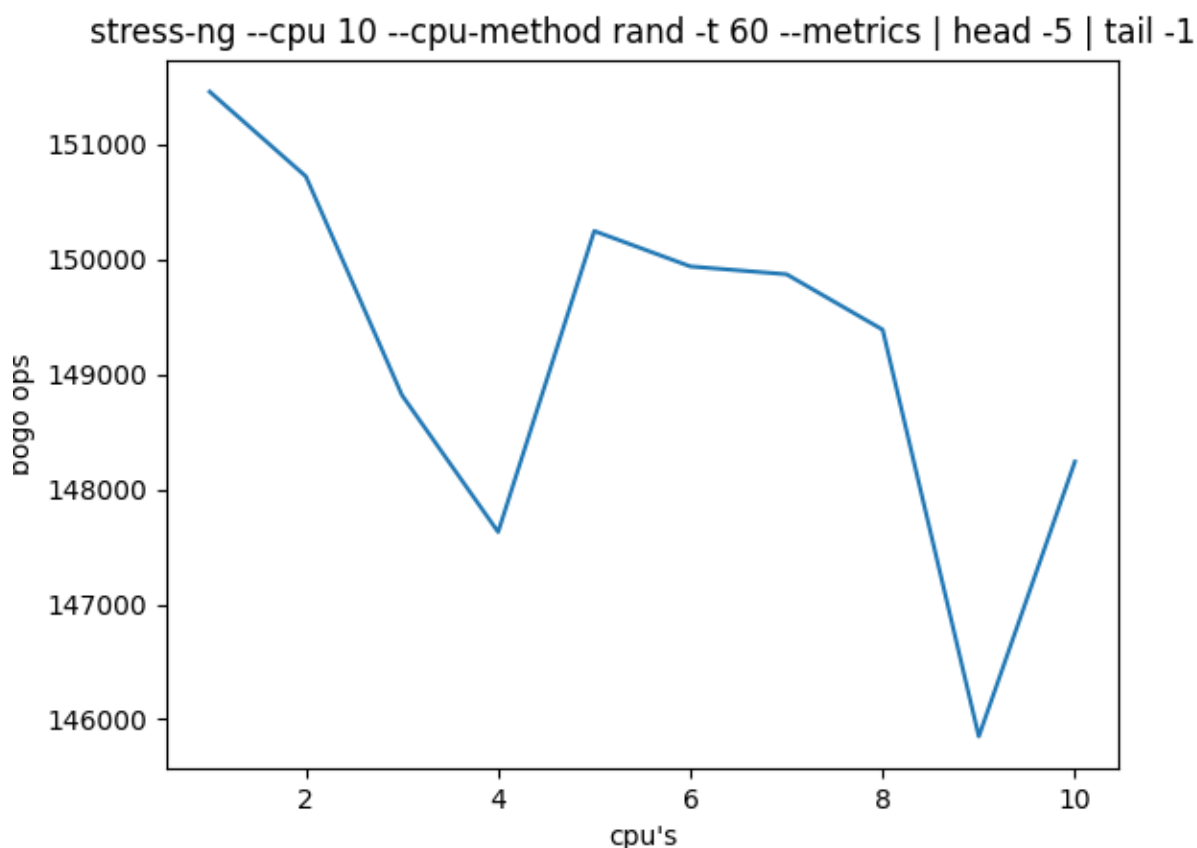
```
bobr — root@2078049-gm08834: ~/os_lab1/OS_lab1 — ssh root@185.1...
[root@2078049-gm08834:~/os_lab1/OS_lab1# stress-ng --cpu 1 --cpu-method rand -t 30
stress-ng: info: [648541] setting to a 30 secs run per stressor
stress-ng: info: [648541] dispatching hogs: 1 cpu
```

```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 79x33
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 st
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 стремится 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.

```

bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 79x33
top - 23:09:58 up 7 days, 12:54,  2 users,  load average: 0.57, 0.25, 0.09
Tasks: 106 total,   2 running,  99 sleeping,   4 stopped,   1 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 st
MiB Mem :  957.5 total,  116.7 free,  241.6 used,  599.2 buff/cache
MiB Swap:   0.0 total,   0.0 free,   0.0 used.  550.5 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 648542 root      20   0   61068   1256    20  R   99.3    0.1    0:16.93 stress+
    1 root      20   0 167668  10940   6152  S    0.0    1.1    3:13.02 systemd
    2 root      20   0     0     0     0  S    0.0    0.0    0:00.09 kthrea+
    3 root       0 -20     0     0     0  I    0.0    0.0    0:00.00 rcu_gp

```

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:

```

bobr — root@2078049-gm08834: ~/os_lab1/OS_lab1 — ssh root@185.178.47.210 — 113x22
...9-gm08834: ~/os_lab1/OS_lab1 — ssh root@185.178.47.210
root@2078049-gm08834:~/os_lab1/OS_lab1# perf stat -e cache-misses -e cache-references stress-ng --cache 1 -t 1
stress-ng: info: [772595] setting to a 1 secs run per stressor
stress-ng: info: [772595] dispatching hogs: 1 cache
stress-ng: info: [772596] cache: cache flags used: prefetch clflush fence sfence clflushopt cldemote clwb
stress-ng: info: [772595] skipped: 0
stress-ng: info: [772595] passed: 1: cache (1)
stress-ng: info: [772595] failed: 0
stress-ng: info: [772595] metrics untrustworthy: 0
stress-ng: info: [772595] successful run completed in 1.00 secs

Performance counter stats for 'stress-ng --cache 1 -t 1':
<not supported>      cache-misses
<not supported>      cache-references

1.019061564 seconds time elapsed

0.962996000 seconds user
0.032140000 seconds sys

root@2078049-gm08834:~/os_lab1/OS_lab1#

```

IO

1) io-uring

Посмотрим на скорость чтения/записи памяти в режиме простоя:

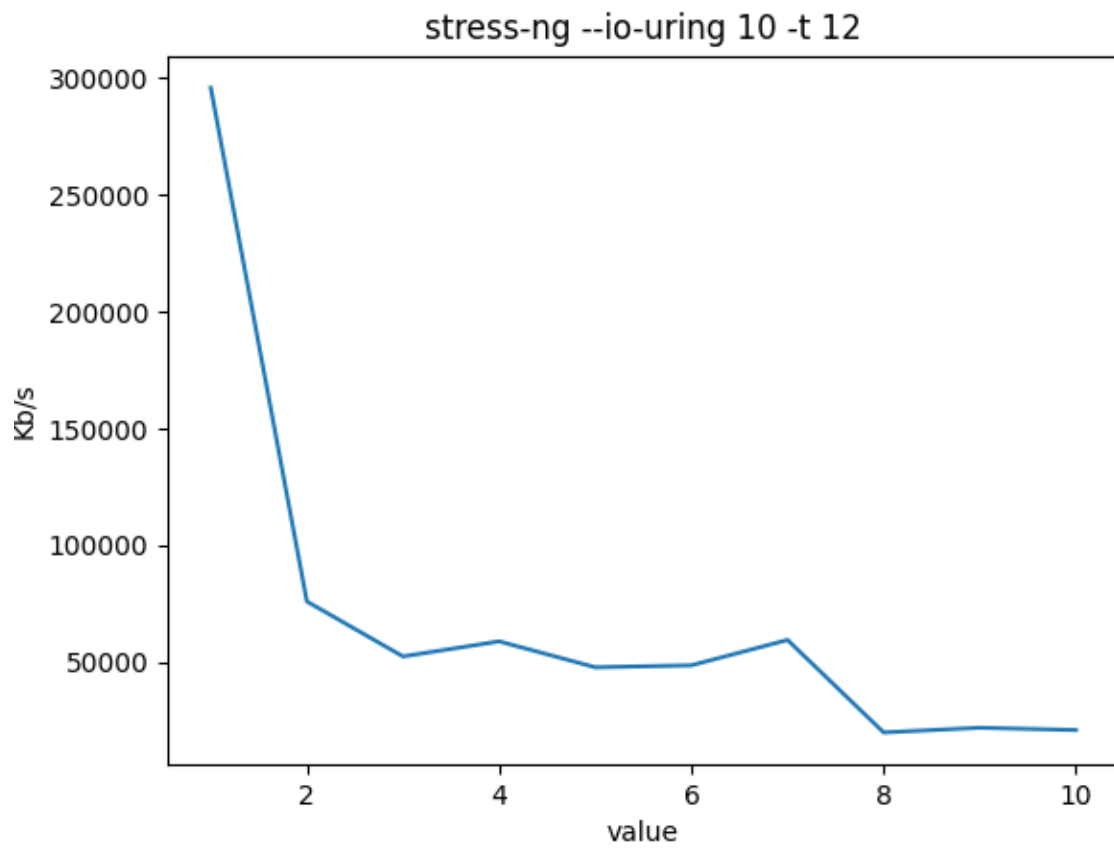
```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 105x24
..._lab1 — ssh root@185.178.47.210      ~ — -zsh      ... — ssh root@185.178.47.210
root@2078049-gm08834:~# iotop -b -n 1
Total DISK READ:      0.00 B/s | Total DISK WRITE:      0.00 B/s
Current DISK READ:    0.00 B/s | Current DISK WRITE:    0.00 B/s
   TID  PRIO  USER    DISK READ  DISK WRITE  SWAPIN      IO   COMMAND
   ---  ---  ---
    1  be/4  root      0.00 B/s   0.00 B/s   ?unavailable?  init autoinstall
    2  be/4  root      0.00 B/s   0.00 B/s   ?unavailable?  [kthreadd]
    3  be/0  root      0.00 B/s   0.00 B/s   ?unavailable?  [rcu_gp]
    4  be/0  root      0.00 B/s   0.00 B/s   ?unavailable?  [rcu_par_gp]
    5  be/0  root      0.00 B/s   0.00 B/s   ?unavailable?  [slub_flushwq]
```

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

```
bobr — root@2078049-gm08834: ~/os_lab1/OS_lab1 — ssh root@185.178.47.210 — 105x16
...08834: ~/os_lab1/OS_lab1 — ssh root@185.178.47.210      ~ — -zsh
root@2078049-gm08834:~/os_lab1/OS_lab1# stress-ng --io-uring 10 -t 60
stress-ng: info:  [771589] setting to a 1 min, 0 secs run per stressor
stress-ng: info:  [771589] dispatching hogs: 10 io-uring
█

bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 105x37
root@2078049-gm08834:~# iotop -b -n 1 | head -2
Total DISK READ:      0.00 B/s | Total DISK WRITE:      25.72 M/s
Current DISK READ:    0.00 B/s | Current DISK WRITE:    33.03 M/s
root@2078049-gm08834:~# █
```

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:


```
bobr — root@2078049-gm08834: ~/os_lab1/OS_lab1 — ssh root@185.178.47.210 — 105x16
...08834: ~/os_lab1/OS_lab1 — ssh root@185.178.47.210 ~ — -zsh +
[root@2078049-gm08834:~/os_lab1/OS_lab1# stress-ng --ioport 20 -t 60
stress-ng: info: [772235] setting to a 1 min, 0 secs run per stressor
stress-ng: info: [772235] dispatching hogs: 20 ioport
]

[root@2078049-gm08834:~# iotop -b -n 1 | head -2
Total DISK READ:      0.00 B/s | Total DISK WRITE:      0.00 B/s
Current DISK READ:    0.00 B/s | Current DISK WRITE:    0.00 B/s
root@2078049-gm08834:~# ]

bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 105x16
[root@2078049-gm08834:~# free -m
total      used      free      shared  buff/cache   available
Mem:        957        213        578         1         164         604
Swap:         0           0           0
root@2078049-gm08834:~# ]
```

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

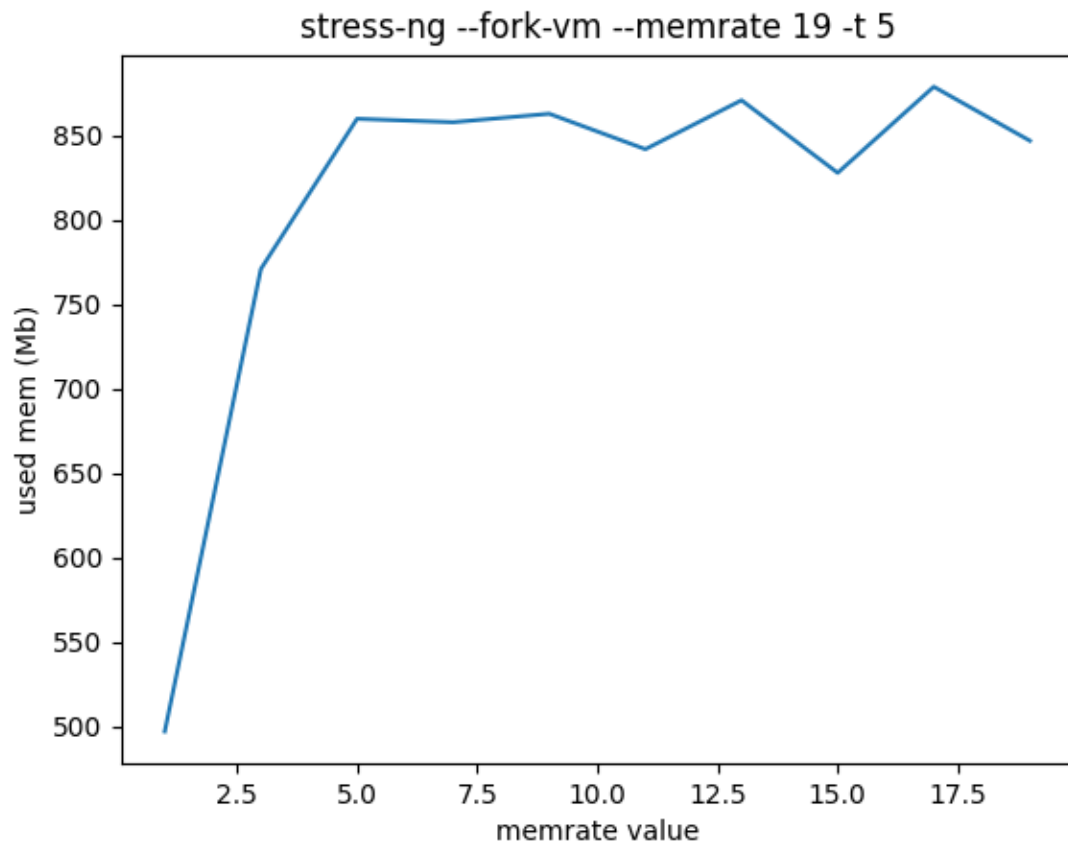
Memory

1) memrate

Посмотрим на состояние памяти при отсутствии какой-либо нагрузки:

```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 106x24
..._lab1 — ssh root@185.178.47.210 ~ — -zsh ...~ — ssh root@185.178.47.210 +
[root@2078049-gm08834:~# free -m
total      free      used      shared  buff/cache   available
Mem:        957        533        286         1         137         532
Swap:         0           0           0
```

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:

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:

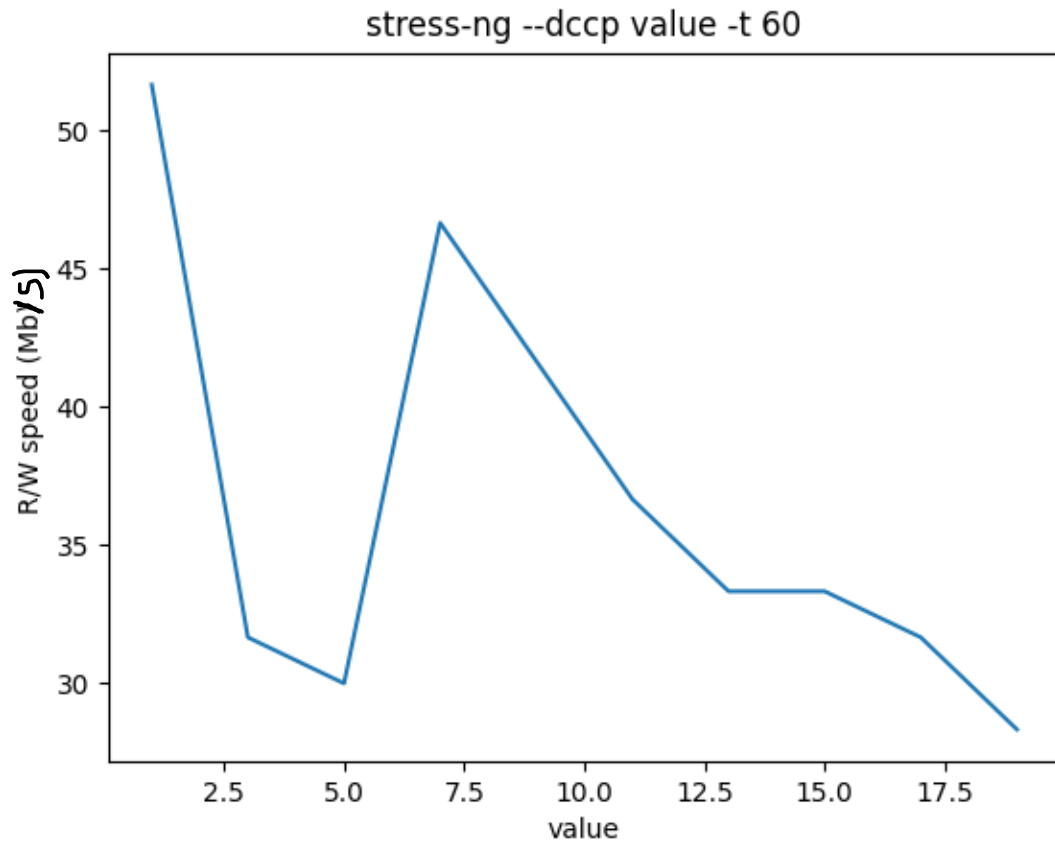
```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 113x26
root@2078049-gm08834:~# ip -h -s link show lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    RX:  bytes packets errors dropped missed mcast
         22.5G  46.0M      0      0      0      0
    TX:  bytes packets errors dropped carrier collsns
         22.5G  46.0M      0      0      0      0
root@2078049-gm08834:~#
```

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

```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 113x15
root@2078049-gm08834:~# stress-ng --dccp 1 -t 30
stress-ng: info:  [784297] setting to a 30 secs run per stressor
stress-ng: info:  [784297] dispatching hogs: 1 dccp
█
```

```
bobr — root@2078049-gm08834: ~ — ssh root@185.178.47.210 — 113x26
root@2078049-gm08834:~# ip -h -s link show lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    RX:  bytes packets errors dropped missed mcast
         22.5G  46.0M      0      0      0      0
    TX:  bytes packets errors dropped carrier collsns
         22.5G  46.0M      0      0      0      0
root@2078049-gm08834:~# ip -h -s link show lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    RX:  bytes packets errors dropped missed mcast
         22.7G  46.5M      0      0      0      0
    TX:  bytes packets errors dropped carrier collsns
         22.7G  46.5M      0      0      0      0
root@2078049-gm08834:~#
```

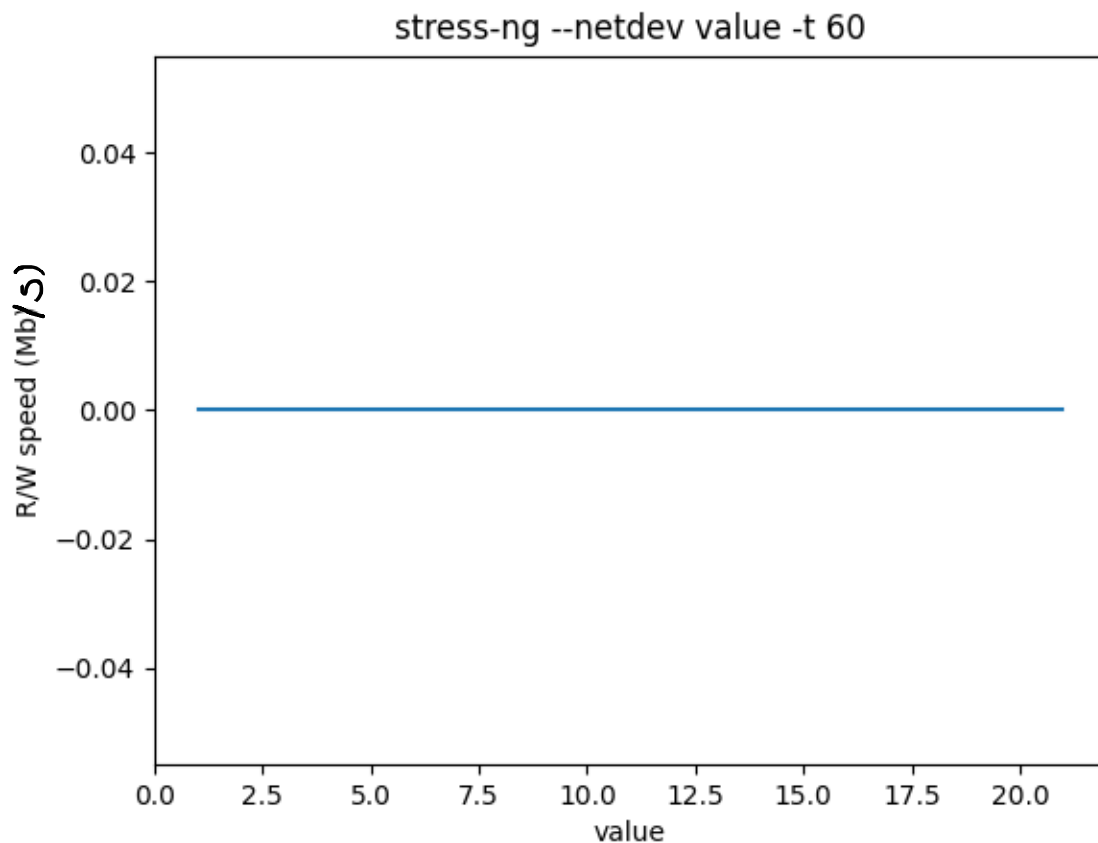
Результаты изменились. Объективной метрикой нагрузки на сеть будет скорость передачи, ее можно рассчитать по формуле: (кол-во байт в конце тестирования – количество байт в начале тестирования) / время тестирования. Строим график:



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:

```

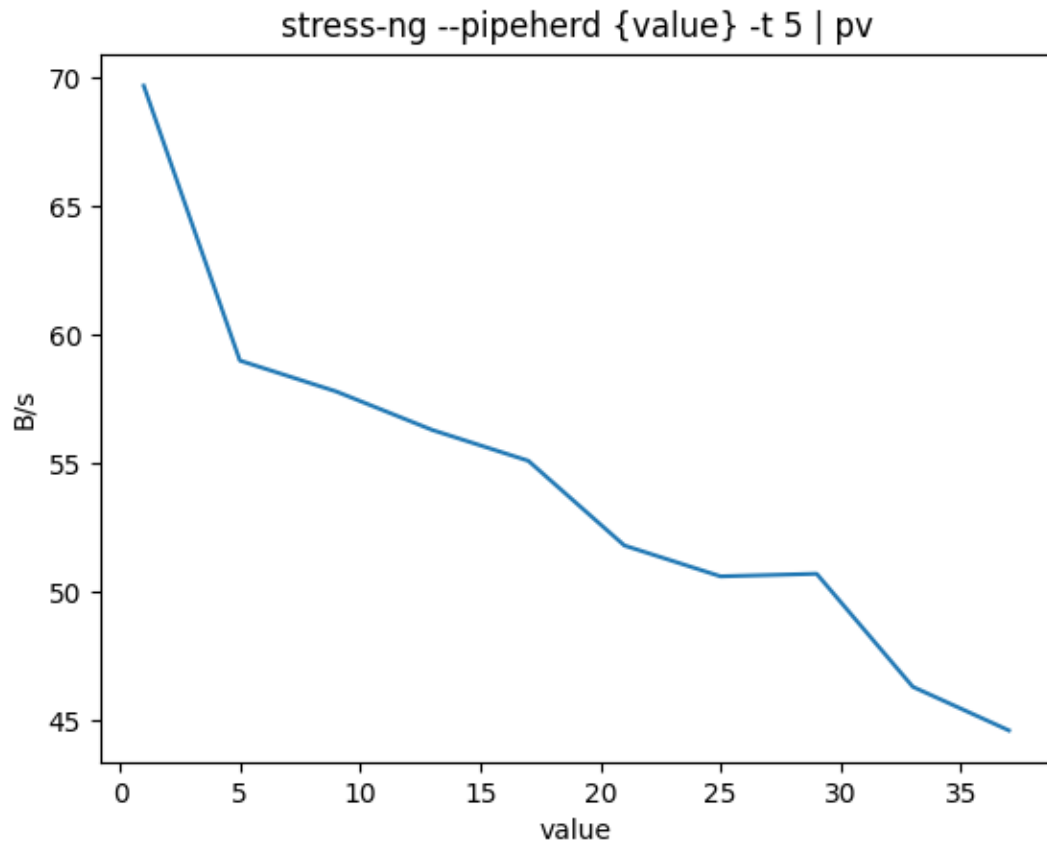
CPU[|||||||||||||||||||||||||||||||||||||||||100.0%]  Tasks: 44, 41 thr; 1 running
Mem[|||||||||||||||||||||||||||||||||||||260M/958M]  Load average: 0.08 0.02 0.05
Swp[|||||||||||||||||||||||||||||||||0K/0K]          Uptime: 15 days, 09:45:39
  
```

Загрузка процессора максимальная, значит оптимальное значение равно 1.

Pipe

1) pipeherd

Используя утилиту rv 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] [ <=>

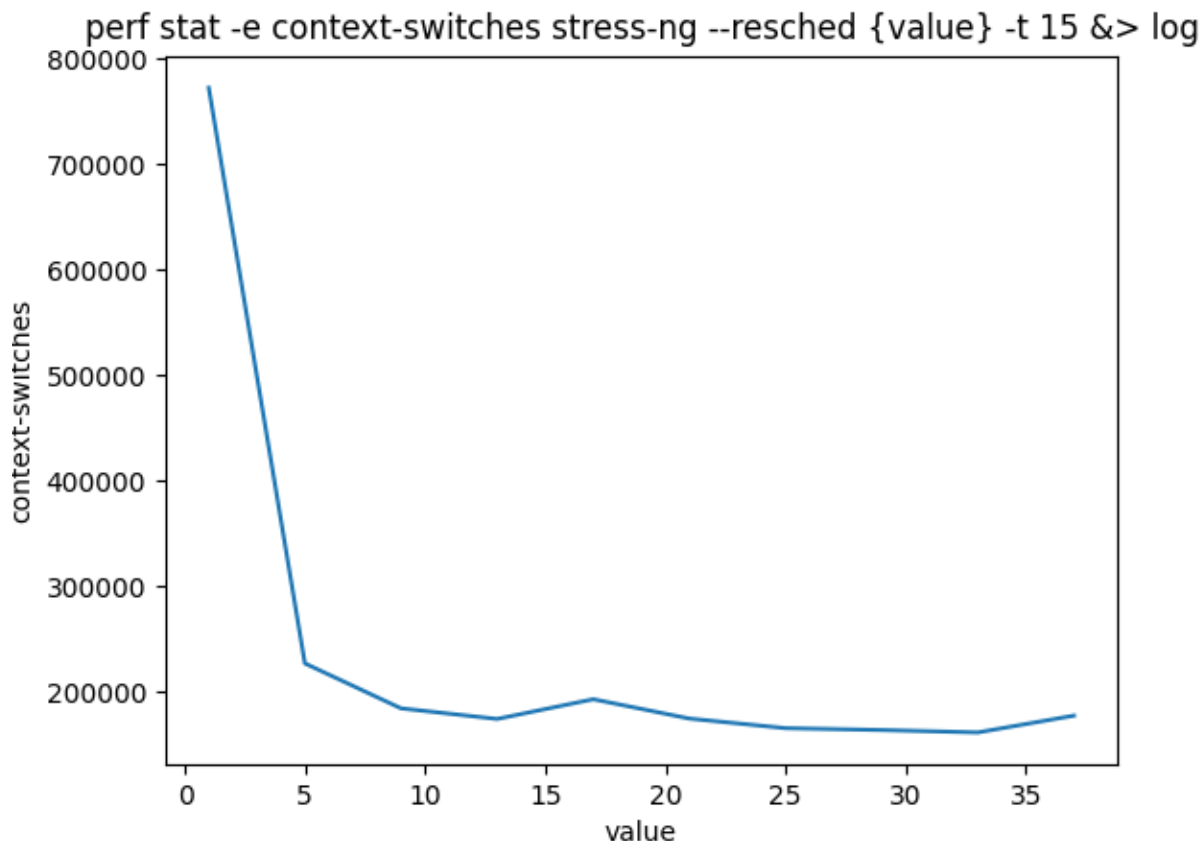
```

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:

[illegible]