

Федеральное государственное автономное образовательное учреждение высшего образования
"Национальный Исследовательский Университет ИТМО"
Мегафакультет Компьютерных Технологий и Управления
Факультет Программной Инженерии и Компьютерной Техники



Лабораторная работа №1
по дисциплине
'Операционные системы'

Выполнил Студент группы Р33102
Лапин Алексей Александрович
Преподаватель:
Осипов Святослав Владимирович

г. Санкт-Петербург
2023г.

Содержание

1 CPU Testing	3
1.1 fft	3
1.1.1 Команда lscpu	3
1.1.2 Команда lstopo	5
1.1.3 Команда pidstat	5
1.1.4 Команда mpstat	8
1.1.5 Команда atop	9
1.1.6 Команда stress-ng	10
1.1.7 Flame Graph	12
1.2 Переходим к активным действиям	12
1.2.1 Команда nice	12
1.2.2 Команда chrt	13
1.2.3 Финал	15
1.3 cdouble	15
1.4 Дополнительные команды для анализа, если интересно	20
1.4.1 turbostat	20
1.4.2 uptime	22
1.4.3 numastat	22
2 Cache Testing	23
2.1 l1cache	23
2.2 l1cache-sets	30
2.3 cache-fence	44
3 IO testing	47
3.1 ioport	47
3.2 io-uring	50
4 Memory testing	53
4.1 zlib	53
4.2 zlib-mem-level	54
4.3 fork-vm	56
5 Network testing	57
5.1 netlink-task	57
5.2 netdev	59
6 Pipe testing	60
6.1 pipe	61
6.2 pipe-ops	61
6.3 pipe-data-size	63
7 Sched testing	64
8 Вывод:	67

1 CPU Testing

Выданные параметры: [fft,cdouble]

Параметры процессора используемого в тесте:

Intel Core i5-8259U

Ядер: 4

Потоков: 8

Базовая частота: 2.30 ГГц

Кэш 1-го уровня: 64К (на ядро)

Кэш 2-го уровня: 256К (на ядро)

Кэш 3-го уровня: 6 Мб (всего)

Входе теста используется ОС Linux установленная на виртуальную машину через Parallels.

Ограничение по процессорам: 8

Ограничение по памяти: 2048 MB

1.1 fft

1.1.1 Команда lscpu

```
Architecture:           x86_64
CPU op-mode(s):        32-bit, 64-bit
Address sizes:         36 bits physical, 48 bits virtual
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:  0-7
Vendor ID:             GenuineIntel
Model name:            Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
CPU family:            6
Model:                 142
Thread(s) per core:   1
Core(s) per socket:   8
Socket(s):            1
Stepping:              10
BogoMIPS:              4608.00
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mc
                      a cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscal
                      l nx rdtscp lm constant_tsc nopl xtopology nonstop_tsc
                      cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid
                      sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer ae
                      s xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowpre
                      fetch invpcid_single pti fsgsbase tsc_adjust bmi1 avx2
                      smep bmi2 invpcid rdseed adx smap clflushopt xsaveopt x
                      savec dtherm arat pln pts
Virtualization features:
Hypervisor vendor:    KVM
Virtualization type:   full
Caches (sum of all):
L1d:                  256 KiB (8 instances)
```

```
L1i:          256 KiB (8 instances)
L2:          2 MiB (8 instances)
L3:          6 MiB (1 instance)

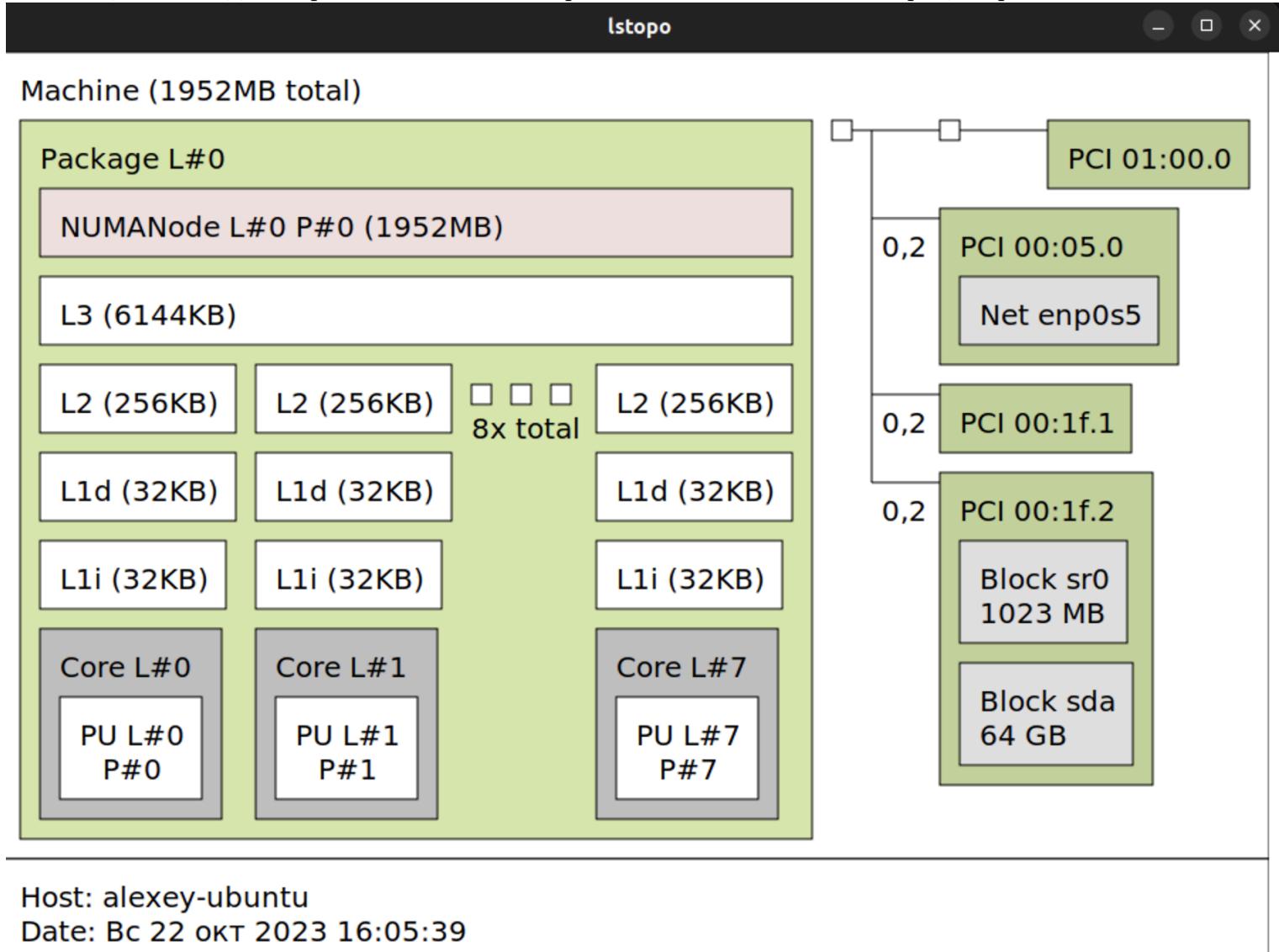
NUMA:
NUMA node(s):      1
NUMA node0 CPU(s): 0-7

Vulnerabilities:
Gather data sampling: Unknown: Dependent on hypervisor status
Itlb multihit:       KVM: Mitigation: VMX unsupported
L1tf:                Mitigation; PTE Inversion
Mds:                 Vulnerable: Clear CPU buffers attempted, no microcode;
                      SMT Host state unknown
Meltdown:            Mitigation; PTI
Mmio stale data:    Vulnerable: Clear CPU buffers attempted, no microcode;
                      SMT Host state unknown
Retbleed:            Vulnerable
Spec rstack overflow: Not affected
Spec store bypass:   Vulnerable
Spectre v1:          Mitigation; usercopy/swapgs barriers and __user pointer
                      sanitization
Spectre v2:          Mitigation; Retpolines, STIBP disabled, RSB filling, PB
                      RSB-eIBRS Not affected
Srbds:               Unknown: Dependent on hypervisor status
Tsx async abort:    Not affected
```

Видим, что Linux распознает наши 4 ядра по 2 потока, как 8 ядер по 1 потоку.

1.1.2 Команда lstopo

С помощью команды lstopo мы можем посмотреть на топологию нашего процессора:



1.1.3 Команда pidstat

Посмотрим статистику по процессу

Для этого запустим команду: `stress-ng -cpu 1 -cpu-method fft -metrics -timeout 120`

И посмотрим статистику запустив команду: `pidstat -p 58452 1`, где pid мы узнали спомощью команды `top`.

20:45:33	UID	PID	%usr	%system	%guest	%wait	%CPU	CPU	Command
20:45:34	1000	58452	100,00	0,00	0,00	0,00	100,00	3	stress-ng
20:45:35	1000	58452	100,00	0,00	0,00	0,00	100,00	3	stress-ng
20:45:36	1000	58452	100,00	0,00	0,00	0,00	100,00	3	stress-ng
20:45:37	1000	58452	100,00	0,00	0,00	0,00	100,00	3	stress-ng
20:45:38	1000	58452	100,00	0,00	0,00	0,00	100,00	3	stress-ng
20:45:39	1000	58452	99,01	0,00	0,00	0,00	99,01	3	stress-ng
20:45:40	1000	58452	100,00	0,00	0,00	0,00	100,00	3	stress-ng
20:45:41	1000	58452	101,00	0,00	0,00	0,00	101,00	3	stress-ng
20:45:42	1000	58452	100,00	0,00	0,00	0,00	100,00	3	stress-ng

```

20:45:43    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:44    1000    58452  99,01   0,00   0,00   0,00  99,01   3 stress-ng
20:45:45    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:46    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:47    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:48    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:49    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:50    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:51    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:52    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:53    1000    58452  101,00   0,00   0,00   0,00  101,00   3 stress-ng
20:45:54    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:55    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:56    1000    58452  99,01   0,00   0,00   0,00  99,01   3 stress-ng
20:45:57    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:58    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:45:59    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:46:00    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:46:01    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:46:02    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
20:46:03    1000    58452  100,00   0,00   0,00   0,00  100,00   3 stress-ng
^C
Average:   1000    58452  99,97   0,00   0,00   0,00  99,97   - stress-ng

```

Мы видим, что процесс выполнялся на процессоре номер 3 и занимал почти все его время(100%). При этом процесс выполнялся практически все время в режиме *пользователя*, что объяснимо так как fft - вычислительная задача.

Результат *stress-ng*:

```

1 alexey@alexey-ubuntu:~$ stress-ng --cpu 1 --cpu-method fft --metrics --timeout 120
3 stress-ng: info: [58451] setting to a 120 second (2 mins, 0.00 secs) run per stressor
5 stress-ng: info: [58451] dispatching hogs: 1 cpu
7 stress-ng: info: [58451] successful run completed in 120.00s (2 mins, 0.00 secs)
9 stress-ng: info: [58451] stressor bogo ops real time  usr time  sys time  bogo ops/s      bogo ops/s CPU used
     per
11 stress-ng: info: [58451]                               (secs)      (secs)      (secs)      (real time) (usr+sys time) instance
     (%)
13 stress-ng: info: [58451] cpu          102216    120.00    119.97      0.00      851.79      852.01
     99.97

```

Также посмотрим на приоритет и политику планирования процесса:

```

2 alexey@alexey-ubuntu:~$ pidstat -R -p 76043 1
4
Linux 6.2.0-35-generic (alexey-ubuntu) 22.10.2023 _x86_64_ (8 CPU)
6
22:28:20      UID      PID      prio policy Command

```

8	22:28:21	1000	76043	0	NORMAL	stress-ng
10	22:28:22	1000	76043	0	NORMAL	stress-ng
12	22:28:23	1000	76043	0	NORMAL	stress-ng
14	22:28:24	1000	76043	0	NORMAL	stress-ng
16	22:28:25	1000	76043	0	NORMAL	stress-ng
18	22:28:26	1000	76043	0	NORMAL	stress-ng
20	22:28:27	1000	76043	0	NORMAL	stress-ng
22	22:28:28	1000	76043	0	NORMAL	stress-ng
24	22:28:29	1000	76043	0	NORMAL	stress-ng
26	^C					
28	Average:	1000	76043	0	NORMAL	stress-ng

NORMAL: *SCHED_NORMAL* (ранее известный как *SCHED_OTHER*) — это алгоритм планирования с разделением времени. Используется по умолчанию для пользовательских процессов. Планировщик динамически корректирует приоритет в зависимости от класса планирования. Для $O(1)$ длительность кванта времени устанавливается на основе статического приоритета: чем выше приоритет, тем большие кванты времени. Для класса *CFS* размер кванта выбирается динамически. Использует класс планирования *CFS*.

1.1.4 Команда mpstat

Запустим команду *mpstat*, чтобы посмотреть загруженность всех процессоров.

Linux 6.2.0-35-generic (alexey-ubuntu) 22.10.2023 _x86_64_ (8 CPU)											
23:00:29	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
23:00:30	all	13,50	0,00	0,25	0,00	0,00	0,00	0,00	0,00	0,00	86,25
23:00:30	0	1,00	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	98,00
23:00:30	1	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00
23:00:30	2	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00
23:00:30	3	1,98	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,02
23:00:30	4	100,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
23:00:30	5	1,01	0,00	1,01	0,00	0,00	0,00	0,00	0,00	0,00	97,98
23:00:30	6	2,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,00
23:00:30	7	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	100,00
23:00:30	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
23:00:31	all	13,22	0,00	0,50	0,00	0,00	0,12	0,00	0,00	0,00	86,16
23:00:31	0	1,96	0,00	0,98	0,00	0,00	0,98	0,00	0,00	0,00	96,08
23:00:31	1	0,00	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00
23:00:31	2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	100,00
23:00:31	3	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00
23:00:31	4	100,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
23:00:31	5	1,00	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	98,00
23:00:31	6	0,00	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00
23:00:31	7	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00
23:00:31	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle
23:00:32	all	13,17	0,00	0,50	0,00	0,00	0,00	0,00	0,00	0,00	86,32
23:00:32	0	0,99	0,00	1,98	0,00	0,00	0,00	0,00	0,00	0,00	97,03
23:00:32	1	0,00	0,00	1,01	0,00	0,00	0,00	0,00	0,00	0,00	98,99
23:00:32	2	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	100,00
23:00:32	3	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00
23:00:32	4	100,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
23:00:32	5	1,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,99
23:00:32	6	1,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,99
23:00:32	7	1,00	0,00	1,00	0,00	0,00	0,00	0,00	0,00	0,00	98,00

Видим, что *stress-ng* загружает только один процессор на 100% (в данном случае процессор 4). Такое поведение и ожидалось так как мы запустили *stress-ng* с параметром *-cpu 1*. Для следующих запусков определим командой *taskset* процессор на котором будет выполняться *stress-ng*.

alexey@alexey-ubuntu:~\$ taskset -c 0 stress-ng --cpu 1 --cpu-method fft --metrics --timeout 120
stress-ng: info: [83847] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [83847] dispatching hogs: 1 cpu
stress-ng: info: [83847] successful run completed in 120.00s (2 mins, 0.00 secs)
stress-ng: info: [83847] stressor bogo ops real time usr time sys time bogo ops/s bogo ops/s CPU used per
stress-ng: info: [83847] (secs) (secs) (secs) (secs) (real time) (usr+sys time) instance (%)
stress-ng: info: [83847] cpu 110118 120.00 119.96 0.00 917.65 917.96 99.97
alexey@alexey-ubuntu:~\$ taskset -c 0 stress-ng --cpu 1 --cpu-method fft --metrics --timeout 120
stress-ng: info: [84662] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [84662] dispatching hogs: 1 cpu
stress-ng: info: [84662] successful run completed in 120.00s (2 mins, 0.00 secs)
stress-ng: info: [84662] stressor bogo ops real time usr time sys time bogo ops/s bogo ops/s CPU used per
stress-ng: info: [84662] (secs) (secs) (secs) (secs) (real time) (usr+sys time) instance (%)
stress-ng: info: [84662] cpu 111239 120.00 119.96 0.00 926.99 927.30 99.97

Как мы можем заметить, производительность немного подросла, особенно это видно при повторном запуске. Это следует из того, что при использовании одного и того же процессора кэши остаются горячими.

alexey@alexey-ubuntu:~\$ mpstat -P ALL 1												
Linux 6.2.0-35-generic (alexey-ubuntu) 22.10.2023 _x86_64_ (8 CPU)												
	CPU	%usr	%nice	%sys	%iowait	%irq	%soft	%steal	%guest	%gnice	%idle	
23:13:41	all	13,84	0,00	0,12	0,00	0,00	0,12	0,00	0,00	0,00	85,91	
23:13:42	0	100,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	
23:13:42	1	2,04	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	97,96	
23:13:42	2	0,98	0,00	0,98	0,00	0,00	0,98	0,00	0,00	0,00	97,06	
23:13:42	3	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00	
23:13:42	4	0,99	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,01	
23:13:42	5	1,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	99,00	
23:13:42	6	2,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,00	
23:13:42	7	2,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,00	
23:13:42	all	13,25	0,00	0,50	0,00	0,00	0,00	0,00	0,00	0,00	86,25	
23:13:43	0	100,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	
23:13:43	1	2,02	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	97,98	
23:13:43	2	0,99	0,00	0,99	0,00	0,00	0,00	0,00	0,00	0,00	98,02	
23:13:43	3	0,99	0,00	0,99	0,00	0,00	0,00	0,00	0,00	0,00	98,02	
23:13:43	4	1,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,99	
23:13:43	5	0,00	0,00	1,98	0,00	0,00	0,00	0,00	0,00	0,00	98,02	
23:13:43	6	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	100,00	
23:13:43	7	1,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	98,99	

1.1.5 Команда atop

С помощью команды *atop*, которая является более специфичным для Linux аналогом команды *top*, мы посмотрим общую загруженность системы.

ATOP - alexey-ubuntu 2023/10/22 23:35:09 10s elapsed																			
PRC sys	0.37s	user	10.56s	#proc	263	#trun	2	#tslpi	480	#tslpu	76	#zombie	0	clones	33	#exit	30		
CPU sys	4%	user	106%	irq	0%	idle	690%	wait	0%	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	0%	user	100%	irq	0%	idle	0%	cpu000	w	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	1%	user	1%	irq	0%	idle	98%	cpu005	w	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	1%	user	1%	irq	0%	idle	98%	cpu006	w	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	0%	user	2%	irq	0%	idle	98%	cpu007	w	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	1%	user	1%	irq	0%	idle	98%	cpu001	w	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	0%	user	1%	irq	0%	idle	99%	cpu004	w	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	1%	user	0%	irq	0%	idle	99%	cpu003	w	steal	0%	guest	0%	curf	2.30GHz				
cpu sys	1%	user	0%	irq	0%	idle	99%	cpu002	w	steal	0%	guest	0%	curf	2.30GHz				
CPL avg1	0.47	avg5	0.30	avg15	0.27	csw	7563	intr	8585	slab	177.9M	slrec	68.0M	shmem	135.9M	shrss	0.0M	numnode	1
MEM tot	1.9G	free	145.6M	cache	850.3M	dirty	0.2M	buff	20.9M	slab	177.9M	slrec	68.0M	shmem	135.9M	shrss	0.0M	numnode	1
SWP tot	5.9G	free	5.8G	swcac	7.2M	vmcom	4.0G	vmlim	6.9G	MBw/s	0.0	MBr/s	0.0	MBw/s	0.0	avio	0.48 ms	tcpie	0
DSK sda	busy	0%	read	5	write	53	discrd	0	KiB/r	4	KiB/w	5	MBr/s	0.0	icmpli	0	icmpon	0	
NET transport	tcpip	0	tcpo	0	udpi	7	udp0	7	tcpao	0	tcppo	0	tcpers	0	drpi	0	drpo	0	
NET network	ip1	7	ipo	7	ipfrw	0	deliv	7	so	0 Kbps	erri	0	erro	0	drpi	0	drpo	0	
NET lo	---	pck1	4	pcko	4	sp	0 Mbps	si	0 Kbps	so	0 Kbps	erri	0	erro	0	drpt	0	drpo	0
NET empos5	---	pck1	3	pcko	3	sp	0 Mbps	si	0 Kbps	so	0 Kbps	erri	0	erro	0	drpt	0	drpo	0
PID	SYSCPU	USRCPU	RDELAY	VGROW	RGROW	RUID	EUID	ST	EXC	THR	S	CPUUNR	CPU	CMD	1/4				
87336	0.00s	10.00s	0.00s	0B	0B	alexey	alexey	--	-	1	R	0	100%	stress-ng					
51920	0.05s	0.20s	0.00s	0B	0B	alexey	alexey	--	-	16	S	5	3%	gnome-shell					
83305	0.03s	0.08s	0.00s	0B	0B	alexey	alexey	--	-	12	S	5	1%	gjs					
52849	0.01s	0.09s	0.00s	0B	0B	alexey	alexey	--	-	4	S	7	1%	gnome-terminal					
52245	0.00s	0.08s	0.00s	0B	0B	alexey	alexey	--	-	8	S	7	1%	prlcc					
86739	0.04s	0.01s	0.00s	0B	0B	alexey	alexey	--	-	1	R	1	1%	atop					
614	0.02s	0.03s	0.00s	0B	0B	messageb	messageb	--	-	1	S	4	1%	dbus-daemon					
52261	0.03s	0.00s	0.00s	0B	0B	alexey	alexey	--	-	3	S	0	0%	prldnd					
1727	0.01s	0.01s	0.00s	0B	0B	geoclue	geoclue	--	-	3	S	2	0%	geoclue					
777	0.01s	0.01s	0.00s	0B	0B	root	root	--	-	1	S	3	0%	cupsd					
53713	0.02s	0.00s	0.00s	0B	0B	alexey	alexey	--	-	1	S	6	0%	top					
15	0.02s	0.00s	0.00s	0B	0B	root	root	--	-	1	I	2	0%	rcu_preempt					
87371	0.02s	0.00s	-	0K	0K	root	root	--	NE	0	E	-	0%	<lpstat>					
249	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	S	5	0%	systemd-journa					
52263	0.01s	0.00s	0.00s	0B	0B	alexey	alexey	--	-	7	S	4	0%	prlsga					
1123	0.00s	0.01s	0.00s	0B	0B	root	root	--	-	3	S	1	0%	cups-browsed					
51715	0.01s	0.00s	0.00s	0B	0B	alexey	alexey	--	-	1	S	7	0%	dbus-daemon					
401	0.01s	0.00s	0.00s	0B	0B	systemd-	systemd-	--	-	1	S	6	0%	systemd-oomd					
606	0.00s	0.01s	0.00s	0B	0B	avahi	avahi	--	-	1	S	4	0%	avahi-daemon					
1086	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	3	S	6	0%	prlshprint					
40	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	S	4	0%	migration/4					
66942	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	I	3	0%	kworker/u64:2-					

Отсюда мы делаем вывод, что кроме нашего процесса самое высокое потребление у *gnome-shell*, но пока мы рассматриваем нагрузку только на одно ядро для нас это неважно, потому что *gnome-shell* будет выполняться на другом ядре.

ATOP - alexey-ubuntu															10s elapsed	
2023/10/22 23:57:49																
PRC	sys	0.22s	user	10.08s	#proc	192	#trun	2	#tslpi	204	#tslpu	76	#zombie	0	#exit	27
CPU	sys	3%	user	95%	irq	0%	idle	702%	wait	0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	0%	user	100%	irq	0%	idle	0%	cpu000	w 0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	1%	user	0%	irq	0%	idle	99%	cpu005	w 0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	1%	user	0%	irq	0%	idle	99%	cpu006	w 0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	0%	user	0%	irq	0%	idle	99%	cpu003	w 0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	0%	user	0%	irq	0%	idle	99%	cpu002	w 0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	0%	user	0%	irq	0%	idle	100%	cpu001	w 0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	0%	user	0%	irq	0%	idle	100%	cpu004	w 0%	steal	0%	guest	0%	curf	2.30GHz
cpu	sys	0%	user	0%	irq	0%	idle	100%	cpu007	w 0%	steal	0%	guest	0%	curf	2.30GHz
CPL	avg1	0.55	avg5	0.71	avg15	0.45	csu	3240	intr	5347			numcpu	8		
MEM	tot	1.9G	free	1.06	cache	513.4M	dirty	0.2M	buff	34.5M	slab	126.2M	shrss	0.0M	numnode	1
SWP	tot	5.9G	free	5.9G	swcac	0.0M							vmcom	1.3G	vm11m	6.9G
PSI	cpusome	0%	memsome	0%	memfull	0%	iosome	0%	iofull	0%	cs	0/1/10	ms	0/0/0	is	0/0/0
DSK	sda	busy	0%	read	0	write	3	discrd	0	MBr/s	0.0	MBw/s	0.0	avio	2.67 ms	
NET	transport		tcp1	0	tcpo	0	udpi	1	udpo	1	tcpao	0	tcpoo	0	tcprs	0
NET	network		ipi	1	ipo	1	ipfrw	0	deliv	1			icmpl	0	icmpo	0
NET	enp0s5	----	pcki	2	pcko	2	sp	0 Mbps	si	0 Kbps	so	0 Kbps	erri	0	erro	0
PID	SYSCPU	USRCPU	RDELAY	VGROW	RGROW	RUID	EUID	ST	EXC	THR	S	CPUUNR	CPU	CMD	1/3	
3783	0.00s	10.00s	0.00s	0B	0B	alexey	alexey	--	-	1	R	0	100%	stress-ng		
3796	0.03s	0.02s	0.01s	6.5M	2.9M	alexey	alexey	--	-	1	R	6	1%	atop		
620	0.03s	0.02s	0.00s	0B	0B	messageb	messageb	--	-	1	S	6	1%	dbus-daemon		
441	0.02s	0.00s	0.00s	0B	0B	systemd-	systemd-	--	-	1	S	7	0%	systemd-oomd		
613	0.01s	0.01s	0.00s	0B	0B	avahi	avahi	--	-	1	S	2	0%	avahi-daemon		
295	0.02s	0.00s	0.00s	0B	0B	root	root	--	-	1	I	5	0%	kworker/5:2-ev		
3802	0.01s	0.01s	-	OK	OK	root	-	NE	0	0	E	-	0%	<lpstat>		
807	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	S	7	0%	cupsd		
1254	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	3	S	5	0%	cups-browsed		
640	0.00s	0.01s	0.00s	0B	0B	root	root	--	-	3	S	6	0%	polkitd		
1303	0.01s	0.00s	0.00s	0B	0B	rtkit	rtkit	--	-	3	S	0	0%	rtkit-daemon		
930	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	4	S	1	0%	prltoolsd		
616	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	S	5	0%	atopaccctl		
15	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	I	7	0%	rcu_preempt		
73	0.00s	0.01s	0.00s	0B	0B	root	root	--	-	1	S	7	0%	Kcompactd0		
98	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	I	4	0%	kworker/4:1-ev		
3205	0.01s	0.00s	0.00s	0B	0B	root	root	--	-	1	I	6	0%	kworker/6:0-ev		
3805	0.01s	0.00s	-	OK	OK	root	-	NE	0	0	E	-	0%	<lpstat>		
3814	0.01s	0.00s	-	OK	OK	root	-	NE	0	0	E	-	0%	<lpstat>		
2005	0.00s	0.00s	0.00s	0B	0B	alexey	alexey	--	-	6	S	2	0%	tracker-miner-		
623	0.00s	0.00s	0.00s	0B	0B	root	root	--	-	3	S	4	0%	NetworkManager		
458	0.00s	0.00s	0.00s	0B	0B	systemd-	systemd-	--	-	2	S	4	0%	systemd-timesy		
997	0.00s	0.00s	0.00s	0B	0B	root	root	--	-	3	S	5	0%	prlshprint		
1060	0.00s	0.00s	0.00s	0B	0B	kernoops	kernoops	--	-	1	S	4	0%	kerneloops		
600	0.00s	0.00s	0.00s	0B	0B	root	root	--	-	1	S	4	0%	acpid		
9	0.00s	0.00s	0.00s	0B	0B	root	root	--	-	1	I	3	0%	kworker/u64:0-		
34	0.00s	0.00s	0.00s	0B	0B	root	root	--	-	1	S	3	0%	migration/3		

Запустили без графической оболочки.

Как и ожидалось прироста производительности нет, пока мы рассматриваем тесты для одного ядра.

```
taskset -c 0 stress-ng --cpu 1 --cpu-method fft --metrics --timeout 120
stress-ng: info: [4312] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [4312] dispatching hogs: 1 cpu
stress-ng: info: [4312] successful run completed in 120.00s (2 mins, 0.00 secs)
stress-ng: info: [4312] stressor      bogo ops real time   usr time   sys time   bogo ops/s   bogo ops/s CPU used per
stress-ng: info: [4312]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%) 
stress-ng: info: [4312]  cpu          108802    120.00     0.00    906.68    906.68    100.00
[1]+ Done taskset -c 0 stress-ng --cpu 1 --cpu-method fft --metrics --timeout 120
```

1.1.6 Команда stress-ng

Воспользуемся встроенным параметрами *stress-ng*.

С помощью опции *-times* можно получить представление о том, сколько пользовательского и системного (ядро) времени используется.

Мы можем запустить *stress-ng* с опцией *-perf*, чтобы увидеть более подробную информацию о том, что делает машина во время работы

```

1 alexey@alexey-ubuntu:~$ taskset -c 0 sudo stress-ng --cpu 1 --cpu-method fft --metrics --timeout 120 --times --perf
[sudo] password for alexey:
3 stress-ng: info: [3347] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [3347] dispatching hogs: 1 cpu
5 stress-ng: info: [3347] successful run completed in 121.46s (2 mins, 1.46 secs)
stress-ng: info: [3347] stressor      bogo ops real time   usr time   sys time   bogo ops/s      bogo ops/s CPU
     used per
7 stress-ng: info: [3347]                               (secs)    (secs)    (secs)    (real time) (usr+sys time)
     instance (%) 
stress-ng: info: [3347]  cpu          109925    121.46    119.88    0.05      905.06      916.58
     98.74
9 stress-ng: info: [3347]  cpu:
stress-ng: info: [3347]    119 873 637 761 CPU Clock           0.99 B/sec
11 stress-ng: info: [3347]    119 867 619 227 Task Clock           0.99 B/sec
stress-ng: info: [3347]    44 Page Faults Total           0,36 /sec
13 stress-ng: info: [3347]    44 Page Faults Minor           0,36 /sec
stress-ng: info: [3347]    0 Page Faults Major           0,00 /sec
15 stress-ng: info: [3347]    763 Context Switches          6,28 /sec
stress-ng: info: [3347]    761 Cgroup Switches          6,27 /sec
17 stress-ng: info: [3347]    0 CPU Migrations           0,00 /sec
stress-ng: info: [3347]    0 Alignment Faults          0,00 /sec
19 stress-ng: info: [3347]    0 Emulation Faults          0,00 /sec
stress-ng: info: [3347]    43 Page Faults User           0,35 /sec
21 stress-ng: info: [3347]    1 Page Faults Kernel          0,01 /sec
stress-ng: info: [3347]    92 System Call Enter          0,76 /sec
23 stress-ng: info: [3347]    91 System Call Exit           0,75 /sec
stress-ng: info: [3347]    1 Kmalloc           0,01 /sec
25 stress-ng: info: [3347]    1 Kfree             0,01 /sec
stress-ng: info: [3347]    3 Kmem Cache Alloc          0,02 /sec
27 stress-ng: info: [3347]    2 Kmem Cache Free           0,02 /sec
stress-ng: info: [3347]    35 MM Page Alloc           0,29 /sec
29 stress-ng: info: [3347]    0 MM Page Free            0,00 /sec
stress-ng: info: [3347]    79.202 RCU Utilization        652.10 /sec
31 stress-ng: info: [3347]    9 Sched Migrate Task          0,07 /sec
stress-ng: info: [3347]    0 Sched Move NUMA           0,00 /sec
33 stress-ng: info: [3347]    765 Sched Wakeup           6,30 /sec
stress-ng: info: [3347]    0 Sched Proc Exec           0,00 /sec
35 stress-ng: info: [3347]    0 Sched Proc Exit           0,00 /sec
stress-ng: info: [3347]    0 Sched Proc Fork           0,00 /sec
37 stress-ng: info: [3347]    0 Sched Proc Free            0,00 /sec
stress-ng: info: [3347]    0 Sched Proc Hang           0,00 /sec
39 stress-ng: info: [3347]    0 Sched Proc Wait           0,00 /sec
stress-ng: info: [3347]    763 Sched Switch           6,28 /sec
41 stress-ng: info: [3347]    2 Signal Generate          0,02 /sec
stress-ng: info: [3347]    1 Signal Deliver           0,01 /sec
43 stress-ng: info: [3347]    32 IRQ Entry             0,26 /sec
stress-ng: info: [3347]    32 IRQ Exit              0,26 /sec
45 stress-ng: info: [3347]    11.185 Soft IRQ Entry         92.09 /sec
stress-ng: info: [3347]    11.185 Soft IRQ Exit          92.09 /sec
47 stress-ng: info: [3347]    0 NMI handler           0,00 /sec
stress-ng: info: [3347]    1 Writeback Dirty Inode        0,01 /sec
49 stress-ng: info: [3347]    0 Migrate MM Pages          0,00 /sec
stress-ng: info: [3347]    1 SKB Consume           0,01 /sec
51 stress-ng: info: [3347]    0 SKB Kfree             0,00 /sec
stress-ng: info: [3347]    0 IOMMU IO Page Fault        0,00 /sec
53 stress-ng: info: [3347]    0 IOMMU Map              0,00 /sec
stress-ng: info: [3347]    0 IOMMU Unmap           0,00 /sec
55 stress-ng: info: [3347]    0 Filemap page-cache add       0,00 /sec
stress-ng: info: [3347]    0 Filemap page-cache del       0,00 /sec
57 stress-ng: info: [3347]    0 OOM Compact Retry          0,00 /sec
stress-ng: info: [3347]    0 OOM Wake Reaper          0,00 /sec
59 stress-ng: info: [3347]    0 Thermal Zone Trip          0,00 /sec
stress-ng: info: [3347] for a 121,46s run time:
61 stress-ng: info: [3347]    971,66s available CPU time

```

```

63 stress-ng: info: [3347]      119,88s user time   ( 12,34%)
stress-ng: info: [3347]       0,05s system time  (  0,01%)
stress-ng: info: [3347]     119,93s total time   ( 12,34%)
65 stress-ng: info: [3347] load average: 1,09 0,69 0,30

```

1.1.7 Flame Graph

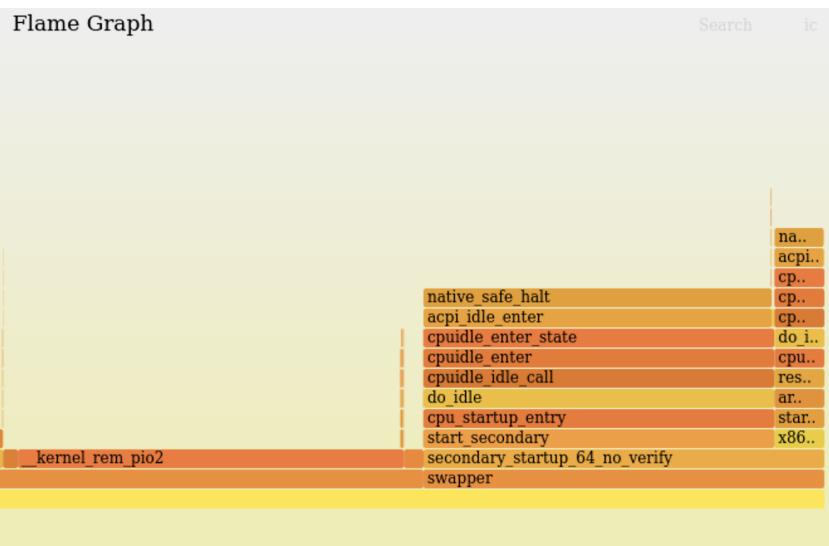
Выполняем профилирование в течении 10 секунд.

```

alexey@alexey-ubuntu:~$ sudo perf record -a -g -F 99 -- sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 2,175 MB perf.data (7907 samples) ]

```

Строим по выводу Flame Graph.



1.2 Переходим к активным действиям

1.2.1 Команда nice

Повысим приоритет нашего процесса с помощью команды *nice*.

Чтобы была конкуренция за ресурс процессора, мы запустим тест на все ядра (8 ядер)

```

alexey@alexey-ubuntu:~$ sudo stress-ng --cpu 0 --cpu-method fft --metrics -t 2m
stress-ng: info: [14672] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [14672] dispatching hogs: 8 cpu
stress-ng: info: [14672] successful run completed in 120.00s (2 mins, 0.00 secs)
stress-ng: info: [14672] stressor      bogo ops real time  usr time  sys time  bogo ops/s    bogo ops/s CPU used per
stress-ng: info: [14672]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [14672] cpu          417426    120.00    940.19    3.97    3478.51      442.11      98.35
alexey@alexey-ubuntu:~$ nice -n 19 sudo stress-ng --cpu 0 --cpu-method fft --metrics -t 2m
[sudo] password for alexey:
stress-ng: info: [17281] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [17281] dispatching hogs: 8 cpu
stress-ng: info: [17281] successful run completed in 120.01s (2 mins, 0.01 secs)
stress-ng: info: [17281] stressor      bogo ops real time  usr time  sys time  bogo ops/s    bogo ops/s CPU used per
stress-ng: info: [17281]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [17281] cpu          443405    120.00    942.91    3.12    3695.01      468.70      98.54

```

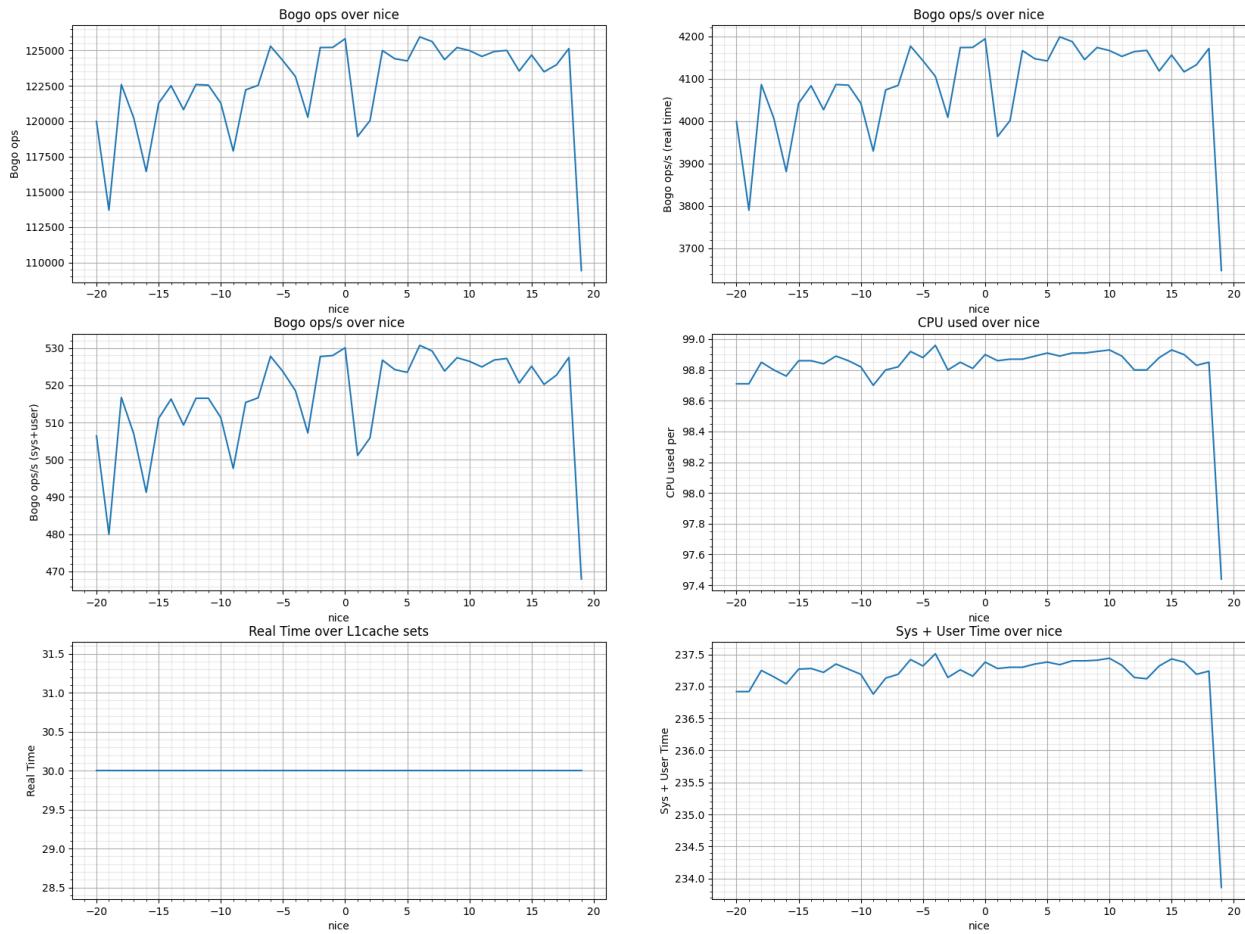
Напишем скрипт, который протестирует программу с разными *nice*.

```

1#!/bin/zsh
echo "nice;bogo_ops;real_time;usr_time;sys_time;bogo_ops/s_real;bogo_ops/s_user+sys;CPU_used_per"
3 for i in {-20..19..1}; do
    sudo nice -n $i stress-ng --cpu 0 --cpu-method fft --metrics -t 2m --stdout |
5 awk -v var="$i" '/[\[\]\d]+[ ]+cpu/{printf var;"$5";"$6";"$7";"$8";"$9";"$10";"$11"\n}'

```

done



1.2.2 Команда chrt

Попробуем разные политики планирования для наших процессов.

RR (Round Robin): *SCHED_RR* — это алгоритм планирования циклическим перебором. Как только поток исчерпает свой квант времени, он перемещается в конец очереди на выполнение для своего уровня приоритета. Это позволяет запускать другие потоки с таким же приоритетом. Использует класс планирования *RT*.

```
alexey@alexey-ubuntu:~$ sudo chrt -g 99 nice -n 19 sudo stress-ng --cpu 0 --cpu-method fft --metrics -t 2m
stress-ng: info: [24669] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [24669] dispatching hogs: 8 cpu

stress-ng: info: [24669] successful run completed in 120.01s (2 mins, 0.01 secs)
stress-ng: info: [24669] stressor      bogo ops real time  usr time  sys time  bogo ops/s      bogo ops/s CPU used per
stress-ng: info: [24669]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [24669] cpu          434915   120.00   912.34     0.01    3624.27        476.70      95.04
```

FIFO (First In, First Out): *SCHED_FIFO* — это алгоритм планирования «первым пришел, первым вышел», который продолжает выполнять поток, находящийся в голове очереди на

выполнение, пока тот не оставит процессор добровольно или не появится поток с более высоким приоритетом. Поток продолжает выполняться, даже если в очереди на выполнение есть другие потоки с таким же приоритетом. Использует класс планирования RT.

```
alexey@alexey-ubuntu:~$ sudo chrt -f 99 sudo stress-ng --cpu 0 --cpu-method fft --metrics -t 2m
[sudo] password for alexey:
stress-ng: info: [31099] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [31099] dispatching hogs: 8 cpu
stress-ng: info: [31099] successful run completed in 120.00s (2 mins, 0.00 secs)
stress-ng: info: [31099] stressor      bogo ops real time  usr time  sys time  bogo ops/s      bogo ops/s CPU used per
stress-ng: info: [31099]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [31099]   cpu          430410    120.00    911.94     0.25      3586.73      471.84      95.02
```

NORMAL: *SCHED_NORMAL* (ранее известный как *SCHED_OTHER*) – это алгоритм планирования с разделением времени. Используется по умолчанию для пользовательских процессов. Планировщик динамически корректирует приоритет в зависимости от класса планирования. Для $O(1)$ длительность кванта времени устанавливается на основе статического приоритета: чем выше приоритет, тем больше квант времени. Для класса *CFS* размер кванта выбирается динамически. Использует класс планирования *CFS*.

```
alexey@alexey-ubuntu:~$ sudo chrt -o 0 sudo stress-ng --cpu 0 --cpu-method fft --metrics -t 2m
stress-ng: info: [26934] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [26934] dispatching hogs: 8 cpu
stress-ng: info: [26934] successful run completed in 120.01s (2 mins, 0.01 secs)
stress-ng: info: [26934] stressor      bogo ops real time  usr time  sys time  bogo ops/s      bogo ops/s CPU used per
stress-ng: info: [26934]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [26934]   cpu          424715    120.00    939.58     4.19      3539.24      450.02      98.31
```

BATCH: *SCHED_BATCH* – этот алгоритм действует как *SCHED_NORMAL*, но ожидается, что поток будет привязан к процессору и не должен прерывать другие интерактивные задачи, связанные с вводом/выводом. Использует класс планирования *CFS*.

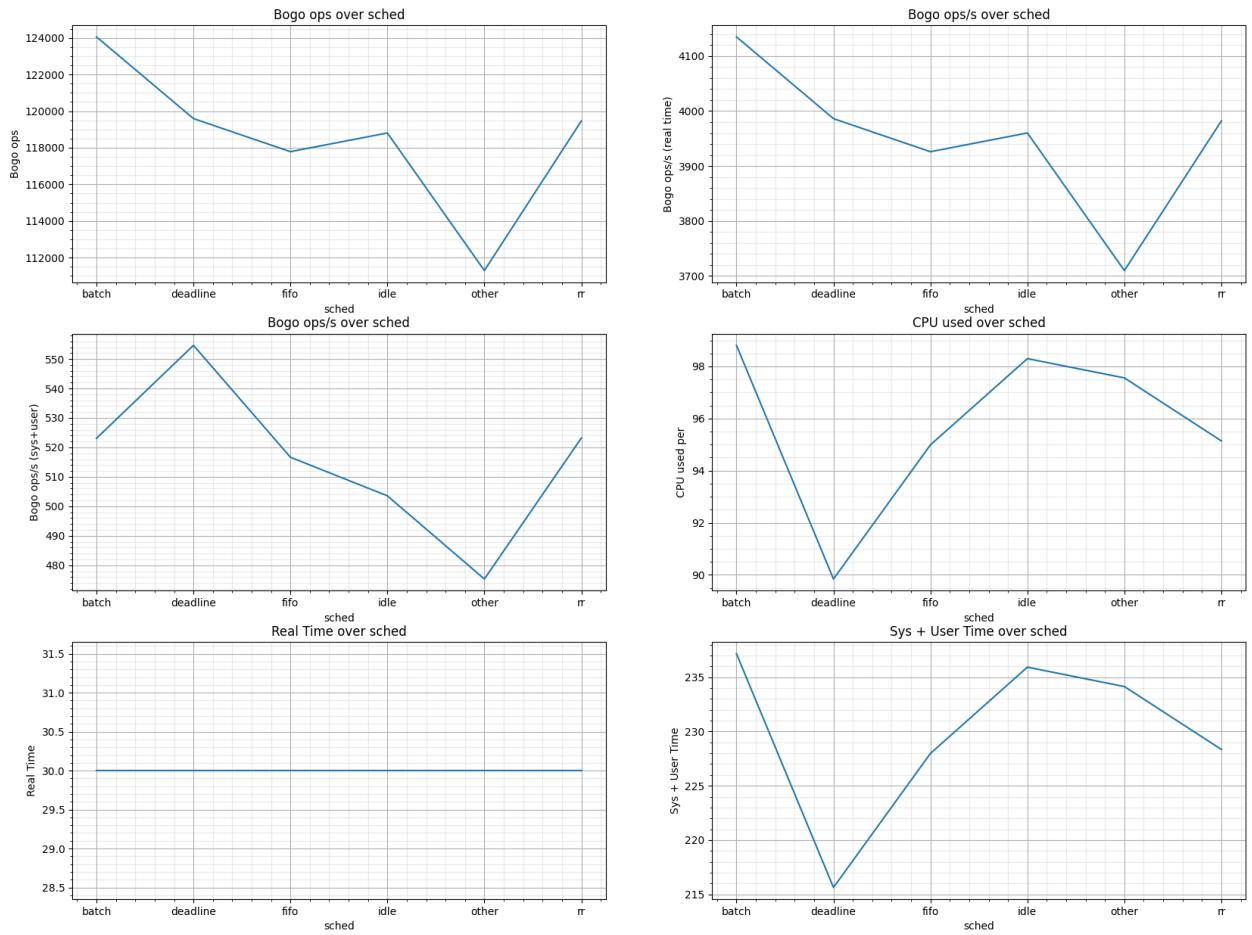
```
alexey@alexey-ubuntu:~$ sudo chrt -b 0 nice -n 19 sudo stress-ng --cpu 0 --cpu-method fft --metrics -t 2m
stress-ng: info: [25315] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [25315] dispatching hogs: 8 cpu
stress-ng: info: [25315] successful run completed in 120.01s (2 mins, 0.01 secs)
stress-ng: info: [25315] stressor      bogo ops real time  usr time  sys time  bogo ops/s      bogo ops/s CPU used per
stress-ng: info: [25315]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [25315]   cpu          441249    120.00    943.89     2.96      3677.02      466.02      98.63
```

IDLE::: *SCHED_IDLE* использует класс планирования *Idle*.

```
alexey@alexey-ubuntu:~$ sudo chrt -i 0 sudo stress-ng --cpu 0 --cpu-method fft --metrics -t 2m
stress-ng: info: [26563] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [26563] dispatching hogs: 8 cpu
stress-ng: info: [26563] successful run completed in 120.01s (2 mins, 0.01 secs)
stress-ng: info: [26563] stressor      bogo ops real time  usr time  sys time  bogo ops/s      bogo ops/s CPU used per
stress-ng: info: [26563]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [26563]   cpu          431462    120.00    935.41     5.60      3595.42      458.51      98.02
```

Напишем скрипт:

```
#!/bin/zsh
2 echo "sched;bogo_ops;real_time;usr_time;sys_time;bogo_ops/s_real;bogo_ops/s_user+sys;CPU_used_per"
3 schedulers=("batch" "deadline" "fifo" "idle" "other" "rr")
4 for sched in "${schedulers[@]}"; do
5     sudo nice -n -20 stress-ng --cpu 0 --cpu-method fft --metrics -t 30 --stdout --sched $sched |
6     awk -v var="$sched" '/[\[]\d]+[ ]+cpu/{printf var;"$5";"$6";"$7";"$8";"$9";"$10";"$11"\n}', done
```



Как мы видим, выбор политики планирования не сильно влияет на производительность нашего теста. Все значения в рамках погрешности.

1.2.3 Финал

Отключаем графический интерфейс и применяем все настройки выше.

```
~ sudo nice -n -20 stress-ng --cpu 0 --cpu-method fft --metrics -t 2m --stdout --sched batch
stress-ng: info: [3869] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [3869] dispatching hogs: 8 cpu
stress-ng: info: [3869] successful run completed in 120.00s (2 mins, 0.00 secs)
stress-ng: info: [3869] stressor      bogo ops real time  usr time  sys time   bogo ops/s    bogo ops/s CPU used per
stress-ng: info: [3869]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [3869] cpu           503563     120.00    955.42      0.72        4196.32      526.66    99.60
```

Итого мы с 417426 bogo ops смогли повысить производительность до 503563 bogo ops.

1.3 cdouble

Так как *cdouble* тоже вычислительная задача, то попробуем применить все настройки выше. Но сначала тест без настроек.

```

alexey@alexey-ubuntu:~$ stress-ng --cpu 0 --cpu-method cdouble --metrics -t 2m
stress-ng: info: [16163] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [16163] dispatching hogs: 8 cpu
stress-ng: info: [16163] successful run completed in 120.11s (2 mins, 0.11 secs)
stress-ng: info: [16163] stressor      bogo ops real time   usr time   sys time   bogo ops/s   bogo ops/s CPU used per
stress-ng: info: [16163]           (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [16163]  cpu          1844831   120.01    914.30     7.99    15372.70    2000.27    96.07

```

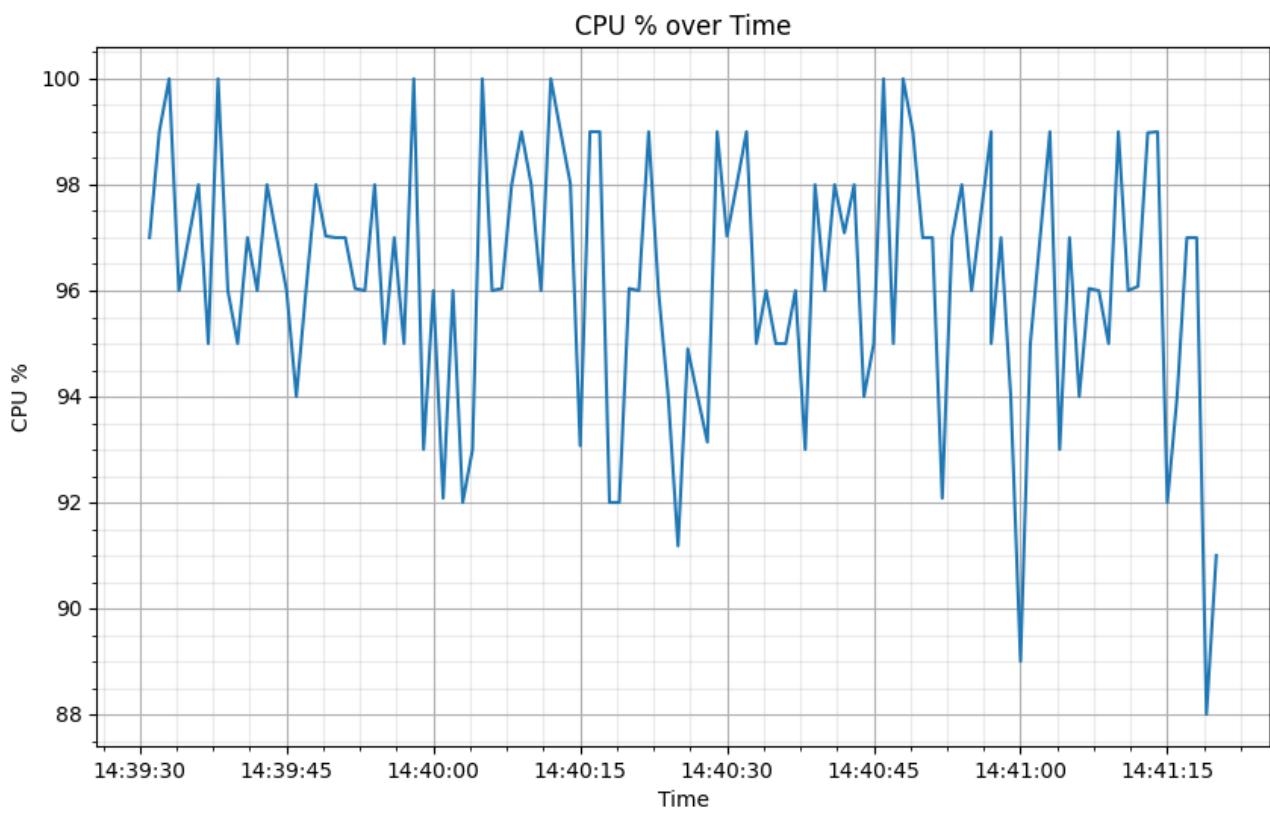
Посмотрим потребление процессора.

	UID	PID	%usr	%system	%guest	%wait	%CPU	CPU	Command
14:39:30									
14:39:31	1000	16167	96,00	1,00	0,00	4,00	97,00	0	stress-ng
14:39:32	1000	16167	99,01	0,00	0,00	0,99	99,01	0	stress-ng
14:39:33	1000	16167	100,00	0,00	0,00	0,00	100,00	0	stress-ng
14:39:34	1000	16167	96,00	0,00	0,00	2,00	96,00	0	stress-ng
14:39:35	1000	16167	96,00	1,00	0,00	4,00	97,00	0	stress-ng
14:39:36	1000	16167	97,00	1,00	0,00	3,00	98,00	0	stress-ng
14:39:37	1000	16167	95,00	0,00	0,00	4,00	95,00	0	stress-ng
14:39:38	1000	16167	99,00	1,00	0,00	1,00	100,00	0	stress-ng
14:39:39	1000	16167	95,00	1,00	0,00	4,00	96,00	5	stress-ng
14:39:40	1000	16167	95,00	0,00	0,00	5,00	95,00	5	stress-ng
14:39:41	1000	16167	97,00	0,00	0,00	3,00	97,00	5	stress-ng
14:39:42	1000	16167	96,00	0,00	0,00	3,00	96,00	5	stress-ng
14:39:43	1000	16167	97,00	1,00	0,00	3,00	98,00	5	stress-ng
14:39:44	1000	16167	96,00	1,00	0,00	3,00	97,00	5	stress-ng
14:39:45	1000	16167	95,00	1,00	0,00	4,00	96,00	5	stress-ng
14:39:46	1000	16167	93,00	1,00	0,00	6,00	94,00	5	stress-ng
14:39:47	1000	16167	95,05	0,99	0,00	2,97	96,04	5	stress-ng
14:39:48	1000	16167	98,00	0,00	0,00	3,00	98,00	5	stress-ng
14:39:49	1000	16167	95,05	1,98	0,00	1,98	97,03	5	stress-ng
14:39:50	1000	16167	95,00	2,00	0,00	3,00	97,00	3	stress-ng
14:39:51	1000	16167	97,00	0,00	0,00	3,00	97,00	3	stress-ng
14:39:52	1000	16167	95,05	0,99	0,00	3,96	96,04	3	stress-ng
14:39:53	1000	16167	96,00	0,00	0,00	3,00	96,00	3	stress-ng
14:39:54	1000	16167	97,00	1,00	0,00	4,00	98,00	3	stress-ng
14:39:55	1000	16167	95,00	0,00	0,00	3,00	95,00	3	stress-ng
14:39:56	1000	16167	95,00	2,00	0,00	5,00	97,00	3	stress-ng
14:39:57	1000	16167	92,00	3,00	0,00	5,00	95,00	3	stress-ng
14:39:58	1000	16167	100,00	0,00	0,00	0,00	100,00	3	stress-ng
14:39:59	1000	16167	93,00	0,00	0,00	7,00	93,00	3	stress-ng
14:40:00	1000	16167	96,00	0,00	0,00	3,00	96,00	3	stress-ng
14:40:01	1000	16167	92,08	0,00	0,00	6,93	92,08	3	stress-ng
14:40:02	1000	16167	94,00	2,00	0,00	6,00	96,00	3	stress-ng
14:40:03	1000	16167	91,00	1,00	0,00	8,00	92,00	3	stress-ng
14:40:04	1000	16167	93,00	0,00	0,00	6,00	93,00	3	stress-ng
14:40:05	1000	16167	99,00	1,00	0,00	1,00	100,00	3	stress-ng
14:40:06	1000	16167	96,00	0,00	0,00	4,00	96,00	3	stress-ng
14:40:07	1000	16167	96,04	0,00	0,00	2,97	96,04	3	stress-ng
14:40:08	1000	16167	97,00	1,00	0,00	1,00	98,00	3	stress-ng
14:40:09	1000	16167	98,00	1,00	0,00	2,00	99,00	3	stress-ng
14:40:10	1000	16167	98,00	0,00	0,00	2,00	98,00	3	stress-ng

14:40:11	1000	16167	93,00	3,00	0,00	5,00	96,00	3	stress-ng
14:40:11 UID PID %usr %system %guest %wait %CPU CPU Command									
14:40:12	1000	16167	100,00	0,00	0,00	0,00	100,00	3	stress-ng
14:40:13	1000	16167	99,00	0,00	0,00	1,00	99,00	3	stress-ng
14:40:14	1000	16167	97,03	0,99	0,00	1,98	98,02	3	stress-ng
14:40:15	1000	16167	92,08	0,99	0,00	5,94	93,07	3	stress-ng
14:40:16	1000	16167	99,00	0,00	0,00	2,00	99,00	3	stress-ng
14:40:17	1000	16167	99,00	0,00	0,00	1,00	99,00	5	stress-ng
14:40:18	1000	16167	91,00	1,00	0,00	9,00	92,00	5	stress-ng
14:40:19	1000	16167	92,00	0,00	0,00	6,00	92,00	5	stress-ng
14:40:20	1000	16167	96,04	0,00	0,00	3,96	96,04	5	stress-ng
14:40:21	1000	16167	96,00	0,00	0,00	4,00	96,00	5	stress-ng
14:40:22	1000	16167	99,00	0,00	0,00	2,00	99,00	5	stress-ng
14:40:23	1000	16167	96,00	0,00	0,00	4,00	96,00	5	stress-ng
14:40:24	1000	16167	93,00	1,00	0,00	6,00	94,00	5	stress-ng
14:40:25	1000	16167	88,24	2,94	0,00	9,80	91,18	5	stress-ng
14:40:26	1000	16167	93,88	1,02	0,00	5,10	94,90	5	stress-ng
14:40:27	1000	16167	93,00	1,00	0,00	5,00	94,00	5	stress-ng
14:40:28	1000	16167	93,14	0,00	0,00	5,88	93,14	5	stress-ng
14:40:29	1000	16167	99,00	0,00	0,00	1,00	99,00	5	stress-ng
14:40:30	1000	16167	97,03	0,00	0,00	2,97	97,03	5	stress-ng
14:40:31	1000	16167	97,00	1,00	0,00	2,00	98,00	5	stress-ng
14:40:32	1000	16167	97,00	2,00	0,00	2,00	99,00	5	stress-ng
14:40:33	1000	16167	95,00	0,00	0,00	5,00	95,00	5	stress-ng
14:40:34	1000	16167	93,00	3,00	0,00	4,00	96,00	5	stress-ng
14:40:35	1000	16167	95,00	0,00	0,00	5,00	95,00	5	stress-ng
14:40:36	1000	16167	95,00	0,00	0,00	5,00	95,00	5	stress-ng
14:40:37	1000	16167	95,00	1,00	0,00	4,00	96,00	5	stress-ng
14:40:38	1000	16167	91,00	2,00	0,00	7,00	93,00	5	stress-ng
14:40:39	1000	16167	98,00	0,00	0,00	2,00	98,00	5	stress-ng
14:40:40	1000	16167	93,00	3,00	0,00	4,00	96,00	5	stress-ng
14:40:41	1000	16167	96,00	2,00	0,00	1,00	98,00	5	stress-ng
14:40:42	1000	16167	97,09	0,00	0,00	2,91	97,09	0	stress-ng
14:40:43	1000	16167	97,00	1,00	0,00	3,00	98,00	0	stress-ng
14:40:44	1000	16167	94,00	0,00	0,00	6,00	94,00	0	stress-ng
14:40:45	1000	16167	95,00	0,00	0,00	0,00	95,00	0	stress-ng
14:40:46	1000	16167	96,00	4,00	0,00	5,00	100,00	0	stress-ng
14:40:47	1000	16167	94,00	1,00	0,00	5,00	95,00	0	stress-ng
14:40:48	1000	16167	97,00	3,00	0,00	1,00	100,00	0	stress-ng
14:40:49	1000	16167	99,00	0,00	0,00	0,00	99,00	0	stress-ng
14:40:50	1000	16167	97,00	0,00	0,00	2,00	97,00	0	stress-ng
14:40:51	1000	16167	96,00	1,00	0,00	4,00	97,00	4	stress-ng
14:40:52	1000	16167	90,10	1,98	0,00	7,92	92,08	4	stress-ng
14:40:52 UID PID %usr %system %guest %wait %CPU CPU Command									
14:40:53	1000	16167	97,00	0,00	0,00	2,00	97,00	4	stress-ng
14:40:54	1000	16167	97,00	1,00	0,00	1,00	98,00	4	stress-ng

14:40:55	1000	16167	95,00	1,00	0,00	5,00	96,00	4	stress-ng
14:40:57	1000	16167	99,00	0,00	0,00	2,00	99,00	4	stress-ng
14:40:57	1000	16167	94,00	1,00	0,00	4,00	95,00	4	stress-ng
14:40:58	1000	16167	97,00	0,00	0,00	3,00	97,00	4	stress-ng
14:40:59	1000	16167	94,00	0,00	0,00	6,00	94,00	4	stress-ng
14:41:00	1000	16167	88,00	1,00	0,00	12,00	89,00	4	stress-ng
14:41:01	1000	16167	93,00	2,00	0,00	4,00	95,00	4	stress-ng
14:41:03	1000	16167	98,00	1,00	0,00	2,00	99,00	1	stress-ng
14:41:04	1000	16167	93,00	0,00	0,00	6,00	93,00	1	stress-ng
14:41:05	1000	16167	96,00	1,00	0,00	4,00	97,00	1	stress-ng
14:41:06	1000	16167	94,00	0,00	0,00	5,00	94,00	1	stress-ng
14:41:07	1000	16167	96,04	0,00	0,00	3,96	96,04	1	stress-ng
14:41:08	1000	16167	92,00	4,00	0,00	4,00	96,00	1	stress-ng
14:41:09	1000	16167	94,00	1,00	0,00	5,00	95,00	1	stress-ng
14:41:10	1000	16167	99,00	0,00	0,00	1,00	99,00	1	stress-ng
14:41:11	1000	16167	95,00	1,00	0,00	4,00	96,00	5	stress-ng
14:41:12	1000	16167	96,08	0,00	0,00	2,94	96,08	5	stress-ng
14:41:13	1000	16167	98,98	0,00	0,00	2,04	98,98	5	stress-ng
14:41:14	1000	16167	99,00	0,00	0,00	1,00	99,00	5	stress-ng
14:41:15	1000	16167	91,00	1,00	0,00	9,00	92,00	5	stress-ng
14:41:16	1000	16167	94,00	0,00	0,00	5,00	94,00	5	stress-ng
14:41:17	1000	16167	96,00	1,00	0,00	4,00	97,00	5	stress-ng
14:41:18	1000	16167	96,00	1,00	0,00	2,00	97,00	6	stress-ng
14:41:19	1000	16167	82,00	6,00	0,00	13,00	88,00	1	stress-ng
14:41:20	1000	16167	89,00	2,00	0,00	9,00	91,00	1	stress-ng

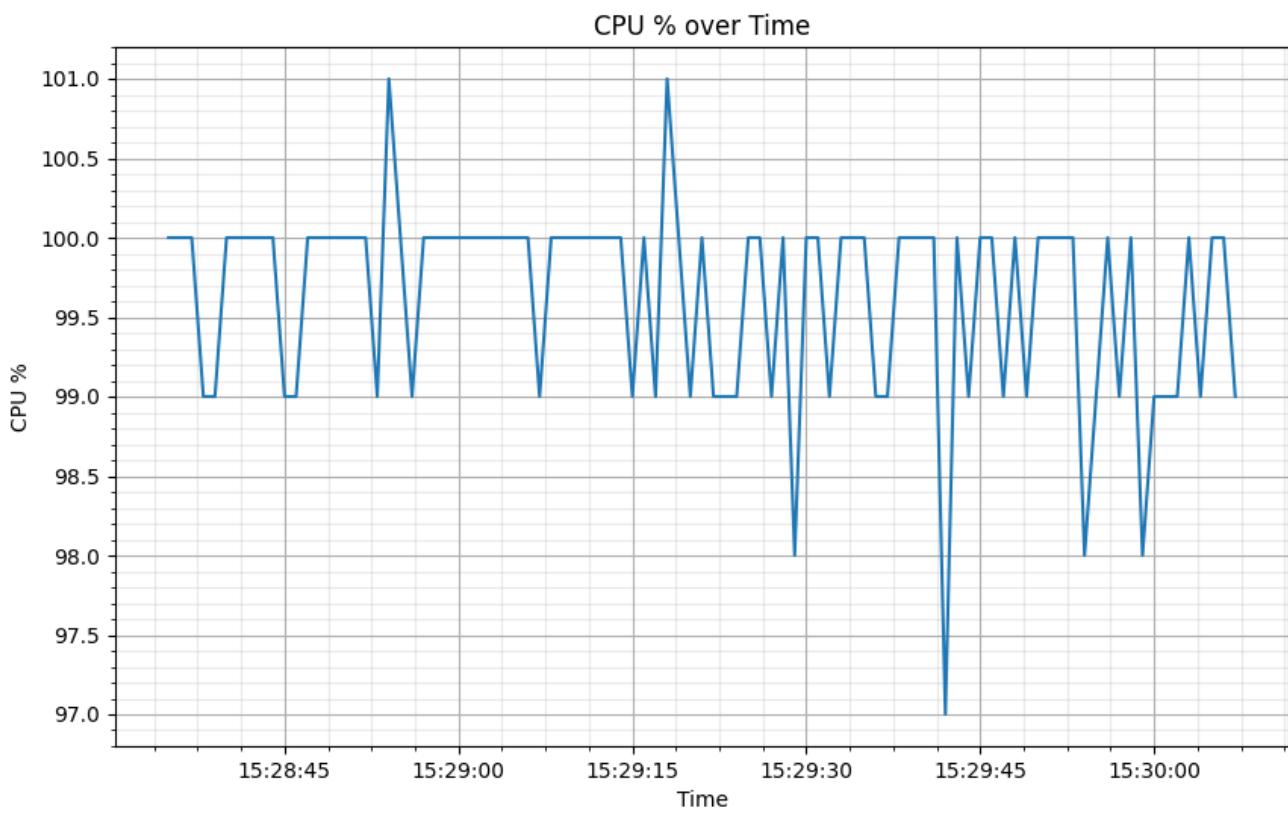
Построим график потребление программой CPU:



Применяем все настройки, как для *fft*. Ставим приоритет -20, пробуем разные политики планирования (самой выгодной опять оказалась NORMAL), закрепляем процессы за конкретными процессорами, отключаем графический интерфейс.

```
alexey@alexey-ubuntu:~$ taskset -c 0-7 sudo chrt -n 19 stress-ng --cpu 0 --cpu-method cdouble --metrics -t 2m
stress-ng: info: [4789] setting to a 120 second (2 mins, 0.00 secs) run per stressor
stress-ng: info: [4789] dispatching hogs: 8 cpu
stress-ng: info: [4789] successful run completed in 120.01s (2 mins, 0.01 secs)
stress-ng: info: [4789] stressor      bogo ops real time   usr time   sys time   bogo ops/s      bogo ops/s CPU used per
stress-ng: info: [4789]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [4789]      cpu          3428179     120.00     956.64      0.27    28568.14      3582.55      99.68
```

Итого с **1'844'831** bogo ops мы повысили производительность до **3'428'179** bogo ops.



1.4 Дополнительные команды для анализа, если интересно

1.4.1 turbostat

```
alexey@alexey-ubuntu:~$ sudo turbostat
turbostat version 2022.10.04 - Len Brown <lenb@kernel.org>
Kernel command line: BOOT_IMAGE=/boot/vmlinuz-6.2.0-35-generic root=UUID=9331bdb4-8632-46ca-936
CPUID(0): GenuineIntel 0x16 CPUID levels
CPUID(1): family:model:stepping 0x6:8e:a (6:142:10) microcode 0x0
CPUID(0x80000000): max_extended_levels: 0x80000008
CPUID(1): SSE3 - - - TSC MSR - HT -
CPUID(6): No-APERF, No-TURBO, DTS, PTM, No-HWP, No-HWPnotify, No-HWPwindow, No-HWPpepp, No-HWPpkl
cpu1: MSR_IA32_MISC_ENABLE: 0x00001810 (No-TCC No-EIST No-MWAIT PREFETCH TURBO)
CPUID(7): No-SGX No-Hybrid
CPUID(0x15): eax_crystal: 2 ebx_tsc: 192 ecx_crystal_hz: 0
TSC: 2304 MHz (24000000 Hz * 192 / 2 / 1000000)
CPUID(0x16): base_mhz: 2300 max_mhz: 3800 bus_mhz: 100
cpu1: MSR_MISC_PWR_MGMT: 0x00000000 (ENable-EIST_Coordination DISable-EPB DISable-OOB)
RAPL: inf sec. Joule Counter Range, at 0 Watts
cpu1: MSR_PLATFORM_INFO: 0x00001700
0 * 100.0 = 0.0 MHz max efficiency frequency
23 * 100.0 = 2300.0 MHz base frequency
cpu1: MSR_IA32_POWER_CTL: 0x00000000 (C1E auto-promotion: DISabled)
cpu1: MSR_PKG_CST_CONFIG_CONTROL: 0x00000000 (UNlocked, pkg-cstate-limit=0 (pc0))
```

```

/dev/cpu_dma_latency: 2000000000 usec (default)
current_driver: acpi_idle
current_governor: menu
current_governor_ro: menu
cpu1: POLL: CPUIDLE CORE POLL IDLE
cpu1: C1: ACPI HLT
NSFOD /sys/devices/system/cpu/cpu1/cpufreq/scaling_driver
cpu1: MSR_MISC_FEATURE_CONTROL: 0x00000000 (L2-Prefetch L2-Prefetch-pair L1-Prefetch L1-IP-Pref
cpu0: MSR_RAPL_POWER_UNIT: 0x00000000 (1.000000 Watts, 1.000000 Joules, 0.000977 sec.)
cpu0: MSR_PKG_POWER_INFO: 0x00000000 (0 W TDP, RAPL 0 - 0 W, 0.000000 sec.)
cpu0: MSR_PKG_POWER_LIMIT: 0x00000000 (Unlocked)
cpu0: PKG Limit #1: DISabled (0.000 Watts, 0.000977 sec, clamp DISabled)
cpu0: PKG Limit #2: DISabled (0.000 Watts, 0.000977* sec, clamp DISabled)
cpu0: MSR_VR_CURRENT_CONFIG: 0x00000000
cpu0: PKG Limit #4: 0.000000 Watts (Unlocked)
cpu0: MSR_DRAM_POWER_LIMIT: 0x00000000 (Unlocked)
cpu0: DRAM Limit: DISabled (0.000 Watts, 0.000977 sec, clamp DISabled)
cpu0: MSR_PPO_POLICY: 0
cpu0: MSR_PPO_POWER_LIMIT: 0x00000000 (Unlocked)
cpu0: Cores Limit: DISabled (0.000 Watts, 0.000977 sec, clamp DISabled)
cpu0: MSR_PP1_POLICY: 0
cpu0: MSR_PP1_POWER_LIMIT: 0x00000000 (Unlocked)
cpu0: GFX Limit: DISabled (0.000 Watts, 0.000977 sec, clamp DISabled)
cpu0: MSR_IA32_TEMPERATURE_TARGET: 0x00c80000 (200 C)
cpu0: MSR_IA32_PACKAGE_THERM_STATUS: 0x00160000 (178 C)
cpu0: MSR_IA32_PACKAGE_THERM_INTERRUPT: 0x00000000 (200 C, 200 C)
cpu1: MSR_PKGC3_IRTL: 0x00000000 (NOTvalid, 0 ns)
cpu1: MSR_PKGC6_IRTL: 0x00000000 (NOTvalid, 0 ns)
cpu1: MSR_PKGC7_IRTL: 0x00000000 (NOTvalid, 0 ns)
cpu1: MSR_PKGC8_IRTL: 0x00000000 (NOTvalid, 0 ns)
cpu1: MSR_PKGC9_IRTL: 0x00000000 (NOTvalid, 0 ns)
cpu1: MSR_PKGC10_IRTL: 0x00000000 (NOTvalid, 0 ns)
turbostat: cpu1: perf instruction counter: No such file or directory
Can not set timer.

Core CPU TSC_MHz IRQ SMI POLL C1 POLL% C1% CPU%c1 CPU%c3 CPU%c6 CPU%c7 CoreTmp PkgTmp Totl%COAr
GFX%CO CPUGFX% PkgWatt CorWatt GFXWatt RAMWatt PKG_% RAM_%
--2303 9689 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178 178 0.00 0.00 0.00 0.00 0.00 36755126701004093
806784702202014720.00 749353303706249472.00 720671569983397120.00 74608482807378272.00 67477450
0 0 2304 1194 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178 178 0.00 0.00 0.00 0.00 0.00 368139479310987
808075843679804672.00 750552534590639488.00 721824899927965824.00 74727882823869232.00 67585438
1 1 2304 1216 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178
2 2 2304 1185 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178
3 3 2304 1191 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178
4 4 2304 1195 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178
5 5 2304 1184 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178
6 6 2304 1295 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178
7 7 2304 1229 0 0 0 0.00 0.00 100.00 0.00 0.00 0.00 178

```

1.4.2 uptime

```
alexey@alexey-ubuntu:~$ uptime
17:50:29 up 1:07, 1 user, load average: 8,11, 4,01, 2,21
```

1.4.3 numastat

```
alexey@alexey-ubuntu:~$ numastat -cm
```

Per-node system memory usage (in MBs):

Token SwapCached not in hash table.
Token SecPageTables not in hash table.
Token FileHugePages not in hash table.
Token FilePmdMapped not in hash table.

	Node 0	Total
MemTotal	1952	1952
MemFree	196	196
MemUsed	1755	1755
Active	558	558
Inactive	719	719
Active(anon)	175	175
Inactive(anon)	436	436
Active(file)	383	383
Inactive(file)	283	283
Unevictable	133	133
Mlocked	20	20
Dirty	0	0
Writeback	0	0
FilePages	819	819
Mapped	177	177
AnonPages	604	604
Shmem	129	129
KernelStack	10	10
PageTables	18	18
NFS_Unstable	0	0
Bounce	0	0
WritebackTmp	0	0
Slab	242	242
SReclaimable	130	130
SUnreclaim	112	112
AnonHugePages	0	0
ShmemHugePages	0	0
ShmemPmdMapped	0	0
HugePages_Total	0	0
HugePages_Free	0	0
HugePages_Surp	0	0
KReclaimable	130	130

2 Cache Testing

Выданные параметры: [l1cache-sets,cache-fence] Но начнем с настройки стрессора

2.1 l1cache

Из документации stress-ng:

-l1cache N: start *N* workers that exercise the CPU level 1 cache with reads and writes. A cache aligned buffer that is twice the level 1 cache size is read and then written in level 1 cache set sized steps over each level 1 cache set. This is designed to exercise cache block evictions. The bogo-op count measures the number of million cache lines touched. Where possible, the level 1 cache geometry is determined from the kernel, however, this is not possible on some architectures or kernels, so one may need to specify these manually. One can specify 3 out of the 4 cache geometric parameters, these are as follows:

Для нахождения оптимального значения параметра l1cache напишем простейший zsh-скрипт:

```
#!/bin/zsh
for i in {1..16..1}; do
    echo "run with l1cache=$i"
    a='stress-ng --l1cache $i --metrics --timeout 2'
    echo "${a}" | cut -c 214-405
done

./l1cache-testing.bash
run with l1cache=1
stress-ng: info: [114470] setting to a 2 second run per stressor
stress-ng: info: [114470] dispatching hogs: 1 l1cache
stress-ng: info: [114471] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size: 32.0K
stress-ng: info: [114470] successful run completed in 2.13s
stress-ng: info: [114470] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [114470]                      (secs)      (secs)      (secs)      (real time)
stress-ng: info: [114470] l1cache           320        2.13        2.12        0.00       150.50

run with l1cache=2
stress-ng: info: [114480] setting to a 2 second run per stressor
stress-ng: info: [114480] dispatching hogs: 2 l1cache
stress-ng: info: [114481] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size: 32.0K
stress-ng: info: [114480] successful run completed in 2.24s
stress-ng: info: [114480] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [114480]                      (secs)      (secs)      (secs)      (real time)
stress-ng: info: [114480] l1cache           640        2.23        4.45        0.00       286.63

run with l1cache=3
stress-ng: info: [114491] setting to a 2 second run per stressor
stress-ng: info: [114491] dispatching hogs: 3 l1cache
stress-ng: info: [114492] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size: 32.0K
stress-ng: info: [114491] successful run completed in 2.17s
```

```

stress-ng: info: [114491] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114491]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [114491] l1cache          960       2.15       6.45       0.00      445.71

run with l1cache=4
stress-ng: info: [114503] setting to a 2 second run per stressor
stress-ng: info: [114503] dispatching hogs: 4 l1cache
stress-ng: info: [114504] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114503] successful run completed in 2.39s
stress-ng: info: [114503] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114503]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [114503] l1cache          1088      2.13       8.50       0.00      509.85

run with l1cache=5
stress-ng: info: [114516] setting to a 2 second run per stressor
stress-ng: info: [114516] dispatching hogs: 5 l1cache
stress-ng: info: [114517] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114516] successful run completed in 2.34s
stress-ng: info: [114516] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114516]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [114516] l1cache          1280      2.29      11.41       0.00      559.11

run with l1cache=6
stress-ng: info: [114533] setting to a 2 second run per stressor
stress-ng: info: [114533] dispatching hogs: 6 l1cache
stress-ng: info: [114534] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114533] successful run completed in 2.52s
stress-ng: info: [114533] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114533]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [114533] l1cache          1408      2.36      14.08       0.00      596.64

run with l1cache=7
stress-ng: info: [114548] setting to a 2 second run per stressor
stress-ng: info: [114548] dispatching hogs: 7 l1cache
stress-ng: info: [114549] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114548] successful run completed in 2.29s
stress-ng: info: [114548] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114548]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [114548] l1cache          1344      2.24      15.62       0.01      598.99

run with l1cache=8
stress-ng: info: [114564] setting to a 2 second run per stressor
stress-ng: info: [114564] dispatching hogs: 8 l1cache
stress-ng: info: [114565] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114564] successful run completed in 2.65s
stress-ng: info: [114564] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114564]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [114564] l1cache          1408      2.47      19.19       0.12      570.22

```

```

run with l1cache=9
stress-ng: info: [114581] setting to a 2 second run per stressor
stress-ng: info: [114581] dispatching hogs: 9 l1cache
stress-ng: info: [114582] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114581] successful run completed in 2.78s
stress-ng: info: [114581] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [114581]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [114581] l1cache                1408       2.49     19.72      0.14      565.79

run with l1cache=10
stress-ng: info: [114599] setting to a 2 second run per stressor
stress-ng: info: [114599] dispatching hogs: 10 l1cache
stress-ng: info: [114600] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114599] successful run completed in 2.56s
stress-ng: info: [114599] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [114599]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [114599] l1cache                1408       2.27     18.05      0.10      620.04

run with l1cache=11
stress-ng: info: [114618] setting to a 2 second run per stressor
stress-ng: info: [114618] dispatching hogs: 11 l1cache
stress-ng: info: [114619] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114618] successful run completed in 2.68s
stress-ng: info: [114618] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [114618]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [114618] l1cache                1536       2.43     19.64      0.07      631.13

run with l1cache=12
stress-ng: info: [114641] setting to a 2 second run per stressor
stress-ng: info: [114641] dispatching hogs: 12 l1cache
stress-ng: info: [114642] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114641] successful run completed in 2.84s
stress-ng: info: [114641] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [114641]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [114641] l1cache                1664       2.64     21.47      0.11      630.08

run with l1cache=13
stress-ng: info: [114662] setting to a 2 second run per stressor
stress-ng: info: [114662] dispatching hogs: 13 l1cache
stress-ng: info: [114663] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114662] successful run completed in 2.88s
stress-ng: info: [114662] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [114662]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [114662] l1cache                1664       2.55     21.28      0.12      653.19

run with l1cache=14
stress-ng: info: [114684] setting to a 2 second run per stressor

```

```

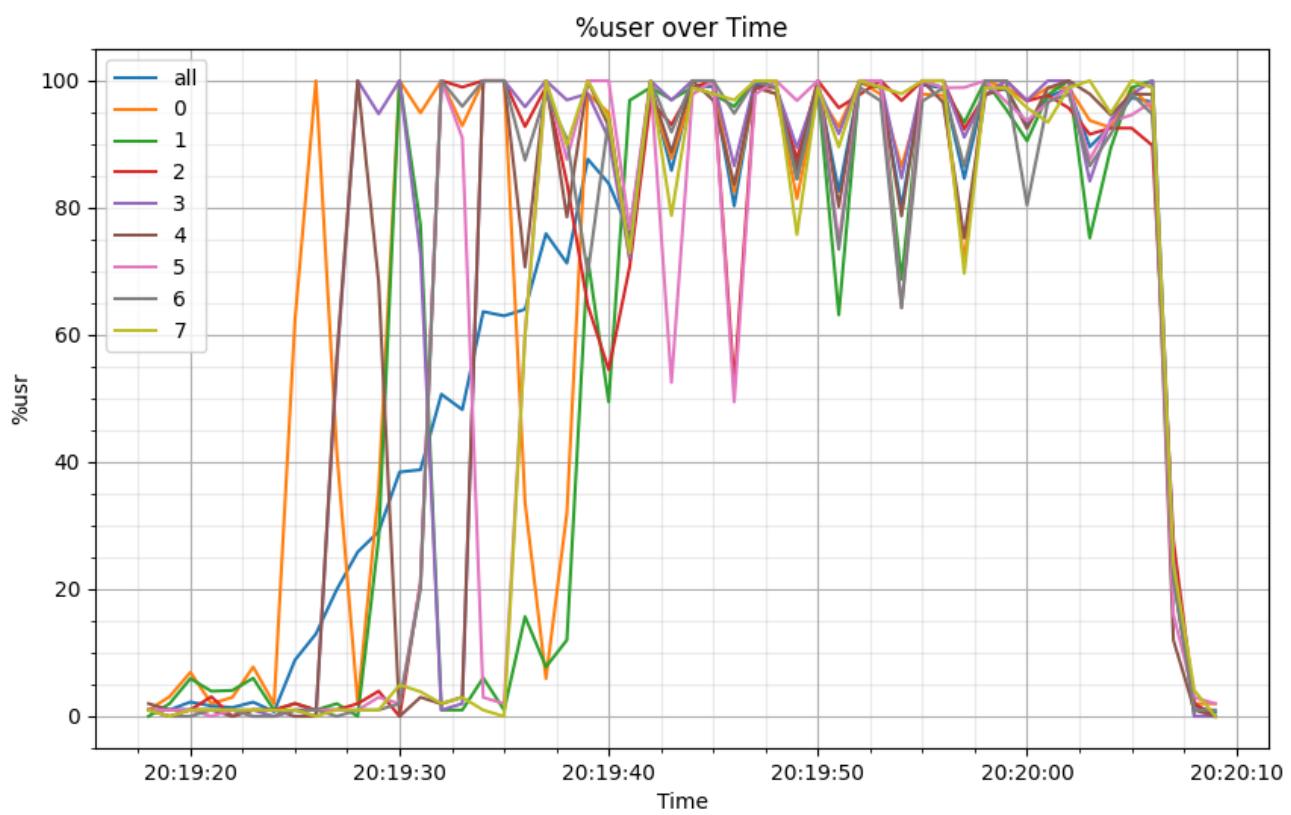
stress-ng: info: [114684] dispatching hogs: 14 l1cache
stress-ng: info: [114685] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114684] successful run completed in 3.05s
stress-ng: info: [114684] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114684]                               (secs)     (secs)     (secs)     (real time)
stress-ng: info: [114684] l1cache                1728       2.78      23.07      0.10      622.63

run with l1cache=15
stress-ng: info: [114710] setting to a 2 second run per stressor
stress-ng: info: [114710] dispatching hogs: 15 l1cache
stress-ng: info: [114711] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114710] successful run completed in 3.19s
stress-ng: info: [114710] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114710]                               (secs)     (secs)     (secs)     (real time)
stress-ng: info: [114710] l1cache                1920       2.93      24.26      0.12      655.63

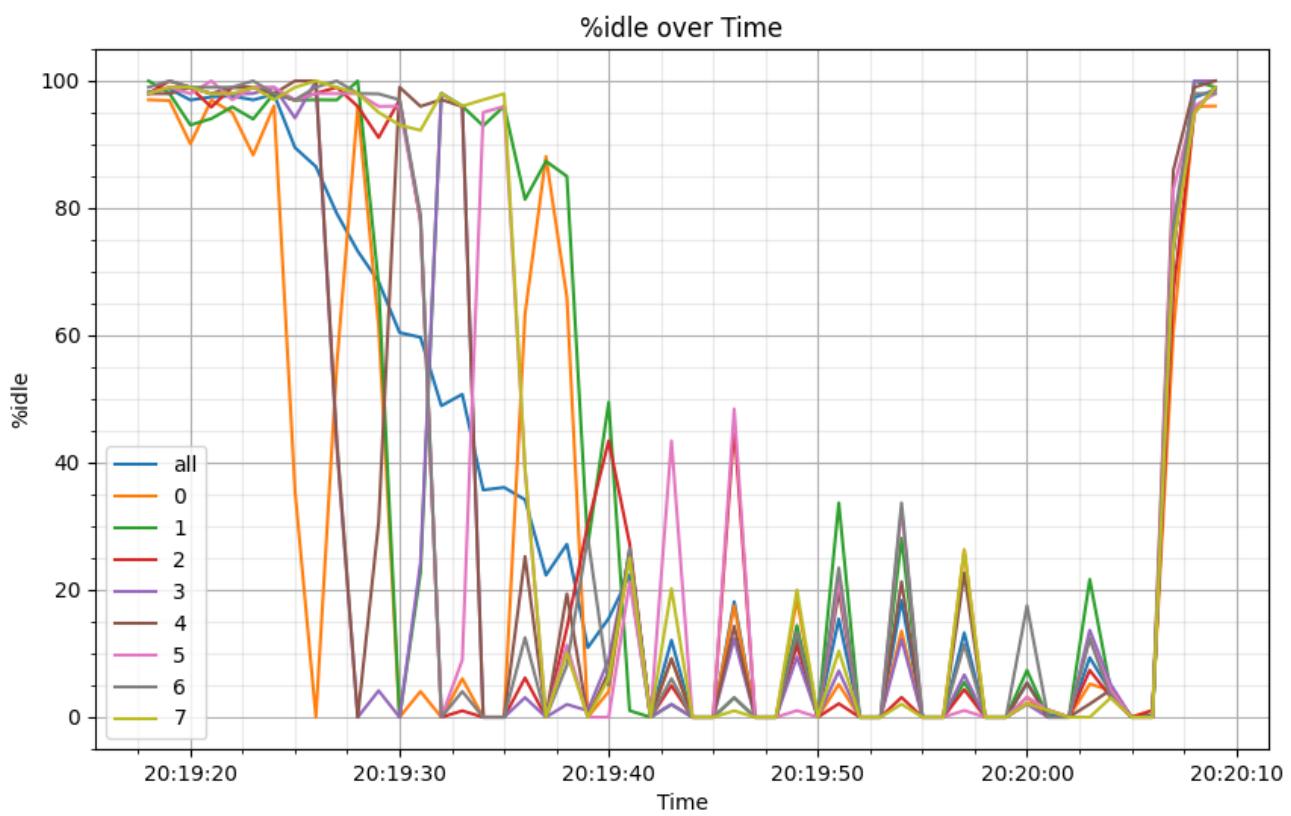
run with l1cache=16
stress-ng: info: [114737] setting to a 2 second run per stressor
stress-ng: info: [114737] dispatching hogs: 16 l1cache
stress-ng: info: [114738] stress-ng-l1cache: l1cache: size: 32.0K, sets: 64, ways: 8, line size
stress-ng: info: [114737] successful run completed in 3.40s
stress-ng: info: [114737] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [114737]                               (secs)     (secs)     (secs)     (real time)
stress-ng: info: [114737] l1cache                2048       3.24      26.16      0.11      632.94

```

Построим графики!

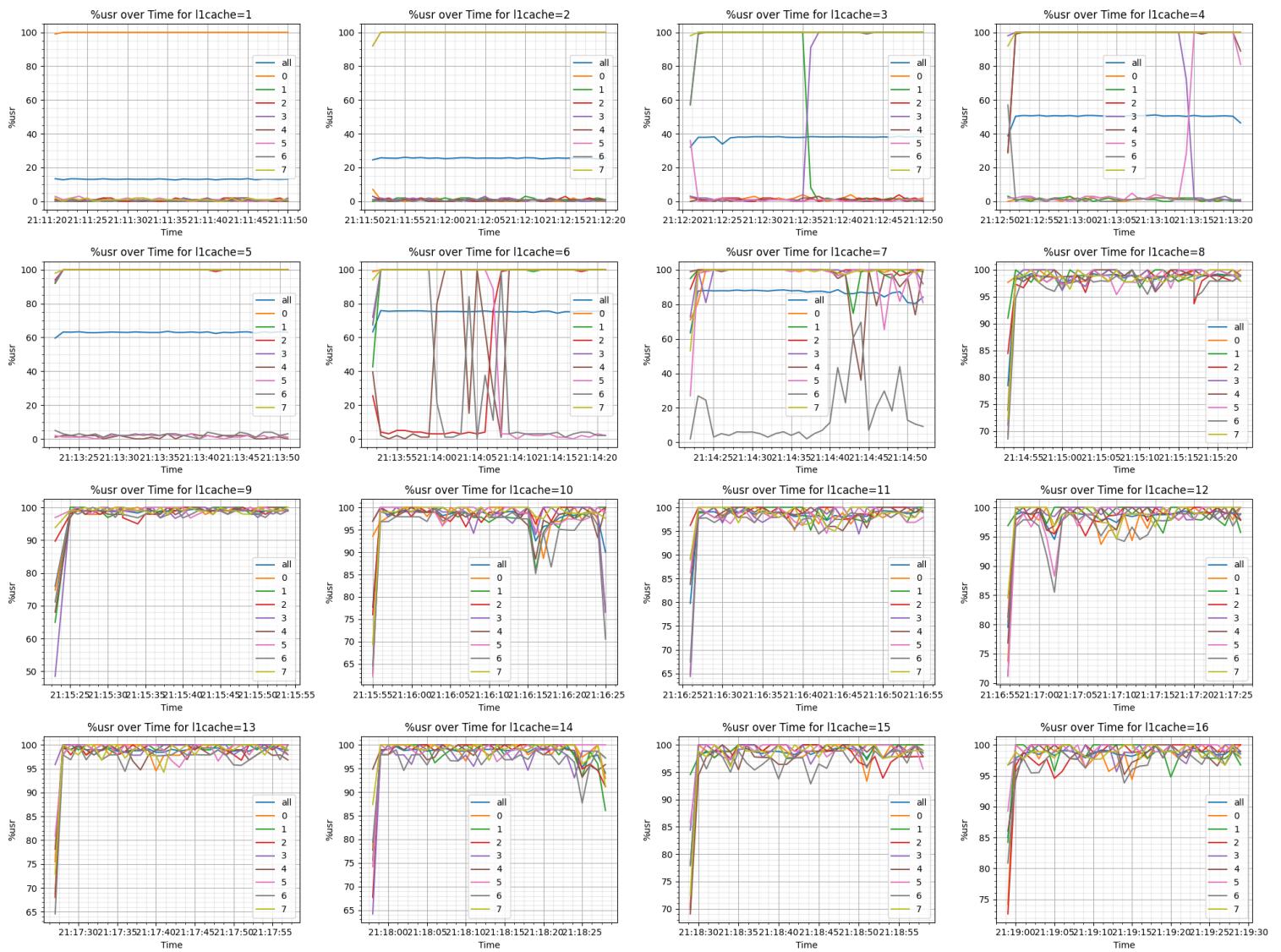


Вот ещё есть график для idle

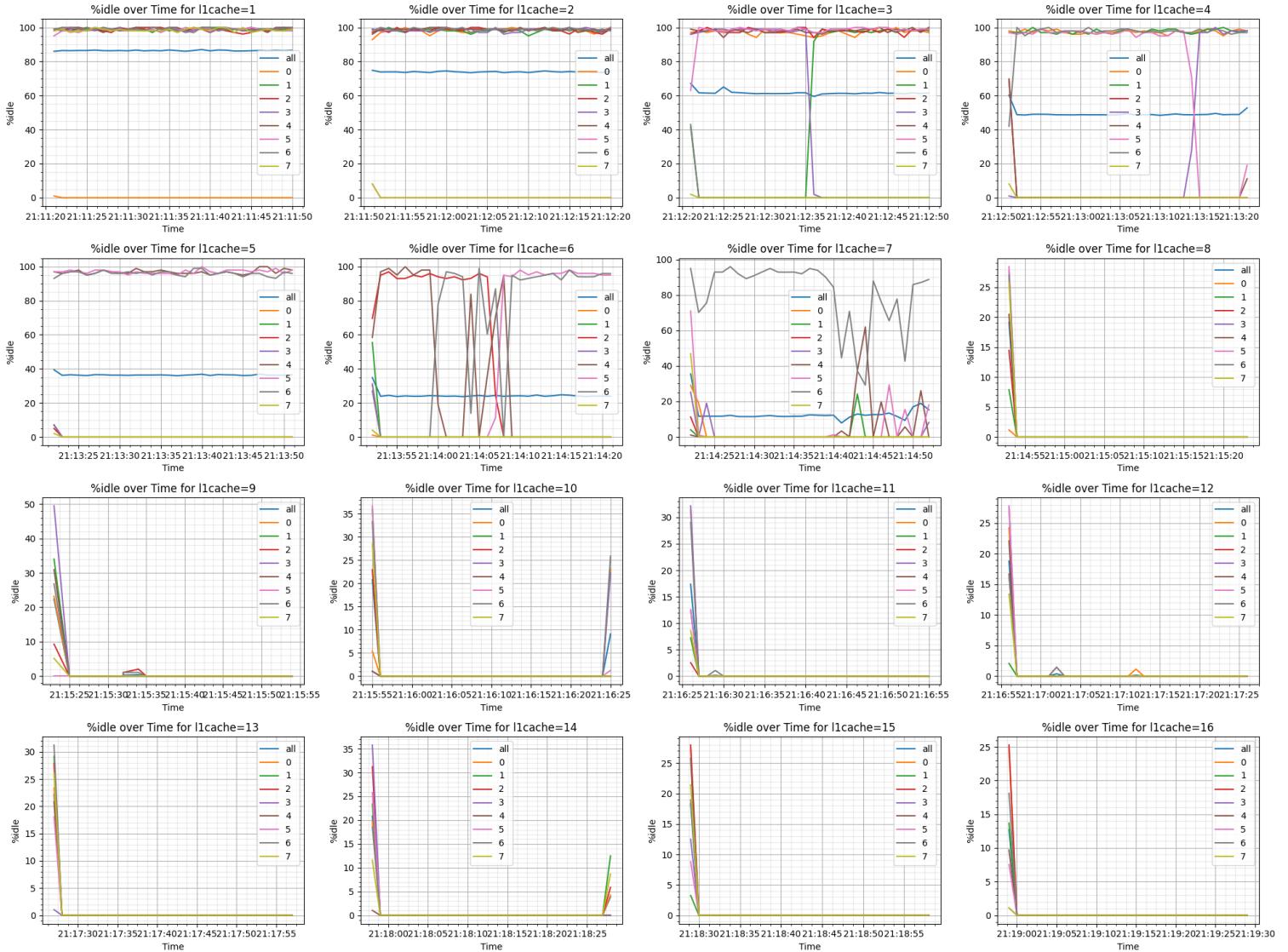


Будем на них внимательно смотреть... шучу, сейчас будут более информативные графики.

user time



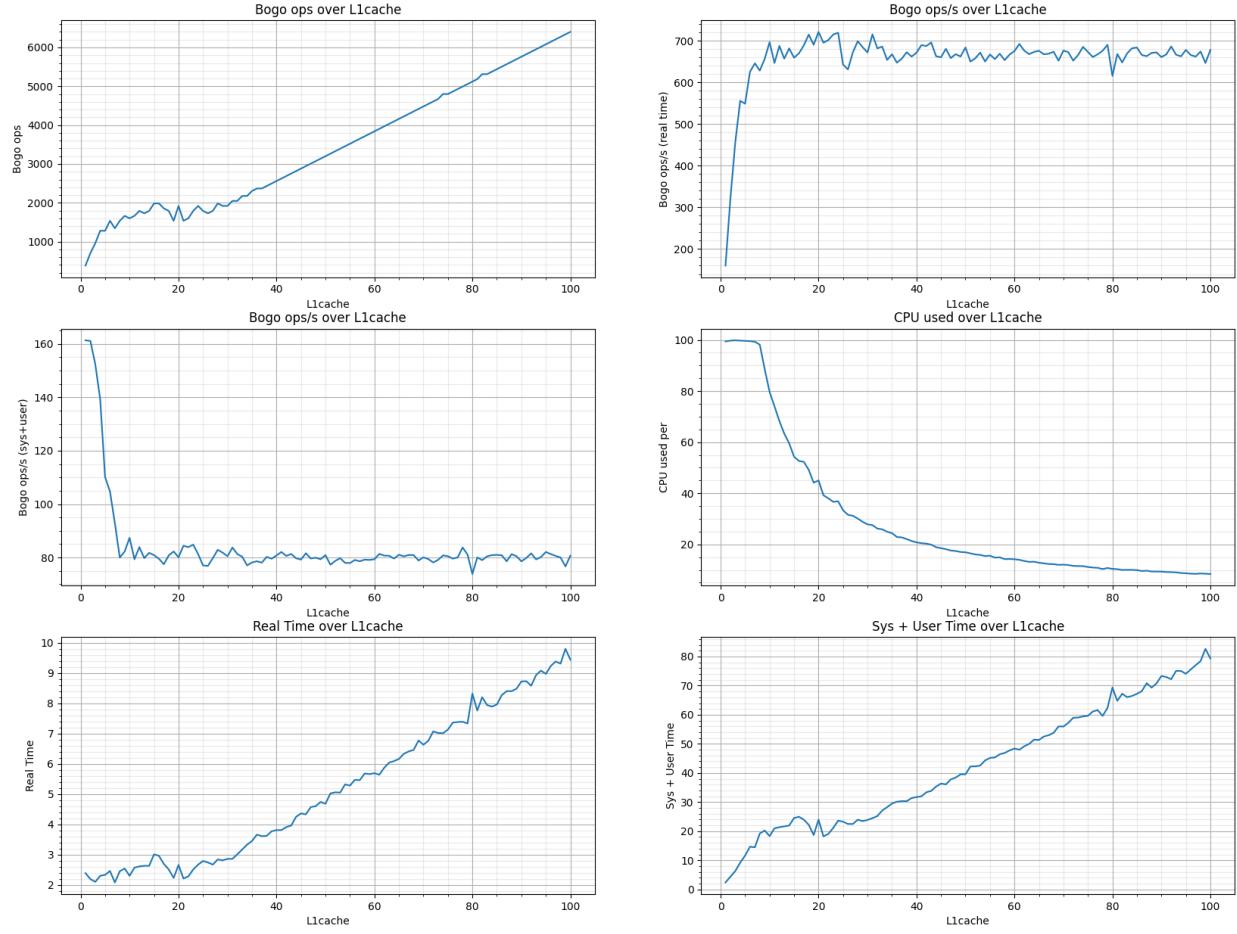
idle time



Видим, что после 8 workers наши ядра заняты посностью, что соответствует количеству виртуальных ядер на процессоре.

Так же из графиков видно, что если не все ядра заняты работой, то может произойти переключение на другое ядро из-за чего может упасть производительность.

Проверим наши доводы построив график *bogo ops* от *l1cache*.



Видим, хоть с увеличением воркеров растет bogo ops, но нагрузка на каждый CPU (после 8 воркеров) падает.

Это увеличивает время ожидания в блокировке.

Вывод: оптимальное значение параметра l1cache равно 8.

2.2 l1cache-sets

Из документации stress-ng:

-l1cache-sets N: specify the number of level 1 cache sets

Для нахождения оптимального значения параметра l1cache-sets напишем простейший zsh-скрипт:

```
#!/bin/bash
mpstat -P ALL 1 &
mpstat_pid=$!
for i in {1..16..1}; do
    echo "l1cache-sets=$i"
    stress-ng --l1cache 8 --l1cache-sets $i --metrics --timeout 30 > /dev/null &
    stress_ng_pid=$! # Get the process ID of stress-ng
```

```

wait $stress_ng_pid # Wait for stress-ng to finish before moving to the next iteration
done
kill $mpstat_pid

./l1cache-sets-testing.zsh >> ~/output.txt
stress-ng: info: [144992] setting to a 30 second run per stressor
stress-ng: info: [144992] dispatching hogs: 8 l1cache
stress-ng: info: [144993] stress-ng-l1cache: l1cache: size: 32.0K, sets: 1, ways: 8, line size
stress-ng: info: [144992] successful run completed in 30.62s
stress-ng: info: [144992] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [144992]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [144992] l1cache          274     30.45    236.47     1.42      9.00
stress-ng: info: [145077] setting to a 30 second run per stressor
stress-ng: info: [145077] dispatching hogs: 8 l1cache
stress-ng: info: [145078] stress-ng-l1cache: l1cache: size: 32.0K, sets: 2, ways: 8, line size
stress-ng: info: [145077] successful run completed in 30.46s
stress-ng: info: [145077] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145077]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145077] l1cache          556     30.28    236.11     1.36     18.36
stress-ng: info: [145168] setting to a 30 second run per stressor
stress-ng: info: [145168] dispatching hogs: 8 l1cache
stress-ng: info: [145169] stress-ng-l1cache: l1cache: size: 32.0K, sets: 3, ways: 8, line size
stress-ng: info: [145168] successful run completed in 30.70s
stress-ng: info: [145168] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145168]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145168] l1cache          840     30.55    238.67     1.21     27.50
stress-ng: info: [145252] setting to a 30 second run per stressor
stress-ng: info: [145252] dispatching hogs: 8 l1cache
stress-ng: info: [145253] stress-ng-l1cache: l1cache: size: 32.0K, sets: 4, ways: 8, line size
stress-ng: info: [145252] successful run completed in 30.52s
stress-ng: info: [145252] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145252]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145252] l1cache          1168    30.31    236.89     1.29     38.54
stress-ng: info: [145338] setting to a 30 second run per stressor
stress-ng: info: [145338] dispatching hogs: 8 l1cache
stress-ng: info: [145339] stress-ng-l1cache: l1cache: size: 32.0K, sets: 5, ways: 8, line size
stress-ng: info: [145338] successful run completed in 30.56s
stress-ng: info: [145338] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145338]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145338] l1cache          1370    30.35    236.78     1.29     45.15
stress-ng: info: [145425] setting to a 30 second run per stressor
stress-ng: info: [145425] dispatching hogs: 8 l1cache
stress-ng: info: [145426] stress-ng-l1cache: l1cache: size: 32.0K, sets: 6, ways: 8, line size
stress-ng: info: [145425] successful run completed in 30.49s
stress-ng: info: [145425] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145425]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145425] l1cache          1734    30.33    236.89     1.19     57.18
stress-ng: info: [145512] setting to a 30 second run per stressor
stress-ng: info: [145512] dispatching hogs: 8 l1cache

```

```

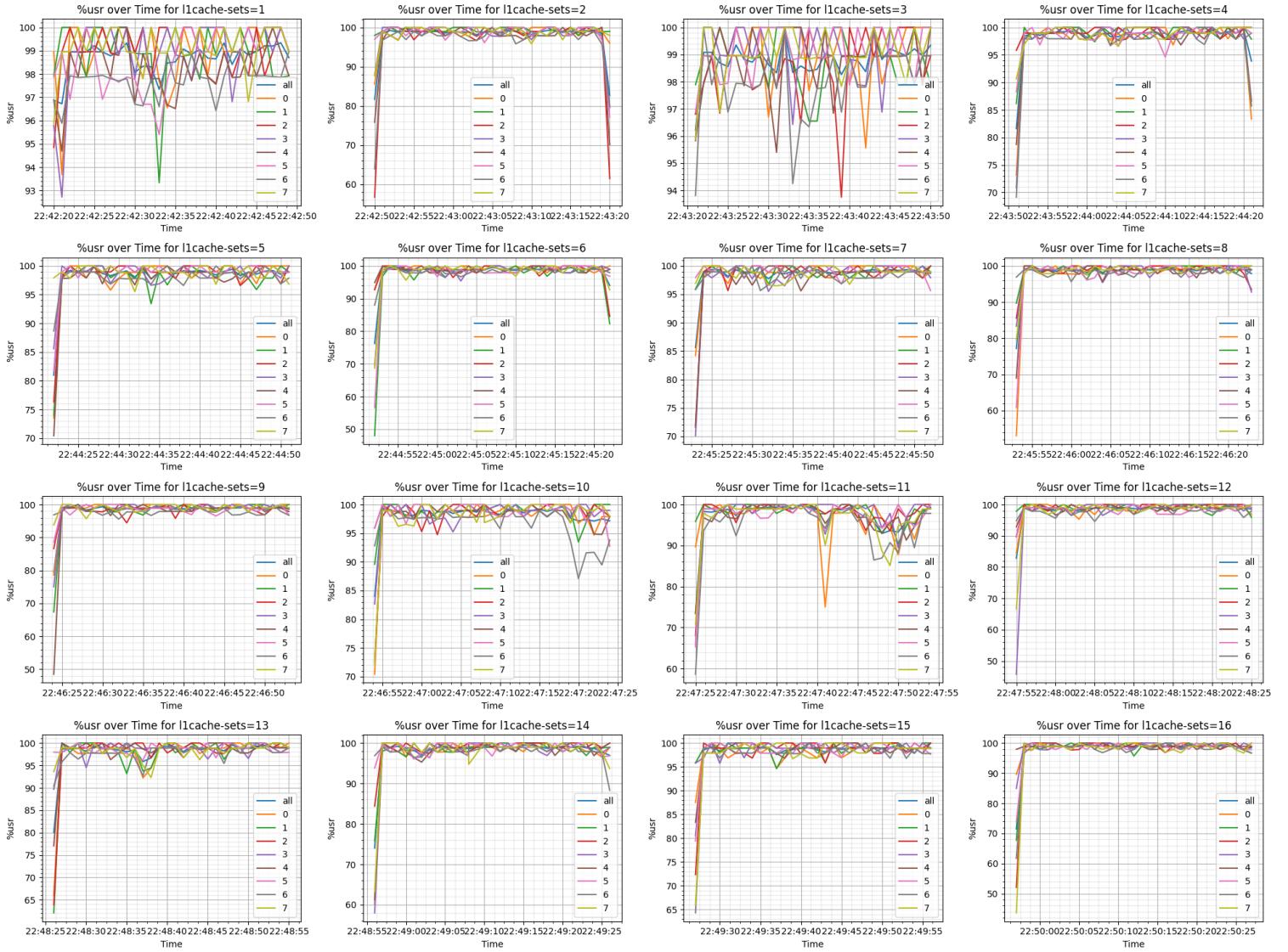
stress-ng: info: [145513] stress-ng-l1cache: l1cache: size: 32.0K, sets: 7, ways: 8, line size
stress-ng: info: [145512] successful run completed in 30.61s
stress-ng: info: [145512] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145512]                                     (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145512] l1cache          1904    30.40    237.42     1.24    62.64
stress-ng: info: [145599] setting to a 30 second run per stressor
stress-ng: info: [145599] dispatching hogs: 8 l1cache
stress-ng: info: [145600] stress-ng-l1cache: l1cache: size: 32.0K, sets: 8, ways: 8, line size
stress-ng: info: [145599] successful run completed in 30.66s
stress-ng: info: [145599] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145599]                                     (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145599] l1cache          2120    30.46    238.40     1.14    69.61
stress-ng: info: [145683] setting to a 30 second run per stressor
stress-ng: info: [145683] dispatching hogs: 8 l1cache
stress-ng: info: [145684] stress-ng-l1cache: l1cache: size: 32.0K, sets: 9, ways: 8, line size
stress-ng: info: [145683] successful run completed in 30.43s
stress-ng: info: [145683] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145683]                                     (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145683] l1cache          2439    30.29    236.52     1.24    80.51
stress-ng: info: [145767] setting to a 30 second run per stressor
stress-ng: info: [145767] dispatching hogs: 8 l1cache
stress-ng: info: [145768] stress-ng-l1cache: l1cache: size: 32.0K, sets: 10, ways: 8, line size
stress-ng: info: [145767] successful run completed in 30.48s
stress-ng: info: [145767] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145767]                                     (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145767] l1cache          2730    30.23    234.56     1.64    90.31
stress-ng: info: [145855] setting to a 30 second run per stressor
stress-ng: info: [145855] dispatching hogs: 8 l1cache
stress-ng: info: [145856] stress-ng-l1cache: l1cache: size: 32.0K, sets: 11, ways: 8, line size
stress-ng: info: [145855] successful run completed in 30.61s
stress-ng: info: [145855] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [145855]                                     (secs)   (secs)   (secs)   (real time)
stress-ng: info: [145855] l1cache          2926    30.46    227.35     1.90    96.05
stress-ng: info: [146002] setting to a 30 second run per stressor
stress-ng: info: [146002] dispatching hogs: 8 l1cache
stress-ng: info: [146003] stress-ng-l1cache: l1cache: size: 32.0K, sets: 12, ways: 8, line size
stress-ng: info: [146002] successful run completed in 30.41s
stress-ng: info: [146002] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [146002]                                     (secs)   (secs)   (secs)   (real time)
stress-ng: info: [146002] l1cache          3252    30.23    235.57     1.21    107.58
stress-ng: info: [146094] setting to a 30 second run per stressor
stress-ng: info: [146094] dispatching hogs: 8 l1cache
stress-ng: info: [146095] stress-ng-l1cache: l1cache: size: 32.0K, sets: 13, ways: 8, line size
stress-ng: info: [146094] successful run completed in 30.47s
stress-ng: info: [146094] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [146094]                                     (secs)   (secs)   (secs)   (real time)
stress-ng: info: [146094] l1cache          3419    30.24    234.76     1.40    113.06
stress-ng: info: [146181] setting to a 30 second run per stressor

```

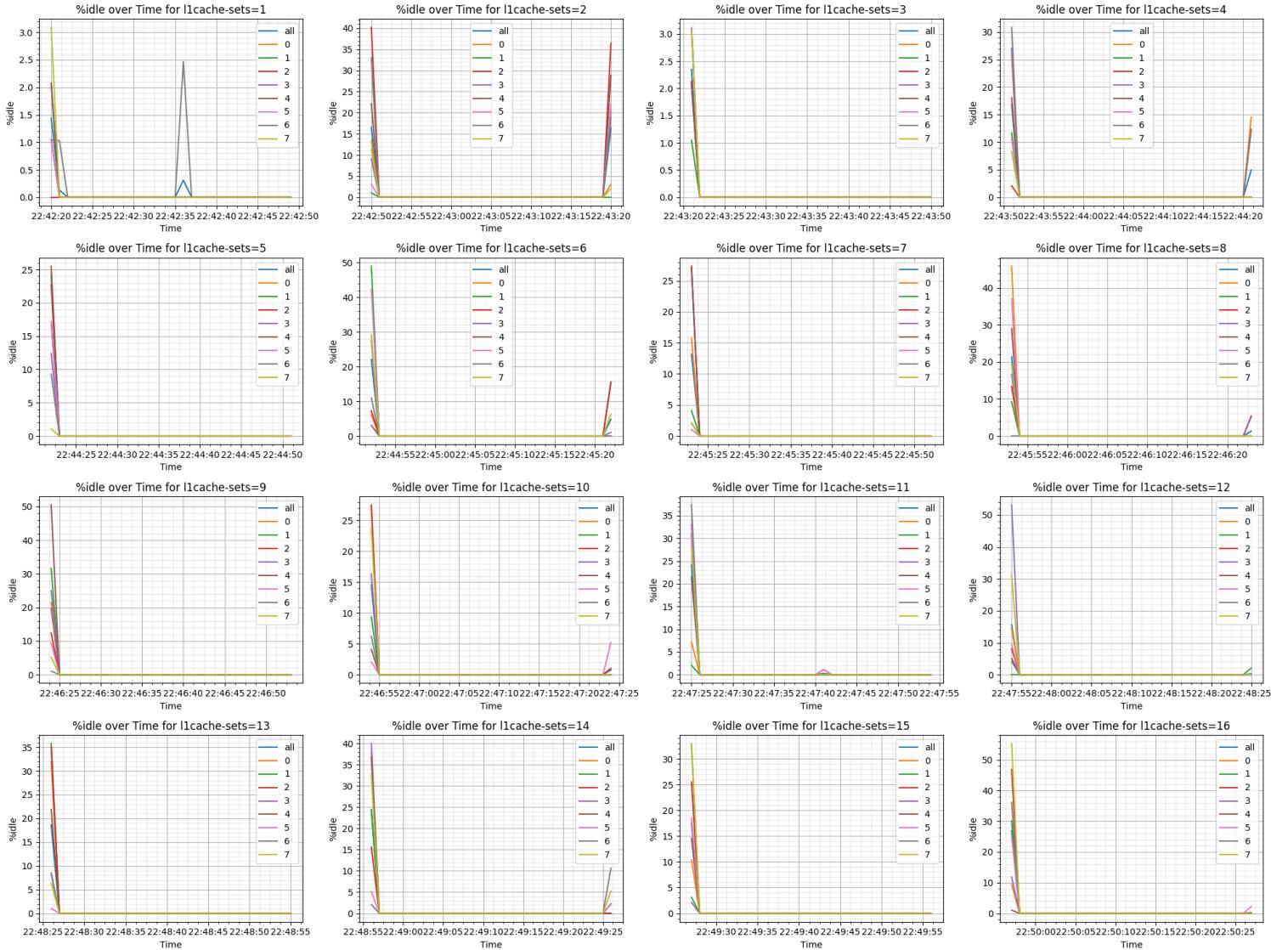
```
stress-ng: info: [146181] dispatching hogs: 8 l1cache
stress-ng: info: [146182] stress-ng-l1cache: l1cache: size: 32.0K, sets: 14, ways: 8, line size
stress-ng: info: [146181] successful run completed in 30.52s
stress-ng: info: [146181] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [146181]                               (secs)   (secs)   (secs)   (real time)
stress-ng: info: [146181] l1cache          3794    30.34    236.64     1.26    125.04
stress-ng: info: [146266] setting to a 30 second run per stressor
stress-ng: info: [146266] dispatching hogs: 8 l1cache
stress-ng: info: [146267] stress-ng-l1cache: l1cache: size: 32.0K, sets: 15, ways: 8, line size
stress-ng: info: [146266] successful run completed in 30.59s
stress-ng: info: [146266] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [146266]                               (secs)   (secs)   (secs)   (real time)
stress-ng: info: [146266] l1cache          4260    30.34    236.48     1.40    140.42
stress-ng: info: [146354] setting to a 30 second run per stressor
stress-ng: info: [146354] dispatching hogs: 8 l1cache
stress-ng: info: [146355] stress-ng-l1cache: l1cache: size: 32.0K, sets: 16, ways: 8, line size
stress-ng: info: [146354] successful run completed in 30.45s
stress-ng: info: [146354] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [146354]                               (secs)   (secs)   (secs)   (real time)
stress-ng: info: [146354] l1cache          4464    30.22    236.06     1.18    147.73
```

Графики...графики...графики...

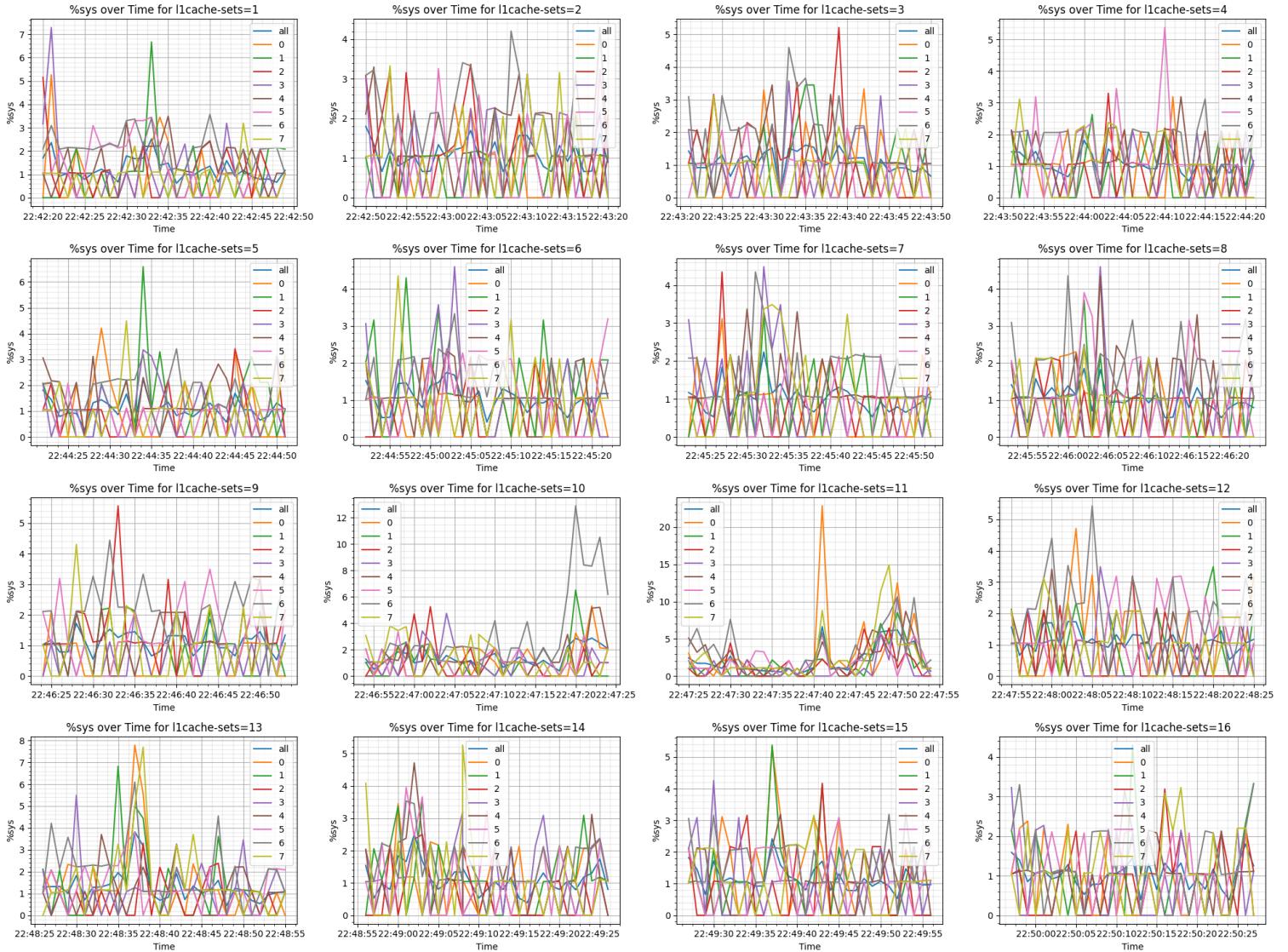
user time



idle time



sys time



Делаем вывод что чем больше set, тем выше bogo-ops.

Тут должен быть *perf*, но у меня он ничего не поддерживает:

```
sudo perf stat -B -e cache-references,cache-misses,cycles,instructions,branches,faults,migrations
stress-ng: info: [161315] setting to a 30 second run per stressor
stress-ng: info: [161315] dispatching hogs: 8 l1cache
stress-ng: info: [161316] stress-ng-l1cache: l1cache: size: 32.0K, sets: 8, ways: 8, line size
stress-ng: info: [161315] successful run completed in 30.63s
stress-ng: info: [161315] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [161315]                                (secs)      (secs)      (secs)      (real time)
stress-ng: info: [161315] l1cache                      2264        30.47      238.64      1.11          74.30
```

Performance counter stats for 'stress-ng --l1cache 8 --l1cache-sets 8 --metrics --timeout 30'@1.1GHz

<not supported>	cache-references
<not supported>	cache-misses
<not supported>	cycles
<not supported>	instructions

```

<not supported>      branches
    1678      faults
        9      migrations
<not supported>      L1-dcache-loads
<not supported>      L1-dcache-load-misses
<not supported>      L1-dcache-stores

```

30,642159810 seconds time elapsed

238,688613000 seconds user
1,177890000 seconds sys

Попробуем провести тот же тест но с большими set

```

#!/bin/bash
mpstat -P ALL 1 &
mpstat_pid=$!
for i in {100..1000..100}; do
    echo "l1cache-sets=$i"
    stress-ng --l1cache 8 --l1cache-sets $i --metrics --timeout 30 > /dev/null &
    stress_ng_pid=$! # Get the process ID of stress-ng
    wait $stress_ng_pid # Wait for stress-ng to finish before moving to the next iteration
done
kill $mpstat_pid

./l1cache-sets-testing-2.zsh > ~/output2.txt
stress-ng: info: [163942] setting to a 30 second run per stressor
stress-ng: info: [163942] dispatching hogs: 8 l1cache
stress-ng: info: [163943] stress-ng-l1cache: l1cache: size: 32.0K, sets: 100, ways: 8, line si
stress-ng: info: [163942] successful run completed in 30.52s
stress-ng: info: [163942] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [163942]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [163942] l1cache          26800    30.37    233.71     2.18     882.42
stress-ng: info: [164026] setting to a 30 second run per stressor
stress-ng: info: [164026] dispatching hogs: 8 l1cache
stress-ng: info: [164027] stress-ng-l1cache: l1cache: size: 32.0K, sets: 200, ways: 8, line si
stress-ng: info: [164026] successful run completed in 30.50s
stress-ng: info: [164026] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164026]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [164026] l1cache          55200    30.31    235.99     1.28     1821.41
stress-ng: info: [164112] setting to a 30 second run per stressor
stress-ng: info: [164112] dispatching hogs: 8 l1cache
stress-ng: info: [164113] stress-ng-l1cache: l1cache: size: 32.0K, sets: 300, ways: 8, line si
stress-ng: info: [164112] unsuccessful run completed in 1.34s
stress-ng: info: [164112] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164112]                      (secs)   (secs)   (secs)   (real time)
stress-ng: info: [164112] l1cache          2400     0.00     0.00     0.00     0.00
stress-ng: fail: [164112] l1cache instance 0 corrupted bogo-ops counter, 300 vs 0
stress-ng: fail: [164112] l1cache instance 0 hash error in bogo-ops counter and run flag, 2376

```

```

stress-ng: fail: [164112] l1cache instance 1 corrupted bogo-ops counter, 300 vs 0
stress-ng: fail: [164112] l1cache instance 1 hash error in bogo-ops counter and run flag, 2376
stress-ng: fail: [164112] l1cache instance 2 corrupted bogo-ops counter, 300 vs 0
info: 5 failures reached, aborting stress process
stress-ng: fail: [164112] l1cache instance 2 hash error in bogo-ops counter and run flag, 2376
stress-ng: fail: [164112] l1cache instance 3 corrupted bogo-ops counter, 300 vs 0
stress-ng: fail: [164112] l1cache instance 3 hash error in bogo-ops counter and run flag, 2376
stress-ng: fail: [164112] l1cache instance 4 corrupted bogo-ops counter, 300 vs 0
stress-ng: fail: [164112] l1cache instance 4 hash error in bogo-ops counter and run flag, 2376
stress-ng: fail: [164112] l1cache instance 5 corrupted bogo-ops counter, 300 vs 0
stress-ng: fail: [164112] l1cache instance 5 hash error in bogo-ops counter and run flag, 2376
stress-ng: fail: [164112] l1cache instance 6 corrupted bogo-ops counter, 300 vs 0
stress-ng: fail: [164112] l1cache instance 6 hash error in bogo-ops counter and run flag, 2376
stress-ng: fail: [164112] l1cache instance 7 corrupted bogo-ops counter, 300 vs 0
stress-ng: fail: [164112] l1cache instance 7 hash error in bogo-ops counter and run flag, 2376
stress-ng: fail: [164112] metrics-check: stressor metrics corrupted, data is compromised
stress-ng: info: [164132] setting to a 30 second run per stressor
stress-ng: info: [164132] dispatching hogs: 8 l1cache
stress-ng: info: [164133] stress-ng-l1cache: l1cache: size: 32.0K, sets: 400, ways: 8, line si
stress-ng: info: [164132] successful run completed in 30.55s
stress-ng: info: [164132] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164132]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [164132] l1cache        111600     30.39    237.40      1.17    3671.66
stress-ng: info: [164216] setting to a 30 second run per stressor
stress-ng: info: [164216] dispatching hogs: 8 l1cache
stress-ng: info: [164217] stress-ng-l1cache: l1cache: size: 32.0K, sets: 500, ways: 8, line si
stress-ng: info: [164216] successful run completed in 30.55s
stress-ng: info: [164216] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164216]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [164216] l1cache        125000     30.28    235.22      1.44    4128.48
stress-ng: info: [164302] setting to a 30 second run per stressor
stress-ng: info: [164302] dispatching hogs: 8 l1cache
stress-ng: info: [164303] stress-ng-l1cache: l1cache: size: 32.0K, sets: 600, ways: 8, line si
stress-ng: info: [164302] unsuccessful run completed in 1.34s
stress-ng: info: [164302] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164302]                                (secs)   (secs)   (secs)   (real time)
stress-ng: info: [164302] l1cache        4800       0.00     0.00     0.00     0.00
stress-ng: fail: [164302] l1cache instance 0 corrupted bogo-ops counter, 600 vs 0
stress-ng: fail: [164302] l1cache instance 0 hash error in bogo-ops counter and run flag, 2180
stress-ng: fail: [164302] l1cache instance 1 corrupted bogo-ops counter, 600 vs 0
stress-ng: fail: [164302] l1cache instance 1 hash error in bogo-ops counter and run flag, 2180
stress-ng: fail: [164302] l1cache instance 2 corrupted bogo-ops counter, 600 vs 0
info: 5 failures reached, aborting stress process
stress-ng: fail: [164302] l1cache instance 2 hash error in bogo-ops counter and run flag, 2180
stress-ng: fail: [164302] l1cache instance 3 corrupted bogo-ops counter, 600 vs 0
stress-ng: fail: [164302] l1cache instance 3 hash error in bogo-ops counter and run flag, 2180
stress-ng: fail: [164302] l1cache instance 4 corrupted bogo-ops counter, 600 vs 0
stress-ng: fail: [164302] l1cache instance 4 hash error in bogo-ops counter and run flag, 2180

```

```

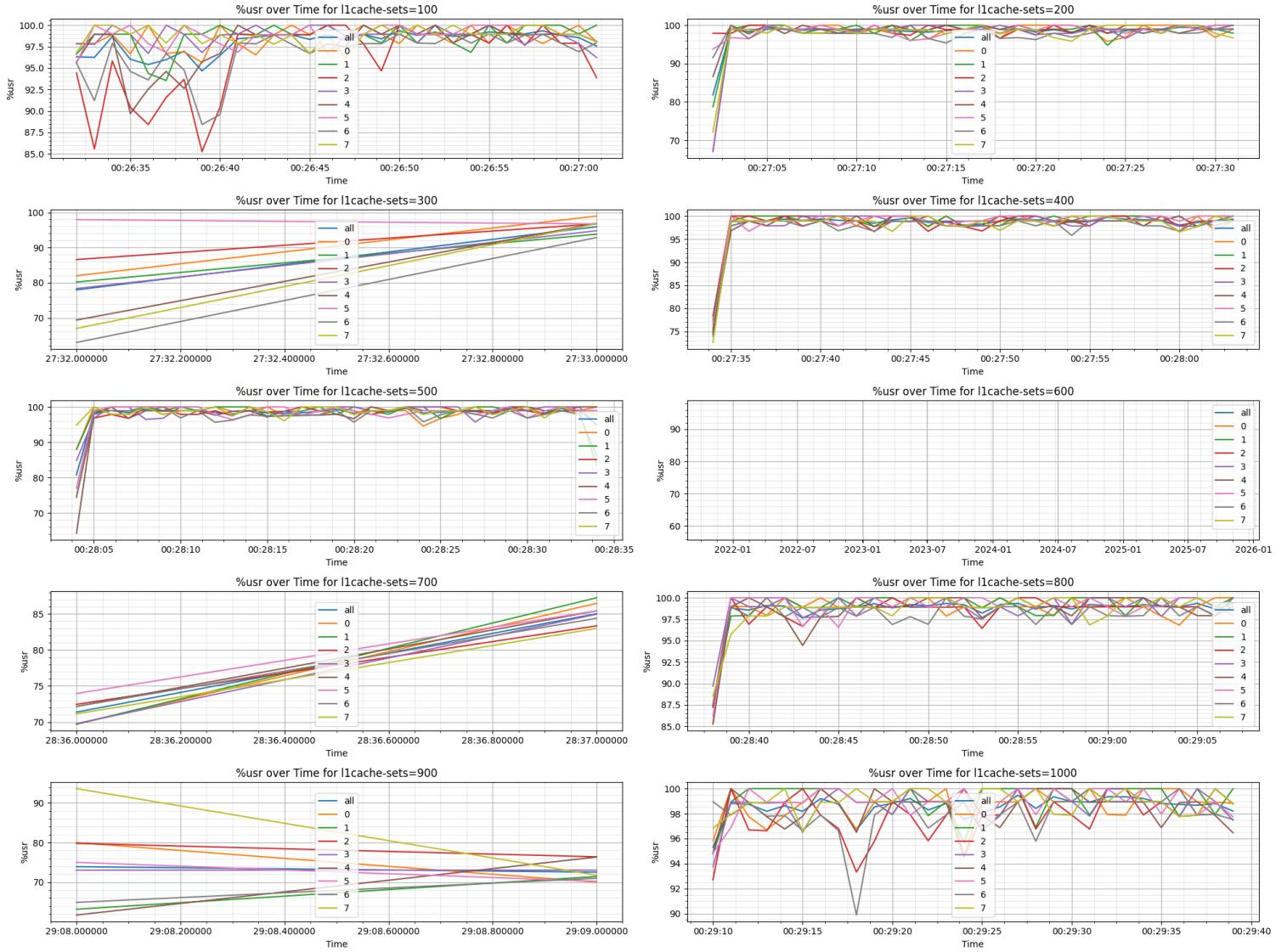
stress-ng: fail: [164302] l1cache instance 5 corrupted bogo-ops counter, 600 vs 0
stress-ng: fail: [164302] l1cache instance 5 hash error in bogo-ops counter and run flag, 2180
stress-ng: fail: [164302] l1cache instance 6 corrupted bogo-ops counter, 600 vs 0
stress-ng: fail: [164302] l1cache instance 6 hash error in bogo-ops counter and run flag, 2180
stress-ng: fail: [164302] l1cache instance 7 corrupted bogo-ops counter, 600 vs 0
stress-ng: fail: [164302] l1cache instance 7 hash error in bogo-ops counter and run flag, 2180
stress-ng: fail: [164302] metrics-check: stressor metrics corrupted, data is compromised
stress-ng: info: [164323] setting to a 30 second run per stressor
stress-ng: info: [164323] dispatching hogs: 8 l1cache
stress-ng: info: [164324] stress-ng-l1cache: l1cache: size: 32.0K, sets: 700, ways: 8, line si
stress-ng: info: [164323] unsuccessful run completed in 1.37s
stress-ng: info: [164323] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164323]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [164323] l1cache          5600      0.00      0.00      0.00      0.00
stress-ng: fail: [164323] l1cache instance 0 corrupted bogo-ops counter, 700 vs 0
stress-ng: fail: [164323] l1cache instance 0 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] l1cache instance 1 corrupted bogo-ops counter, 700 vs 0
stress-ng: fail: [164323] l1cache instance 1 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] l1cache instance 2 corrupted bogo-ops counter, 700 vs 0
info: 5 failures reached, aborting stress process
stress-ng: fail: [164323] l1cache instance 2 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] l1cache instance 3 corrupted bogo-ops counter, 700 vs 0
stress-ng: fail: [164323] l1cache instance 3 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] l1cache instance 4 corrupted bogo-ops counter, 700 vs 0
stress-ng: fail: [164323] l1cache instance 4 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] l1cache instance 5 corrupted bogo-ops counter, 700 vs 0
stress-ng: fail: [164323] l1cache instance 5 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] l1cache instance 6 corrupted bogo-ops counter, 700 vs 0
stress-ng: fail: [164323] l1cache instance 6 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] l1cache instance 7 corrupted bogo-ops counter, 700 vs 0
stress-ng: fail: [164323] l1cache instance 7 hash error in bogo-ops counter and run flag, 8803
stress-ng: fail: [164323] metrics-check: stressor metrics corrupted, data is compromised
stress-ng: info: [164343] setting to a 30 second run per stressor
stress-ng: info: [164343] dispatching hogs: 8 l1cache
stress-ng: info: [164344] stress-ng-l1cache: l1cache: size: 32.0K, sets: 800, ways: 8, line si
stress-ng: info: [164343] successful run completed in 30.47s
stress-ng: info: [164343] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164343]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [164343] l1cache          217600     30.25    236.04     1.24    7193.61
stress-ng: info: [164430] setting to a 30 second run per stressor
stress-ng: info: [164430] dispatching hogs: 8 l1cache
stress-ng: info: [164431] stress-ng-l1cache: l1cache: size: 32.0K, sets: 900, ways: 8, line si
stress-ng: info: [164430] unsuccessful run completed in 1.44s
stress-ng: info: [164430] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164430]                                (secs)    (secs)    (secs)    (real time)
stress-ng: info: [164430] l1cache          7200      0.00      0.00      0.00      0.00
stress-ng: fail: [164430] l1cache instance 0 corrupted bogo-ops counter, 900 vs 0
stress-ng: fail: [164430] l1cache instance 0 hash error in bogo-ops counter and run flag, 3102

```

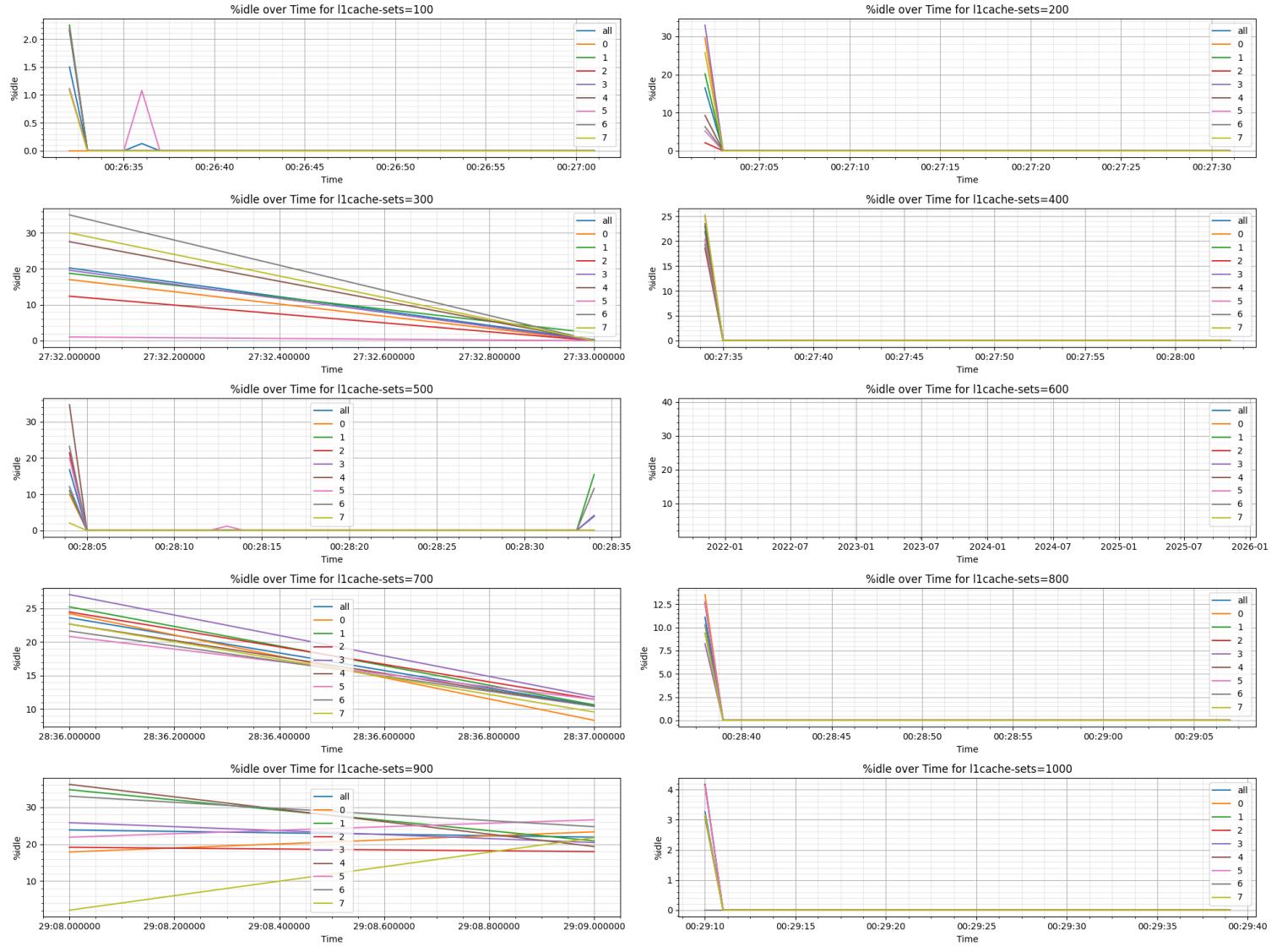
```
stress-ng: fail: [164430] l1cache instance 1 corrupted bogo-ops counter, 900 vs 0
stress-ng: fail: [164430] l1cache instance 1 hash error in bogo-ops counter and run flag, 3102
stress-ng: fail: [164430] l1cache instance 2 corrupted bogo-ops counter, 900 vs 0
info: 5 failures reached, aborting stress process
stress-ng: fail: [164430] l1cache instance 2 hash error in bogo-ops counter and run flag, 3102
stress-ng: fail: [164430] l1cache instance 3 corrupted bogo-ops counter, 900 vs 0
stress-ng: fail: [164430] l1cache instance 3 hash error in bogo-ops counter and run flag, 3102
stress-ng: fail: [164430] l1cache instance 4 corrupted bogo-ops counter, 900 vs 0
stress-ng: fail: [164430] l1cache instance 4 hash error in bogo-ops counter and run flag, 3102
stress-ng: fail: [164430] l1cache instance 5 corrupted bogo-ops counter, 900 vs 0
stress-ng: fail: [164430] l1cache instance 5 hash error in bogo-ops counter and run flag, 3102
stress-ng: fail: [164430] l1cache instance 6 corrupted bogo-ops counter, 900 vs 0
stress-ng: fail: [164430] l1cache instance 6 hash error in bogo-ops counter and run flag, 3102
stress-ng: fail: [164430] l1cache instance 7 corrupted bogo-ops counter, 900 vs 0
stress-ng: fail: [164430] l1cache instance 7 hash error in bogo-ops counter and run flag, 3102
stress-ng: fail: [164430] metrics-check: stressor metrics corrupted, data is compromised
stress-ng: info: [164450] setting to a 30 second run per stressor
stress-ng: info: [164450] dispatching hogs: 8 l1cache
stress-ng: info: [164451] stress-ng-l1cache: l1cache: size: 32.0K, sets: 1000, ways: 8, line s
stress-ng: info: [164450] successful run completed in 30.66s
stress-ng: info: [164450] stressor      bogo ops real time  usr time  sys time  bogo ops/s
stress-ng: info: [164450]                      (secs)      (secs)      (secs)      (real time)
stress-ng: info: [164450] l1cache           268000     30.42     236.70      1.45      8810.88
```

Графики...графики...графики...

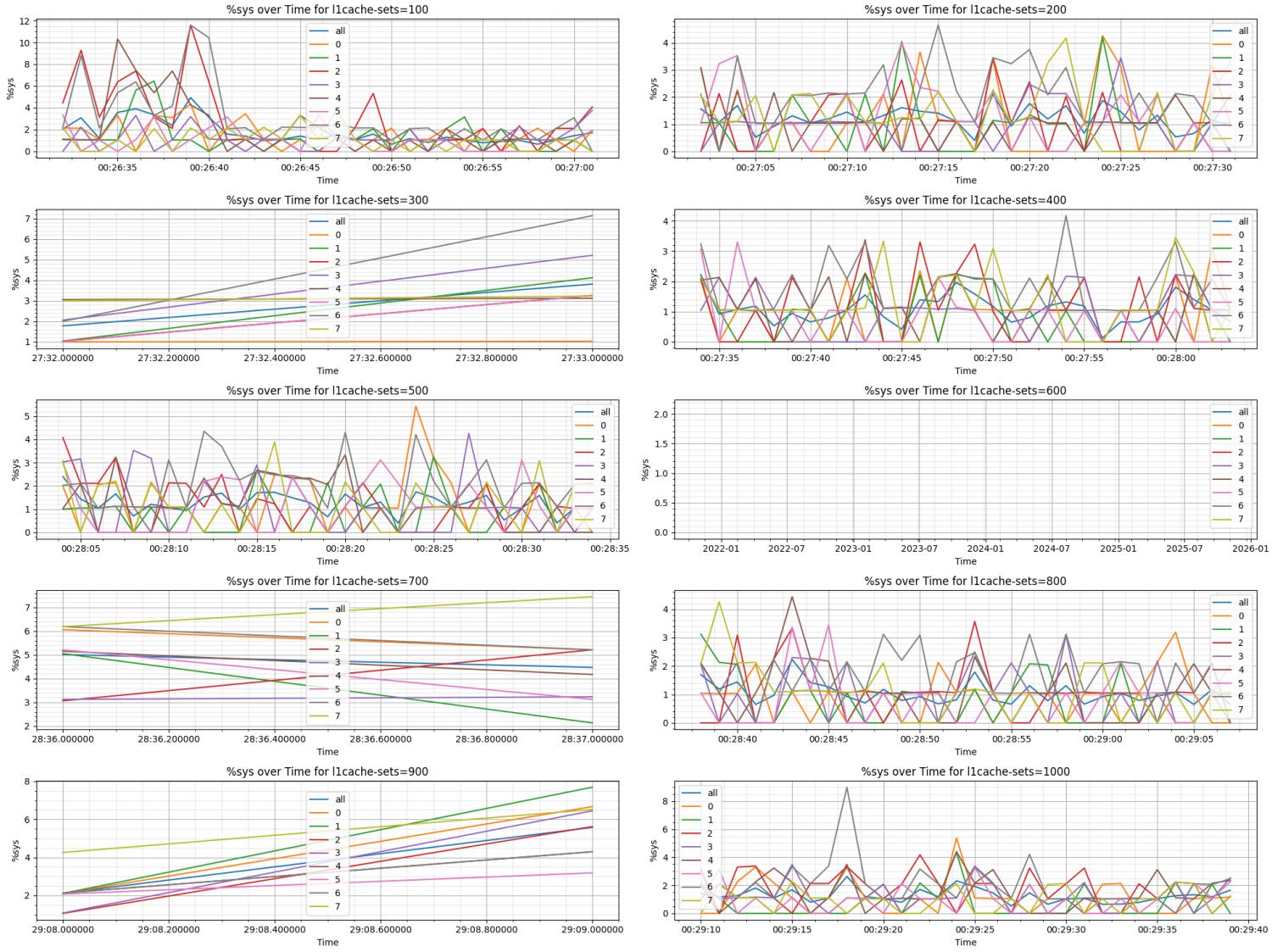
user time



idle time

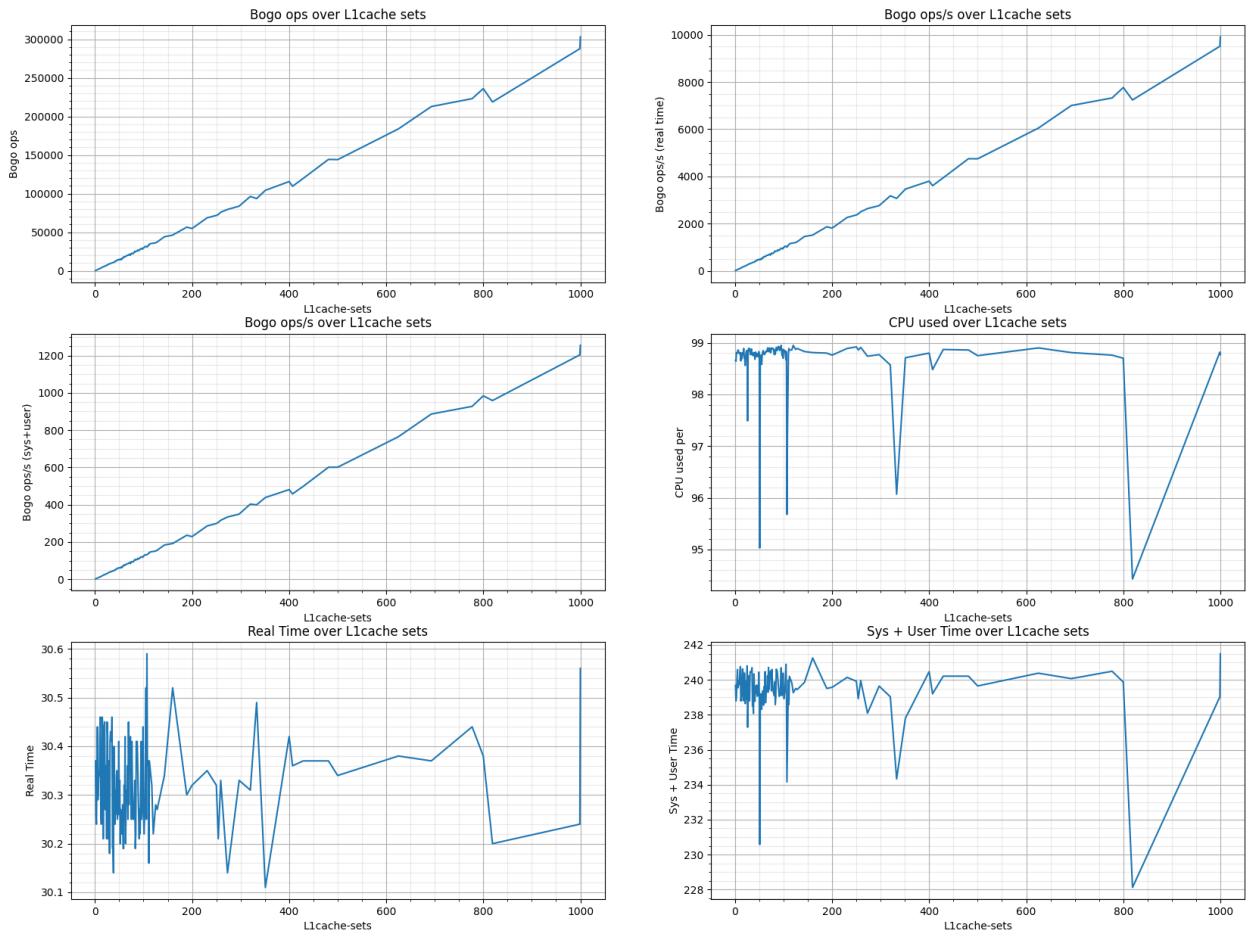


sys time



Вывод: чем больше set, тем выше bogo-ops, но при этом увеличивается вероятность того, что тест сломается. (приведена лучшая попытка)

Еще посмотрим графики *bogo ops*, чтобы проверить наш вывод



Видим, что вывод оказался верным, что чем больше set, тем выше bogo-ops, но при этом увеличивается вероятность того, что тест сломается.

2.3 cache-fence

Из документации stress-ng:

--cache-fence: *force write serialization on each store operation (x86 only). This is a no-op for non-x86 architectures.*

```
mpstat -P ALL 1 &
mpstat_pid=$!
stress-ng --l1cache 8 --l1cache-sets 1000 --cache-fence --metrics --timeout 30 > /dev/null &
stress_ng_pid=$! # Get the process ID of stress-ng
wait $stress_ng_pid # Wait for stress-ng to finish before moving to the next iteration
kill $mpstat_pid

./l1cache-fence.zsh > ~/output.txt
stress-ng: info: [169289] setting to a 30 second run per stressor
stress-ng: info: [169289] dispatching hogs: 8 l1cache
```

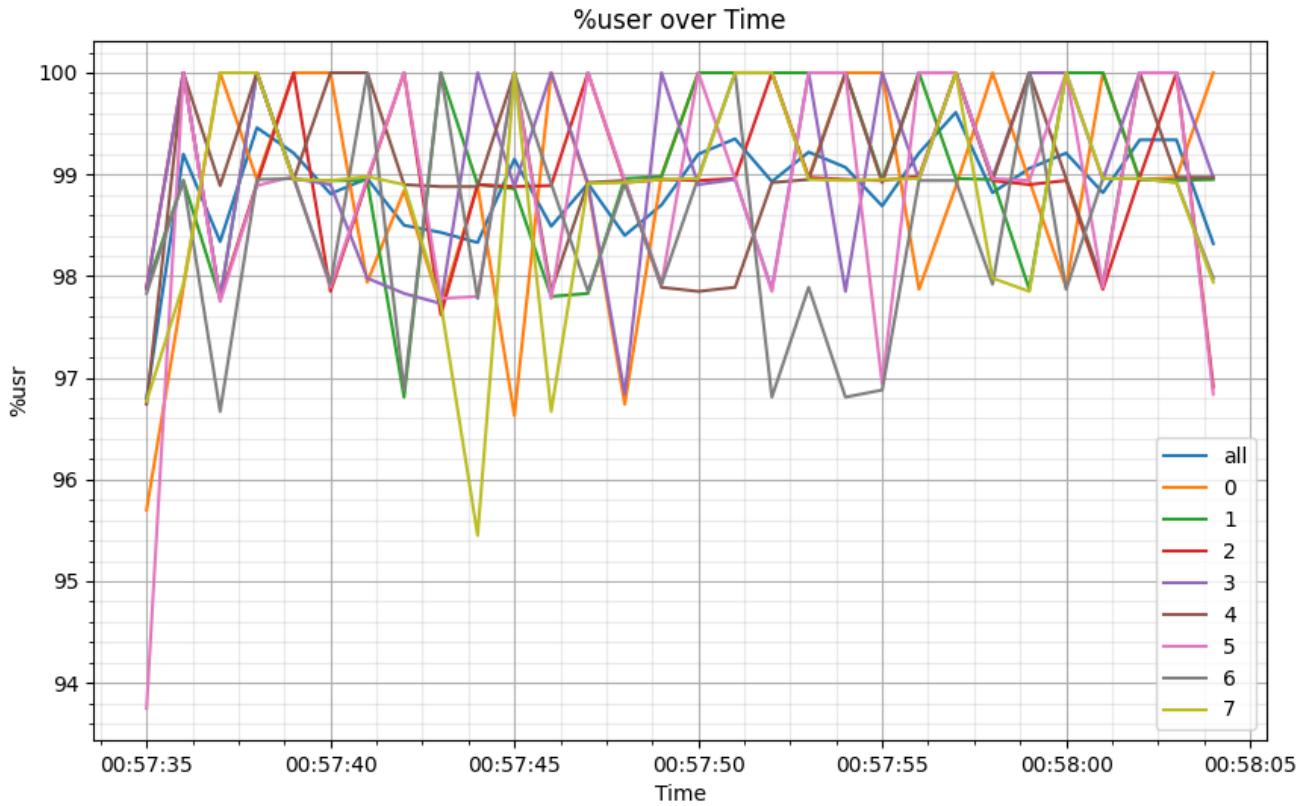
```

stress-ng: info: [169290] stress-ng-l1cache: l1cache: size: 32.0K, sets: 1000, ways: 8, line s
stress-ng: info: [169289] successful run completed in 30.38s
stress-ng: info: [169289] stressor      bogo ops real time   usr time   sys time   bogo ops/s
stress-ng: info: [169289]                               (secs)     (secs)     (secs)     (real time)
stress-ng: info: [169289] l1cache                283000     30.16     235.41      1.12      9382.93

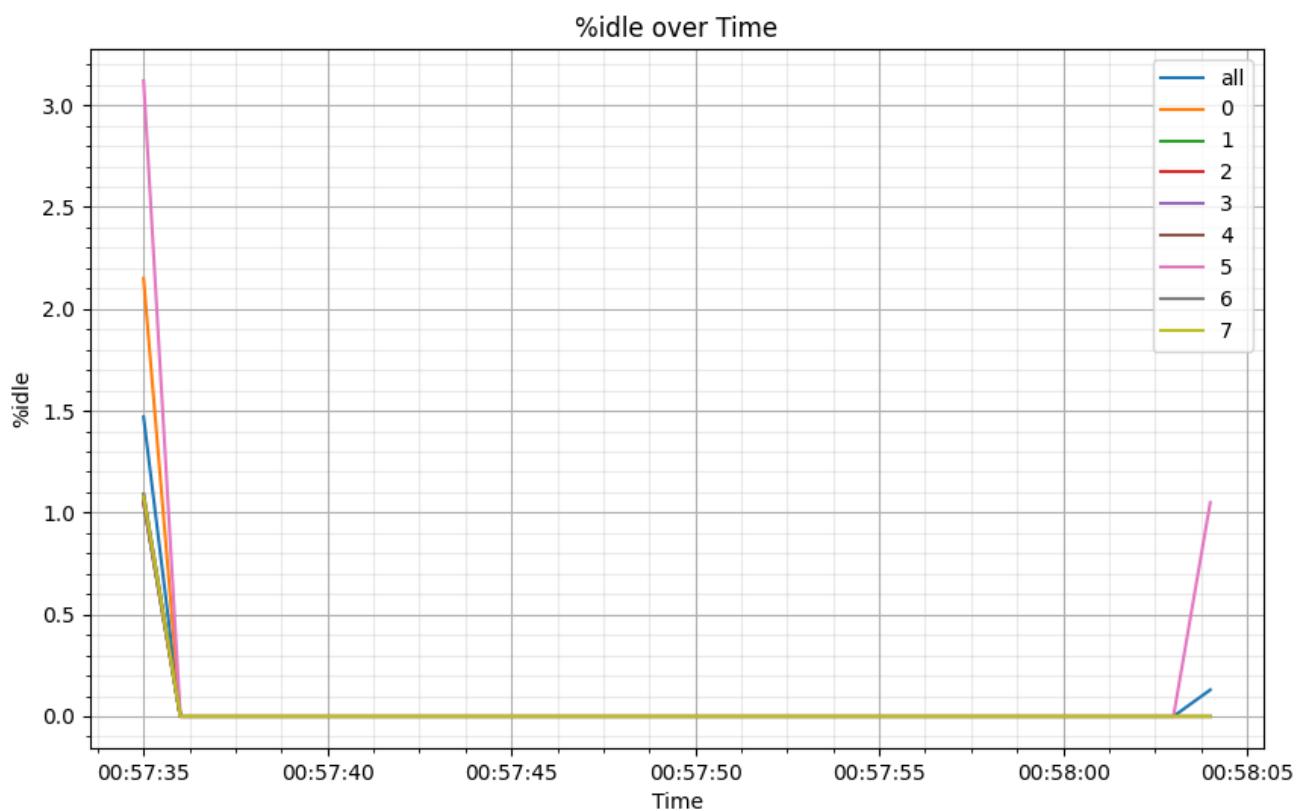
```

Графики...жирафики

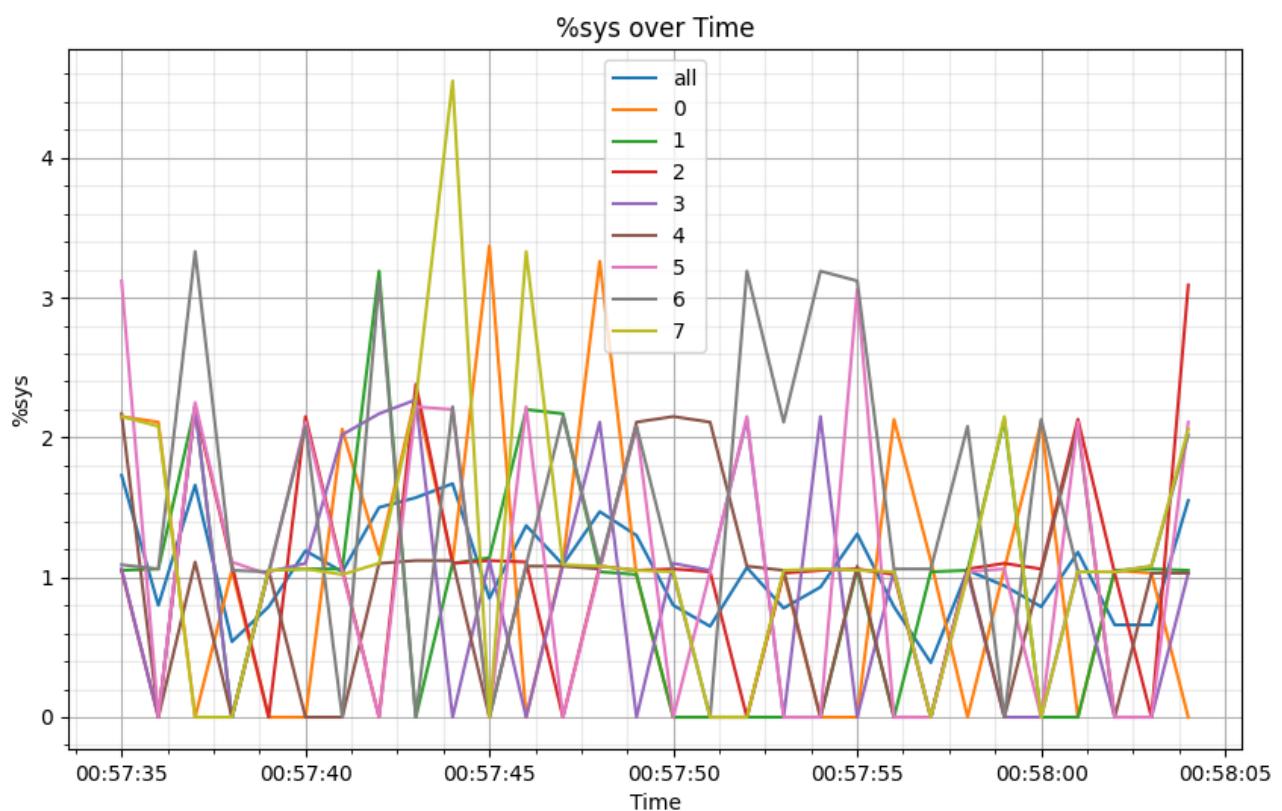
user time



idle time



sys time



Вывод: --cache-fence дает нам небольшой выигрыш по производительности.

3 IO testing

Выданные параметры: [ioport,io-uring] Система до использования stress-ng:

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	PRIOS	USER	DISK READ	DISK WRITE	SWAPIN	IO	GRAPH[IO]	COMMAND		
173767	be/4	root	0,00 B/s	0,00 B/s	0,00 %	0,00 %	kworker/u64:1-events_power_	
1	be/4	root	0,00 B/s	0,00 B/s	0,00 %	0,00 %	systemd	
2	be/4	root	0,00 B/s	0,00 B/s	0,00 %	0,00 %	kthreadd	
3	be/0	root	0,00 B/s	0,00 B/s	0,00 %	0,00 %	rcu_gp	
4	be/0	root	0,00 B/s	0,00 B/s	0,00 %	0,00 %	rcu_par_gp	

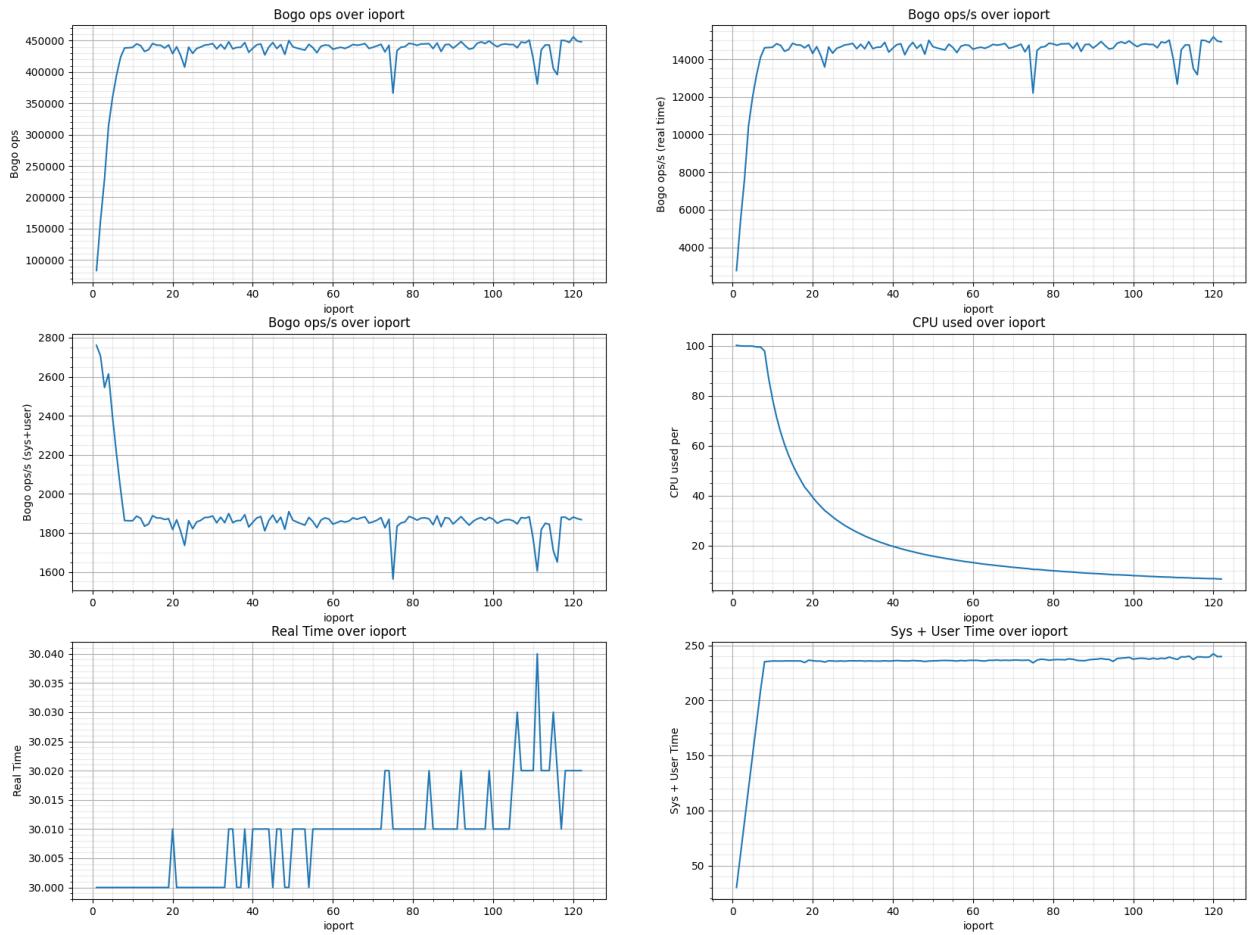
3.1 ioport

Из документации stress-ng:

--ioport N: start N workers than perform bursts of 16 reads and 16 writes of ioport 0x80 (x86 Linux systems only). I/O performed on x86 platforms on port 0x80 will cause delays on the CPU performing the I/O.

Сначала найдем оптимальное значение воркеров. Для этого напишем скрипт:

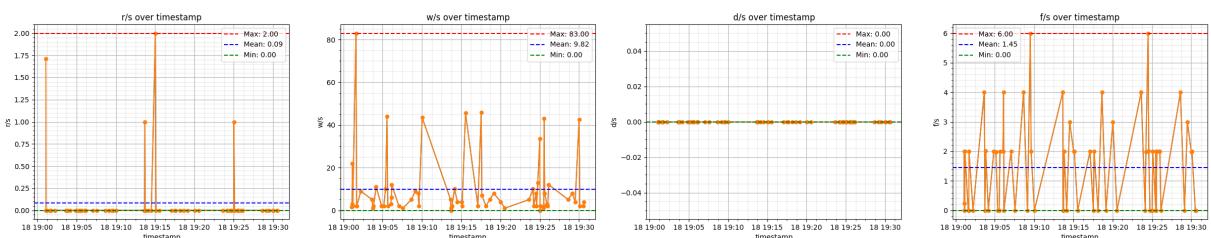
```
1 #!/bin/zsh
echo "sched; bogo_ops; real_time; usr_time; sys_time; bogo_ops/s_real; bogo_ops/s_user+sys; CPU_used_per"
3 for i in {1..8192..1}; do
    sudo stress-ng --ioport $i --metrics --timeout 30 --stdout |
5 awk -v var="$i" '/[\[\]\d]+[ ]+ioport/{printf var;"$5";"$6";"$7";"$8";"$9";"$10";"$11"\n}' ,
done
```



Как мы видим, к 8 воркерам мы достигаем максимального количества bogo ops и далее они не растут. Проведем анализ нагрузки с помощью *iostat* для этого напишем скрипт:

```
#!/bin/zsh
2 iostat -xzdtk -o JSON 1 sda >> ./ioport-iostat.json &
iostat_pid=$!
4 sudo stress-ng --ioport 8 --metrics --timeout 30m > /dev/null &
stress_ng_pid=$! # Get the process ID of stress-ng
6 wait $stress_ng_pid
kill $iostat_pid
```

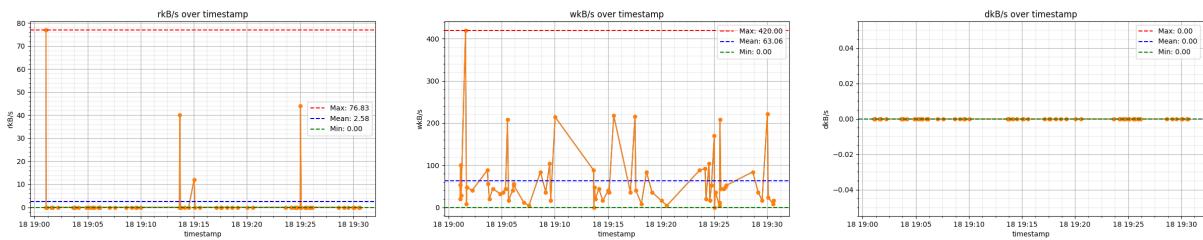
r/s, w/s, d/s, f/s: запросов на чтение, запись, усечение и синхронизацию, выполняемых дисковым устройством в секунду (после слияния);



Видим, что запросов на чтение значительно меньше и они реже, чем запросы на запись. Так же у нас не

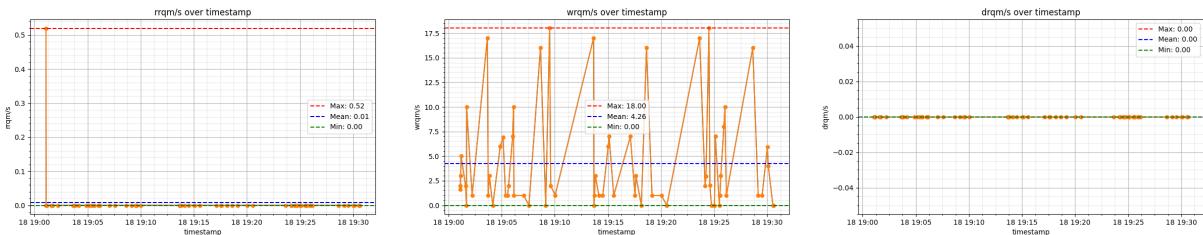
было запросов на освобождение блоков на диске (команда ATA TRIM). Мы можем заметить запросы на синхронизацию, которых сильно меньше чем запросов на запись, но они чаще.

rkB/s, wkB/s, dkB/s: объем чтения, записи и усечения в килобайтах в секунду;



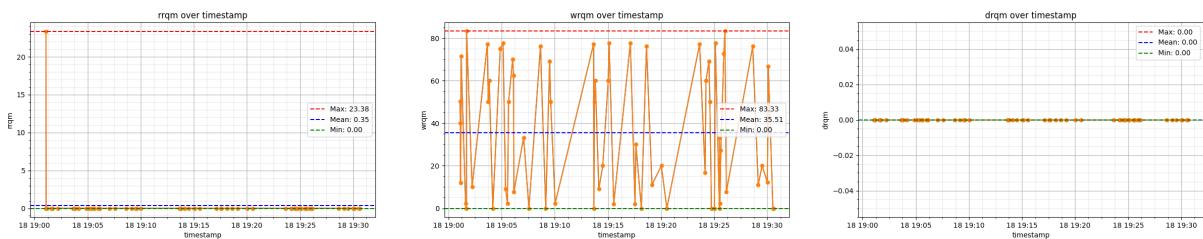
Из графиков видно, что мы читаем меньше, чем записываем. Читали мы максимум чуть меньше 80 Кб/с, записывали максимум чуть больше 400 Кб/с и около 60 Кб/с в среднем.

rrqm/s, rqm/s, drqm/s: количество запросов в секунду на чтение, запись и усечение, добавленных в очередь и объединенных



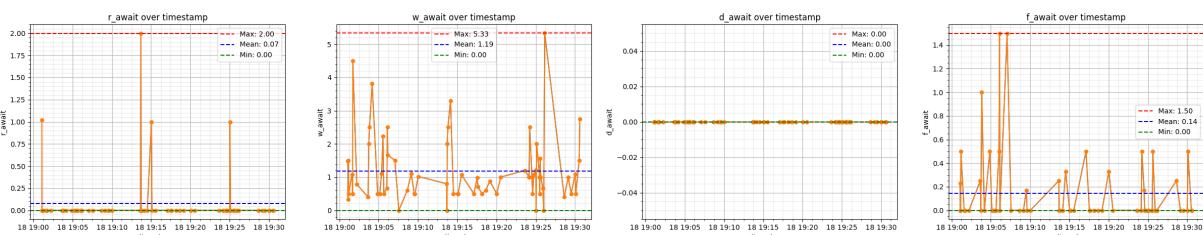
Видим, что запросы на чтения практически не объединялись, в то же время около 4% запросов из очереди на запись были объединены.

%rrqm, %wrqm, %drqm: количество запросов на чтение, запись и усечение, добавленных в очередь и объединенных, в процентах от общего количества запросов каждого типа



Вывод, тот же что и выше.

r_await, w_await, d_await, f_await: среднее время отклика для чтения, записи, усечения и синхронизации, включая время ожидания в очереди драйвера и время отклика устройства (миллисекунд);



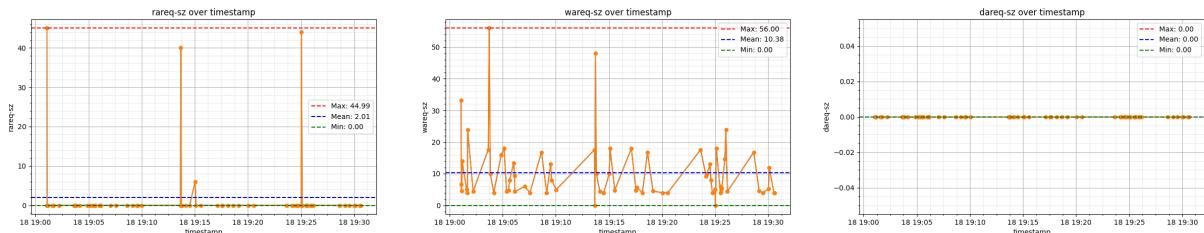
Видим, что время отклика на чтение максимум было 2 миллисекунды (1 раз), 1 миллисекунда (3 раза) и

практически моментально все остальное время.

Запись максимум было около 5 миллисекунд, в среднем около 1 миллисекунды.

Синхронизация в среднем имела отклик около 0.14 миллисекунд.

reduq-sz, wareq-sz, dareq-sz: средний размер чтения, записи и усечения (килобайт).

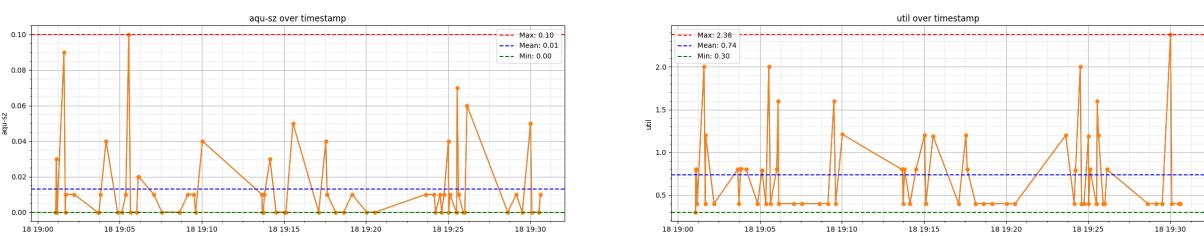


Видим, что мы читали редко и сразу много около 40 килобайт.

Записывали мы частно, но по немногу (в среднем по 10 килобайт).

aqu-sz: средняя длина очереди запросов, которые были отправлены устройству.

%util: процент прошедшего времени, в течение которого устройству были отправлены запросы ввода-вывода (использование полосы пропускания устройства).



Из графиков видно, что в среднем очередь запросов была небольшой. Также полоса пропускания была большей части свободна. **Вывод:** зачем я это вообще делал? В документации ведь ясно сказано, что мы не нагружаем диск, а только обращаемся к ioport 0x80, тем самым нагружаем только CPU. И все это чтение и запись это шум от других программ.

3.2 io-uring

-io-uring N: start N workers that perform iovec write and read I/O operations using the Linux io-uring interface. On each bogo-loop 1024×512 byte writes and $1024 \times$ reads are performed on a temporary file.

К сожалению, из-за того, что *io-uring* у меня постоянно подал с ошибками, или выдавал нулевые bogo ops, тщательно проанализировать его не получилось.

Так как запустить тест с разным количеством io-uring, чтобы посмотреть зависимость от bogo ops у меня не получилось, то привожу пример с обычно наиболее оптимальным количеством воркеров для моего устройства.

```

~ sudo stress-ng --io-uring 8 --metrics --timeout 1m
[sudo] password for alexey:
stress-ng: info: [18081] setting to a 60 second run per stressor
stress-ng: info: [18081] dispatching hogs: 8 io-uring
stress-ng: info: [18081] successful run completed in 60.03s (1 min, 0.03 secs)
stress-ng: info: [18081] stressor      bogo ops real time   usr time   sys time   bogo ops/s      bogo ops/s CPU used p
er
stress-ng: info: [18081]                                     (secs)     (secs)     (secs)     (real time) (usr+sys time) instance (%)
stress-ng: info: [18081] io-uring          1951470      60.03      5.20      217.07      32508.56      8779.73      46.
28

```

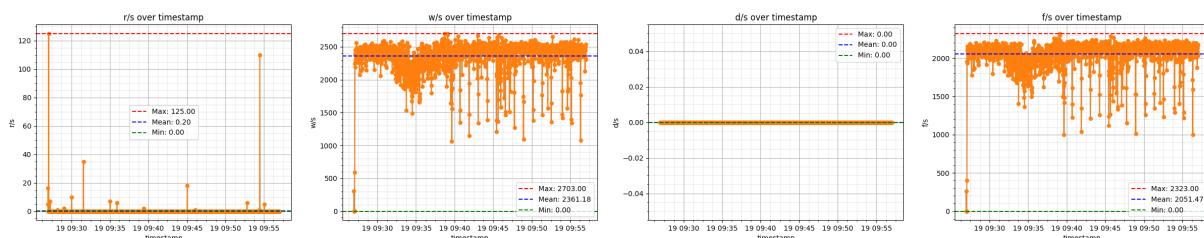
А теперь посмотрим статистики iostat, как для ioport.

```

1#!/bin/zsh
2iostat -xzdtk -o JSON 1 sda >> ./io-uring.json &
3iostat_pid=$!
4sudo stress-ng --io-uring 8 --metrics --timeout 30m > /dev/null &
5stress_ng_pid=$! # Get the process ID of stress-ng
6wait $stress_ng_pid
7kill $iostat_pid

```

r/s, w/s, d/s, f/s: запросов на чтение, запись, усечение и синхронизацию, выполняемых дисковым устройством в секунду (после слияния);



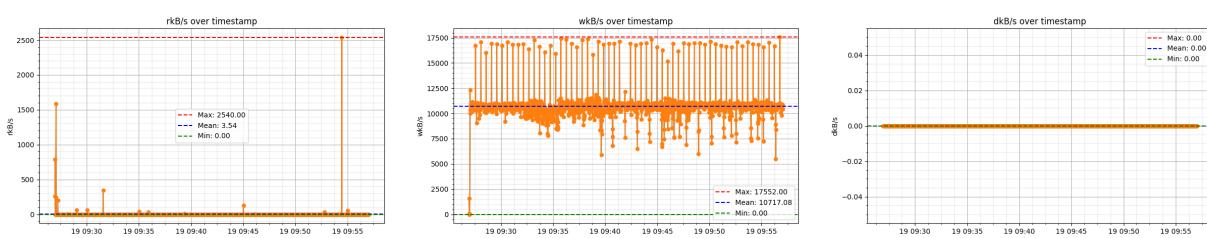
Видим, что читаем мы редко и сразу много.

Записываем мы часто и много (в среднем около 2300 запросов в секунду).

Запросов на освобождение блоков на диске нету.

Количество запросов на синхронизацию, чуть меньше, чем запросов на запись, но видно, что между ними есть зависимость.

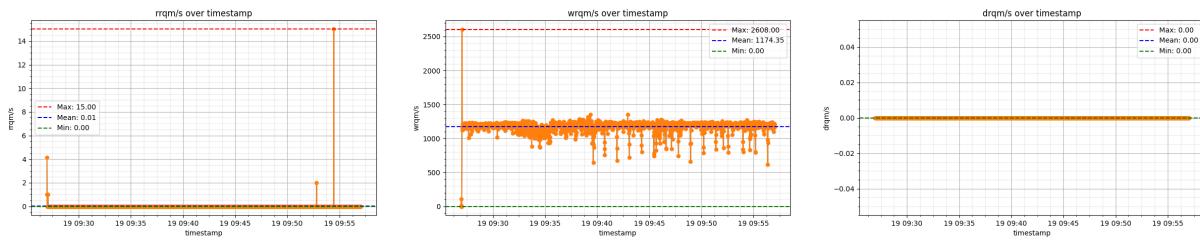
rkB/s, wkB/s, dkB/s: объем чтения, записи и усечения в килобайтах в секунду;



Видим, что при большом количестве запросов скорость чтения может падать.

Записываем, мы в основном около 10000, либо 17500 килобайт в секунду, что намного быстрее, чем читаем

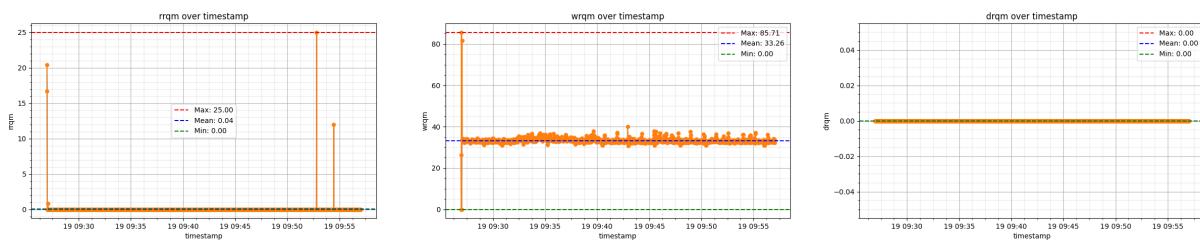
rrqm/s, wrqm/s, drqm/s: количество запросов в секунду на чтение, запись и усечение, добавленных в очередь и обработанных, в процентах от общего количества запросов каждого типа



Видим, что в промежутки большого количества запросов на чтение, некоторые из запросов начинают объединяться.

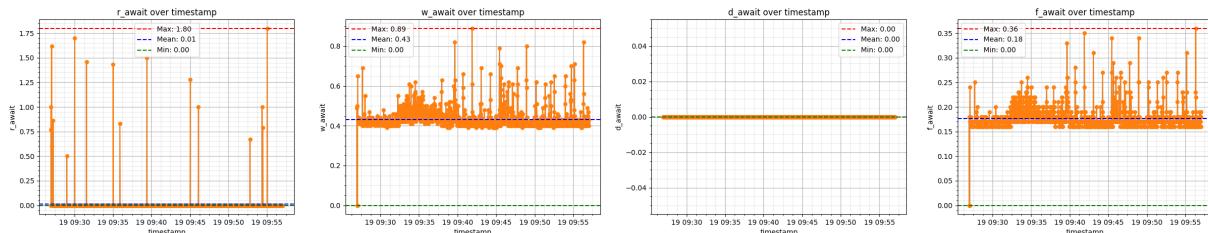
При записи в среднем около 1100 запросов объединялись

%rrqm, %wrqm, %drqm: количество запросов на чтение, запись и усечение, добавленных в очередь и обединенных, в процентах от общего количества запросов каждого типа



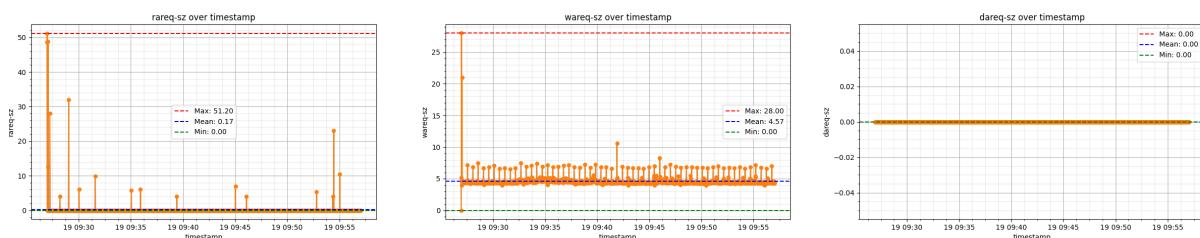
Тоже что и выше, но в процентах от общего количества запросов каждого типа.

r_await, w_await, d_await, f_await: среднее время отклика для чтения, записи, усечения и синхронизации, включая время ожидания в очереди драйвера и время отклика устройства (миллисекунд);



Видим, что время отклика не зависит от объема чтения и больше чем при записи.

reduq-sz, wareq-sz, dareq-sz: средний размер чтения, записи и усечения (килобайт).

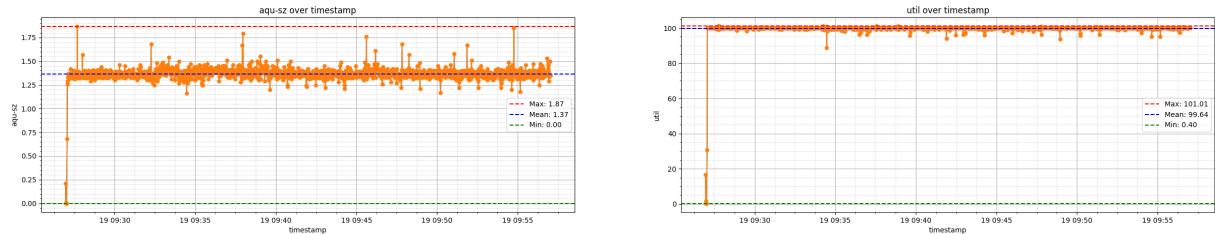


Видим, что нет прямой зависимости между количеством запросов и размером прочитанного (за меньшее число запросов можно прочитать больше).

Записываем мы часто и небольшими порциями всего в среднем по 5 килобайт.

nquoteaque-sz средняя длина очереди запросов, которые были отправлены устройству.

%util: процент прошедшего времени, в течение которого устройству были отправлены запросы ввода-вывода (использование полосы пропускания устройства). Насыщение устройства происходит, когда это значение близко к 100% для устройств, обслуживающих запросы последовательно. Но для устройств, обслуживающих запросы параллельно, таких как RAID-массивы и современные твердотельные накопители, это число не отражает их ограничений по производительности.



Видим, что в среднем в нашей очереди от 1 до 2 запросов.

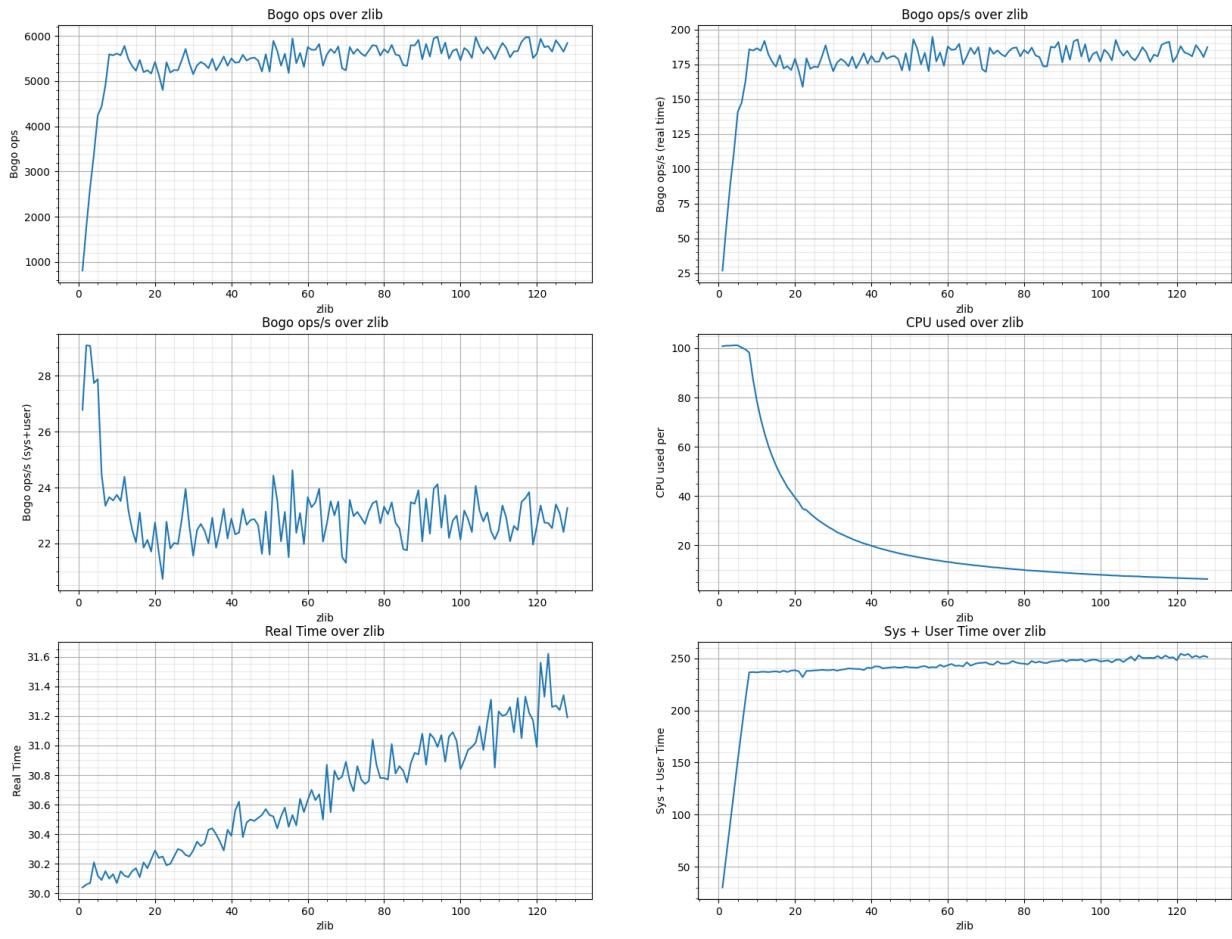
Мы видим, что %util равен 100%, но так как у меня стоит SSD, которое может обслуживать запросы параллельно, то это число ни о чем не говорит.

4 Memory testing

Выданные параметры: [fork-vm,zlib-mem-level]

4.1 zlib

-zlib N: start N workers compressing and decompressing random data using zlib. Each worker has two processes, one that compresses random data and pipes it to another process that decompresses the data. This stressor exercises CPU, cache and memory.



Видим, что 8 воркеров - это оптимальное число.

4.2 zlib-mem-level

-zlib-mem-level L: specify the reserved compression state memory for zlib. Default is 8. Values: 1 = minimum memory usage 9 = maximum memory usage

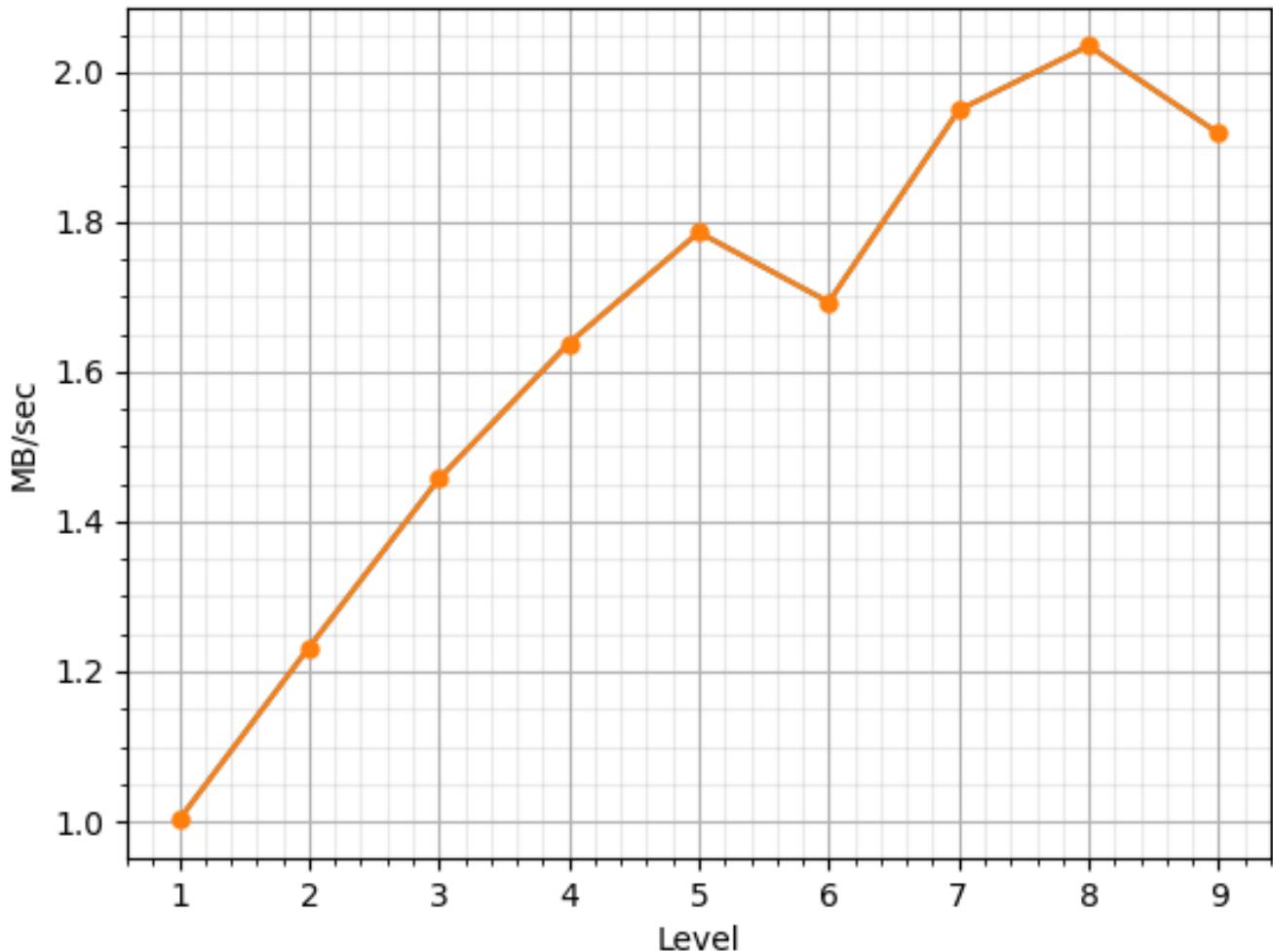
Переберем все значения параметра:

```

1 #!/bin/zsh
for i in {1..9..1}; do
3   for j in {1..10..1}; do
      a='sudo stress-ng --zlib 1 --zlib-mem-level $i --metrics --timeout 10 --stdout |
5     awk '/ratio/,'
     echo "$i;${a:73:95}"
    done
done

```

Level over compression rate

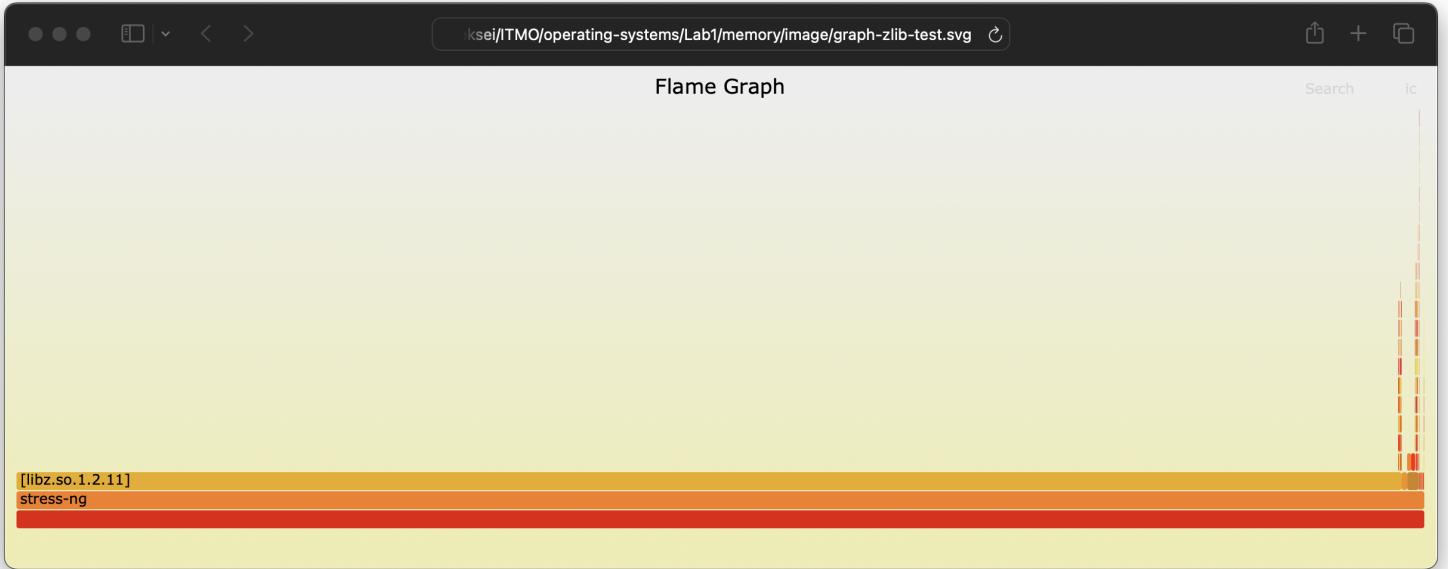


Видим, что с увеличением уровня увеличивается compression state достигая максимального значения при 8.

Посмотрим на Flamegraph процесса zlib.

```
~/FlameGraph master ?6
sudo perf record -F 99 -g stress-ng --zlib 8 --zlib-mem-level 8 --metrics --timeout 10
stress-ng: info: [79415] setting to a 10 second run per stressor
stress-ng: info: [79415] dispatching hogs: 8 zlib
stress-ng: info: [79425] stress-ng-zlib: instance 4: compression ratio: 60.39% (1.64 MB/sec)
stress-ng: info: [79417] stress-ng-zlib: instance 0: compression ratio: 59.81% (1.67 MB/sec)
stress-ng: info: [79423] stress-ng-zlib: instance 3: compression ratio: 53.76% (1.62 MB/sec)
stress-ng: info: [79418] stress-ng-zlib: instance 1: compression ratio: 54.20% (1.72 MB/sec)
stress-ng: info: [79430] stress-ng-zlib: instance 7: compression ratio: 55.33% (1.63 MB/sec)
stress-ng: info: [79420] stress-ng-zlib: instance 2: compression ratio: 55.64% (1.60 MB/sec)
stress-ng: info: [79426] stress-ng-zlib: instance 5: compression ratio: 59.85% (1.52 MB/sec)
stress-ng: info: [79429] stress-ng-zlib: instance 6: compression ratio: 54.89% (1.40 MB/sec)
stress-ng: info: [79415] successful run completed in 10.35s
stress-ng: info: [79415] stressor      bogo ops   real time   usr time   sys time   bogo ops/s   bogo ops/s   CPU used per
stress-ng: info: [79415]                      (secs)     (secs)     (secs)     (real time) (usr+sys time) instance (%)
stress-ng: info: [79415] zlib           1966      10.17      79.99      0.24      193.35      24.50      98.63
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0,504 MB perf.data (7940 samples) ]

~/FlameGraph master ?6
sudo perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > graph-zlib-test.svg
```



4.3 fork-vm

-fork-vm: enable detrimental performance virtual memory advice using madvise on all pages of the forked process. Where possible this will try to set every page in the new process with using madvise MADV_MERGEABLE, MADV_WILLNEED, MADV_HUGEPAGE and MADV_RANDOM flags. Linux only.

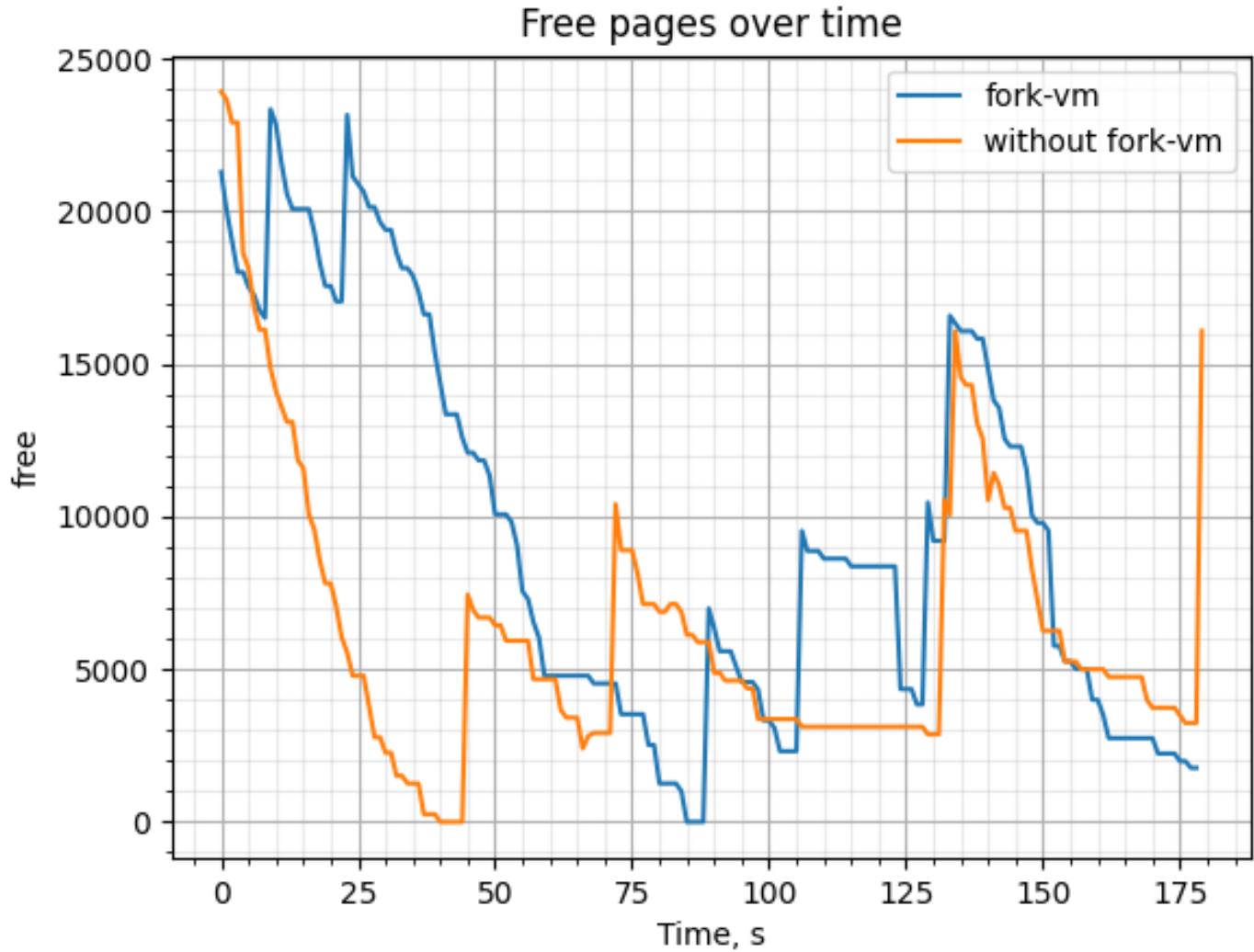
MADV_MERGEABLE: Включите объединение одинаковых страниц ядра (KSM) для страниц в диапазоне, указанном *addr* и *длиной*. Ядро регулярно сканирует те области пользовательской памяти, которые были помечены как доступные для объединения, в поисках страниц с идентичным содержимым. Они заменяются одной страницей, защищенной от записи (которая автоматически копируется, если процесс позже захочет обновить содержимое страницы). KSM объединяет только частные анонимные страницы (см. *ттар(2)*). Функция KSM предназначена для приложений, которые генерируют множество экземпляров одних и тех же данных (например, систем виртуализации, таких как *KVM*). Он может потреблять много вычислительной мощности; используйте его с осторожностью.

MADV_WILLNEED: Данные будут использоваться повторно (кэширование желательно)

MADV_HUGEPAGE: Включить функцию Transparent Huge Pages (THP) для страниц в диапазоне, заданном *addr* и *length*. В настоящее время Transparent Huge Pages работает только с частными анонимными страницами (см. *ттар(2)*). Ядро будет регулярно сканировать области, помеченные как кандидаты на огромные страницы, чтобы заменить их на огромные страницы. Ядро также будет выделять огромные страницы напрямую, если область естественным образом выровнена по размеру огромной страницы (см. *posix_memalign(2)*). Эта функция предназначена в первую очередь для приложений, использующих большие отображения данных и обращающихся к большим областям памяти за один раз (например, системы виртуализации, такие как *QEMU*). Это может привести к нерациональному расходованию памяти (например, отображение размером 2 МБ, при котором доступ осуществляется только к одному байту, приведет к использованию 2 МБ проводной памяти вместо одной страницы размером 4 КБ).

MADV_RANDOM: Доступ к данным в файле будет произвольным

Посмотрим, как изменяется количество свободных страниц виртуальной памяти с помощью *vmstat* при включенной и выключенной опции *fork-vm*;



Видим, что особого влияния на количество свободных страниц виртуальной памяти нет.

5 Network testing

Выданные параметры: [netdev, netlink-task]

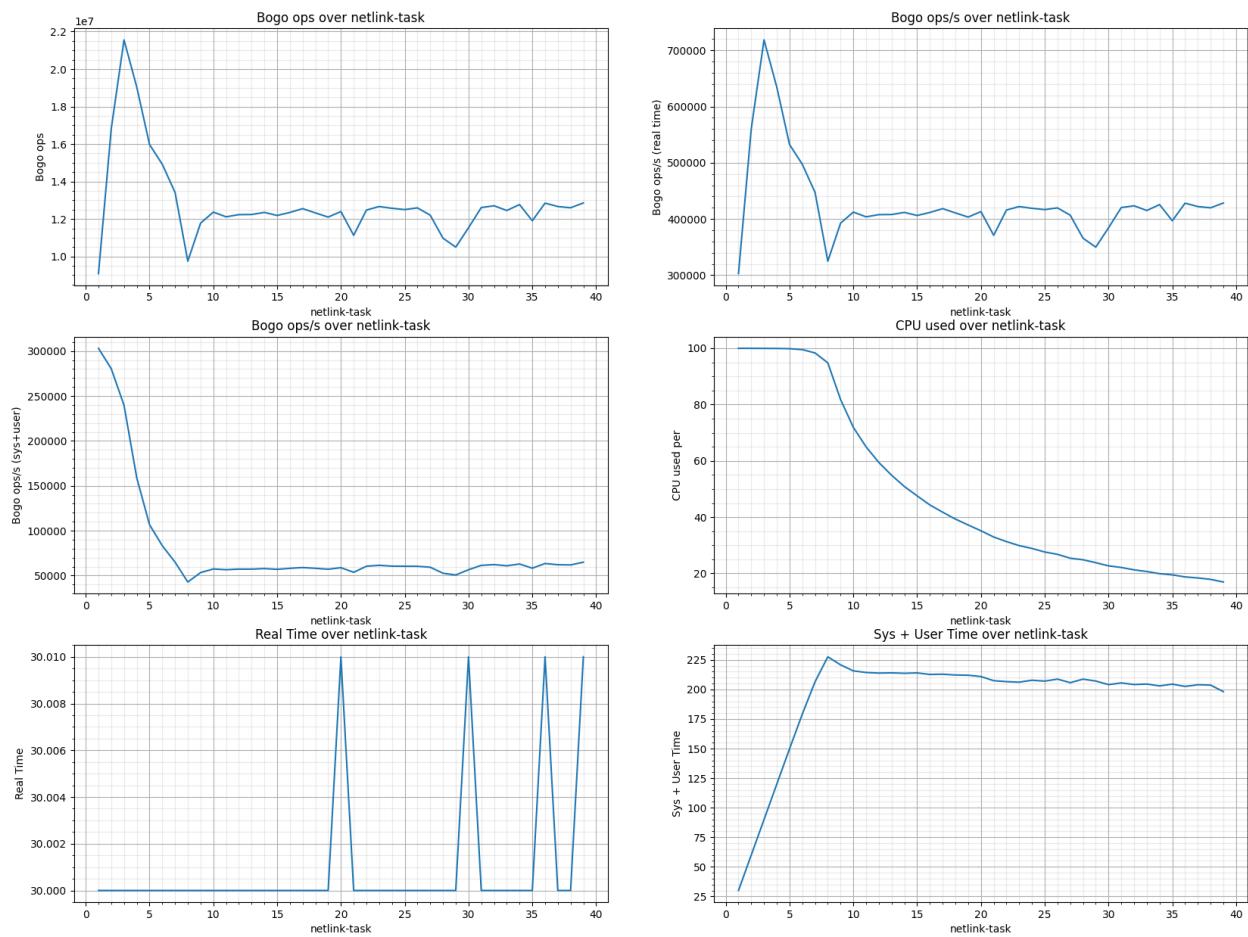
5.1 netlink-task

-netlink-task N: start *N* workers that collect task statistics via the netlink taskstats interface. This stressor can only be run on Linux and requires CAP_NET_ADMIN capability.

netlink: это специальное семейство адресов сокетов (*AF_NETLINK*), предназначенных для получения информации о ядре. Процедура использования *netlink* включает открытие сетевого

сокета с семейством адресов `AF_NETLINK` и последующее выполнение серии вызовов `send(2)` и `recv(2)` для передачи запросов и получения информации в двоичных структурах.

Найдем оптимальное количество воркеров:



Максимум bogo ops достигается при 3. С помощью команды `ss -a` мы можем посмотреть с какими сокетами работает `stress-ng`:

```
~/FlameGraph master 76 01:43:28
└─ ss -a | awk '/stress-ng/' genl:stress-ng/257279
nl  UNCONN 0      *
genl:stress-ng/257313
genl:stress-ng/257325
genl:stress-ng/257332
genl:stress-ng/257297
genl:stress-ng/257282
genl:stress-ng/257322
genl:stress-ng/257303
```

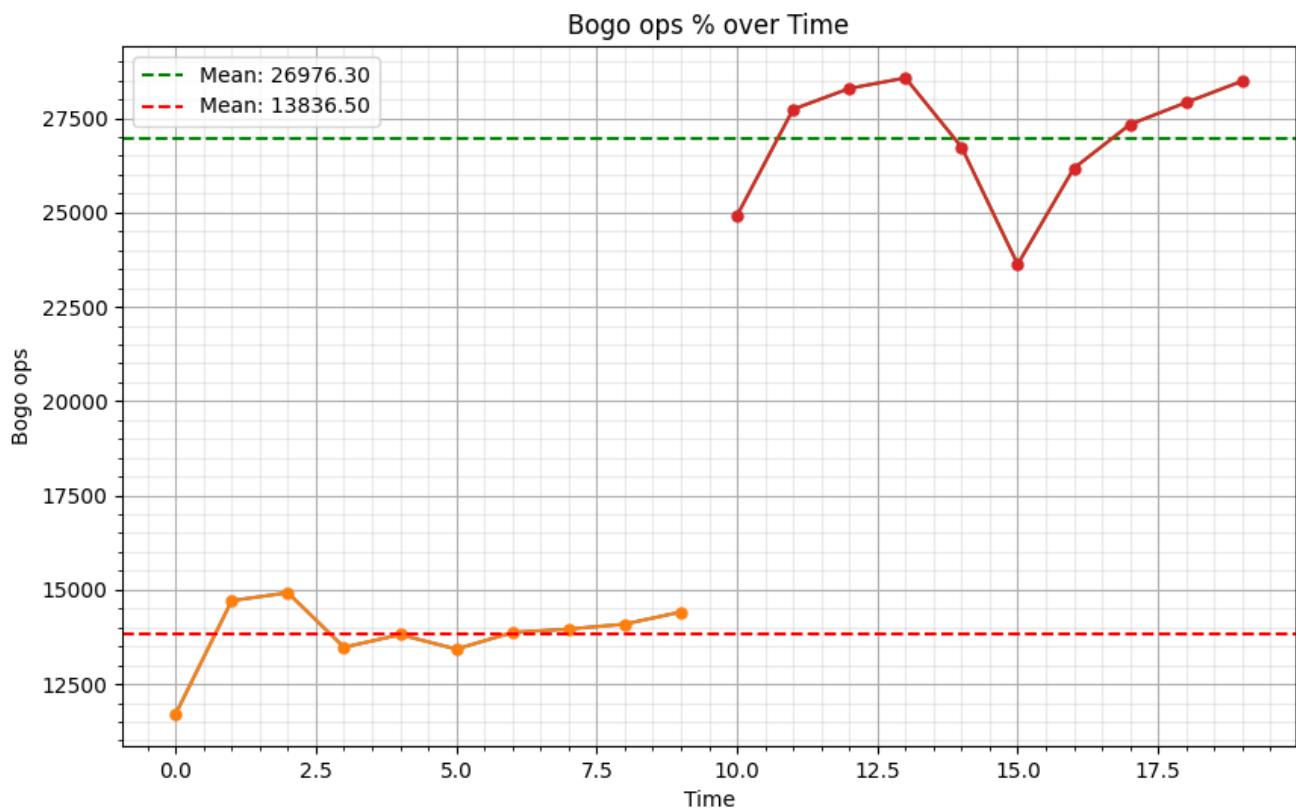
Также нагрузку, которую создает `netlink-task` на `netlink` можно заметить если запустить параллельно другой тест работающий с `netlink`.

-netlink-proc N: *start N workers that spawn child processes and monitor fork/exec/exit process*

events via the proc netlink connector. Each event received is counted as a bogo op. This stressor can only be run on Linux and requires CAP_NET_ADMIN capability.

Напишем скрипт запускающий несколько раз netlink-proc с netlink-task и без, сравним bogo ops.

```
#!/bin/zsh
2 for i in {1..10..1};do
    sudo stress-ng --netlink-proc 5 --netlink-task 100 --metrics --timeout 30 --stdout |
4     awk -v var="$i" '/[\[\]\d]+[ ]+netlink-proc/{printf "netlink;\"$5\";$6\";$7\";$8\";$9\";$10\";$11\"\n"}'
done
6 for i in {1..10..1};do
    sudo stress-ng --netlink-proc 5 --metrics --timeout 30 --stdout |
8     awk -v var="$i" '/[\[\]\d]+[ ]+netlink-proc/{printf "empty;\"$5\";$6\";$7\";$8\";$9\";$10\";$11\"\n"}'
done
```

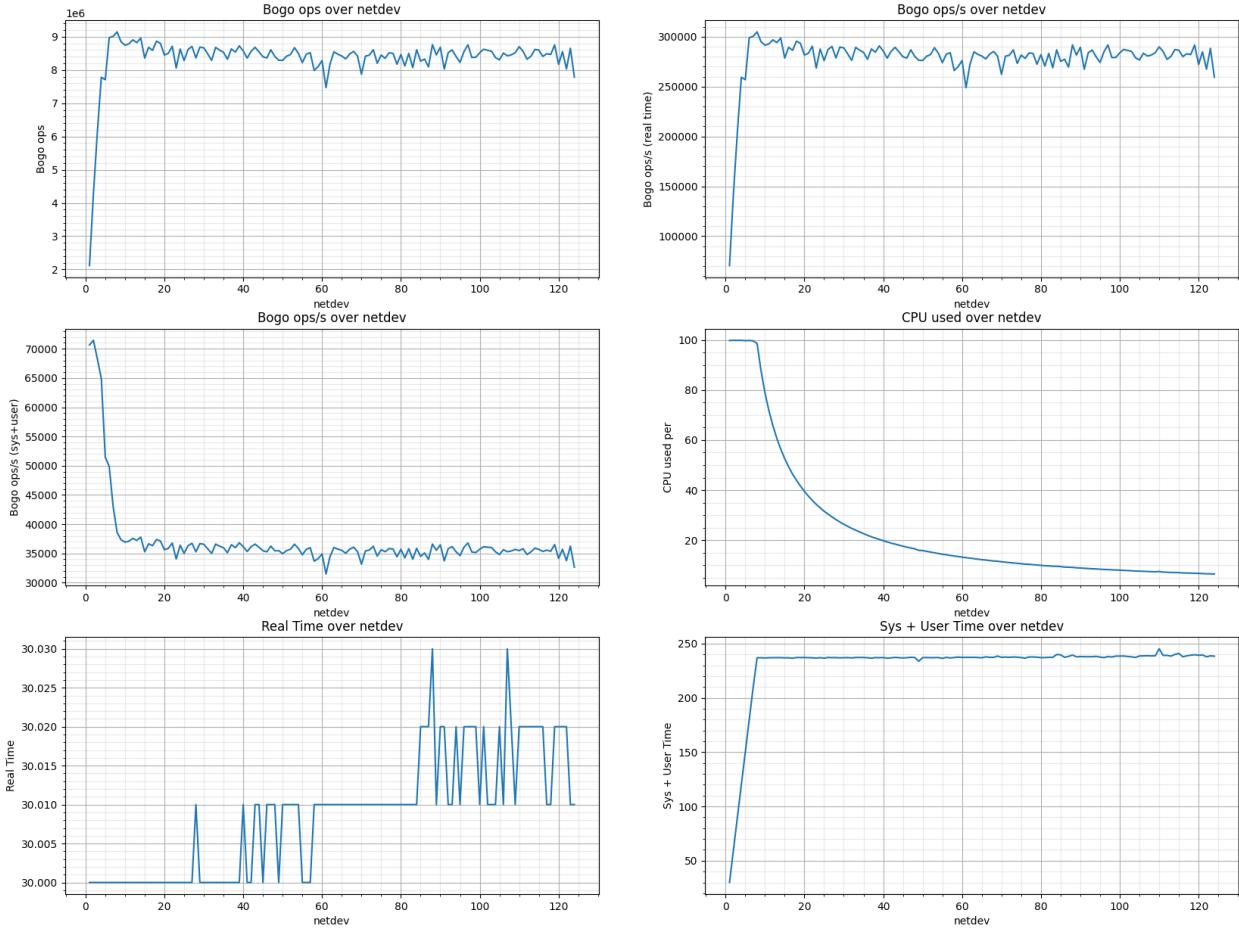


Видим, что из-за нагрузки производительность netlink падает.

5.2 netdev

-netdev N: start N workers that exercise various netdevice ioctl commands across all the available network devices. The ioctls exercised by this stressor are as follows: SIOCGIFCONF, SIOCGIFINDEX, SIOCGIFNAME, SIOCGIFFLAGS, SIOCGIFADDR, SIOCGIFNETMASK, SIOCGIFMETRIC, SIOCGIFHWADDR, SIOCGIFMAP and SIOCGIFTXQLEN. See netdevice(7) for more details of these ioctl commands.

Найдем оптимальное количество воркеров:



Максимум bogo ops достигается при 8. Командой `strace` мы можем посмотреть на команды, которые отправляет netdev сетевым интерфейсам:

```
~/FlameGraph master ?6
└─ sudo strace -e trace=ioctl -fp 250490
strace: Process 250490 attached
ioctl(4, SIOCGIFFLAGS, {ifr_name="lo", ifr_flags=IFF_UP|IFF_LOOPBACK|IFF_RUNNING}) = 0
ioctl(4, SIOCGIFPFLAGS, 0x55dff5b97148) = -1 EINVAL (Invalid argument)
ioctl(4, SIOCGIFADDR, {ifr_name="lo", ifr_addr={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("127.0.0.1")}}) = 0
ioctl(4, SIOCGIFNETMASK, {ifr_name="lo", ifr_netmask={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("255.0.0.0")}}) = 0
ioctl(4, SIOCGIFMETRIC, {ifr_name="lo", ifr_metric=0}) = 0
ioctl(4, SIOCGIFMTU, {ifr_name="lo", ifr_mtu=65536}) = 0
ioctl(4, SIOCGIFHWADDR, {ifr_name="lo", ifr_hwaddr={sa_family=ARPHRD_LOOPBACK, sa_data=00:00:00:00:00:00}}) = 0
ioctl(4, SIOCGIFMAP, {ifr_name="lo", ifr_map={mem_start=0, mem_end=0, base_addr=0, irq=0, dma=0, port=0}}) = 0
ioctl(4, SIOCGIFTXQLEN, {ifr_name="lo", ifr_qlen=1000}) = 0
ioctl(4, SIOCGIFDSTADDR, {ifr_name="lo", ifr_dstaddr={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("127.0.0.1")}}) = 0
ioctl(4, SIOCGIFBRDADDR, {ifr_name="lo", ifr_broadaddr={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("0.0.0.0")}}) = 0
ioctl(4, SIOCGIFMEM, 0x55dff5b97148) = -1 ENOTTY (Inappropriate ioctl for device)
ioctl(4, SIOCGIFCONF, {ifc_len=80 /* 2 * sizeof(struct ifreq) */, ifc_buf=NULL}) = 0
ioctl(4, SIOCGIFCONF, {ifc_len=80 /* 2 * sizeof(struct ifreq) */, ifc_buf=[{ifr_name="lo", ifr_addr={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("127.0.0.1")}}, {ifr_name="enp0s5", ifr_addr={sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("10.211.55.4")}}]})
```

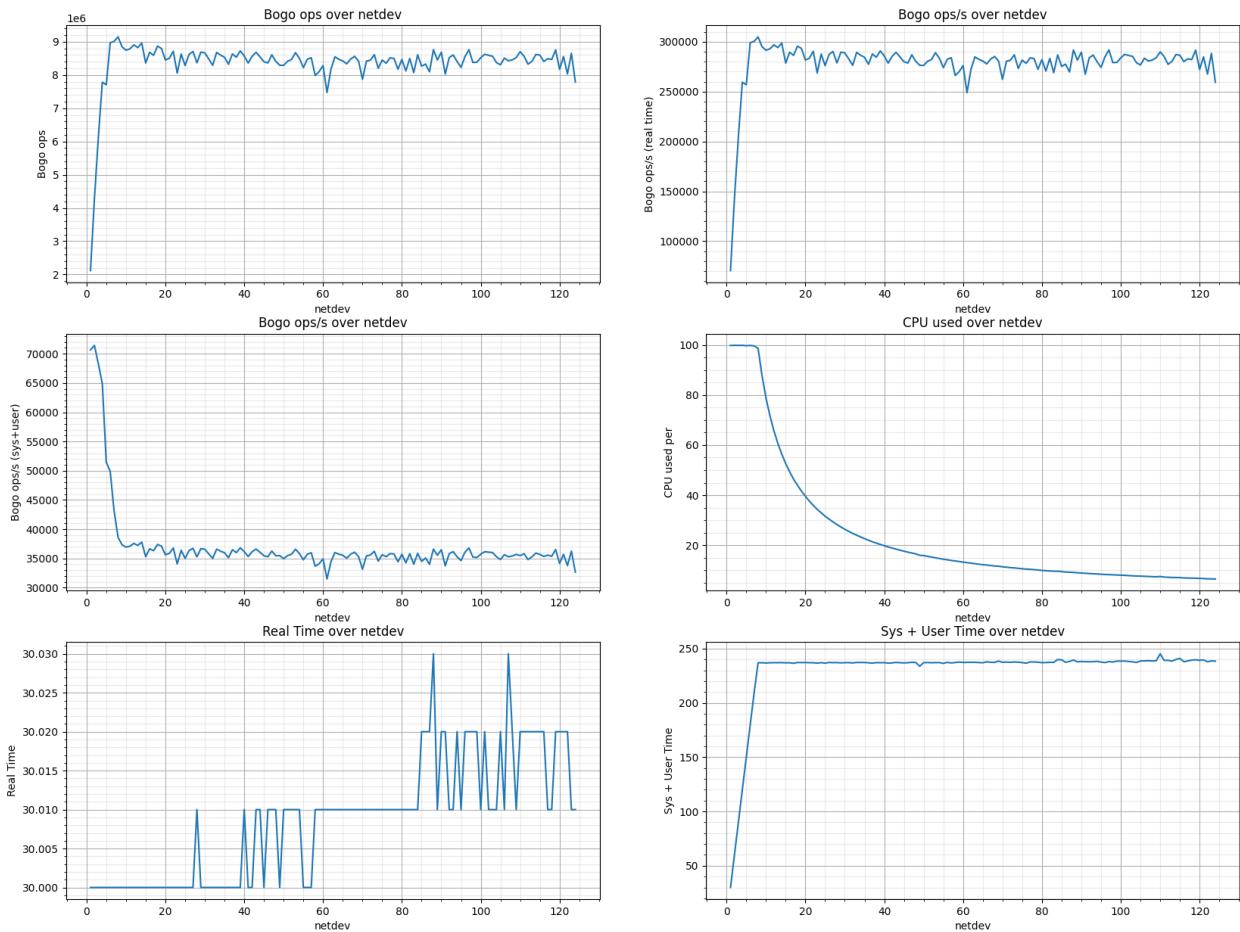
6 Pipe testing

Выданные параметры: [pipe-data-size, pipe-ops]

6.1 pipe

—**pipe N:** start N workers that perform large pipe writes and reads to exercise pipe I/O. This exercises memory write and reads as well as context switching. Each worker has two processes, a reader and a writer.

Найдем оптимальное количество воркеров:



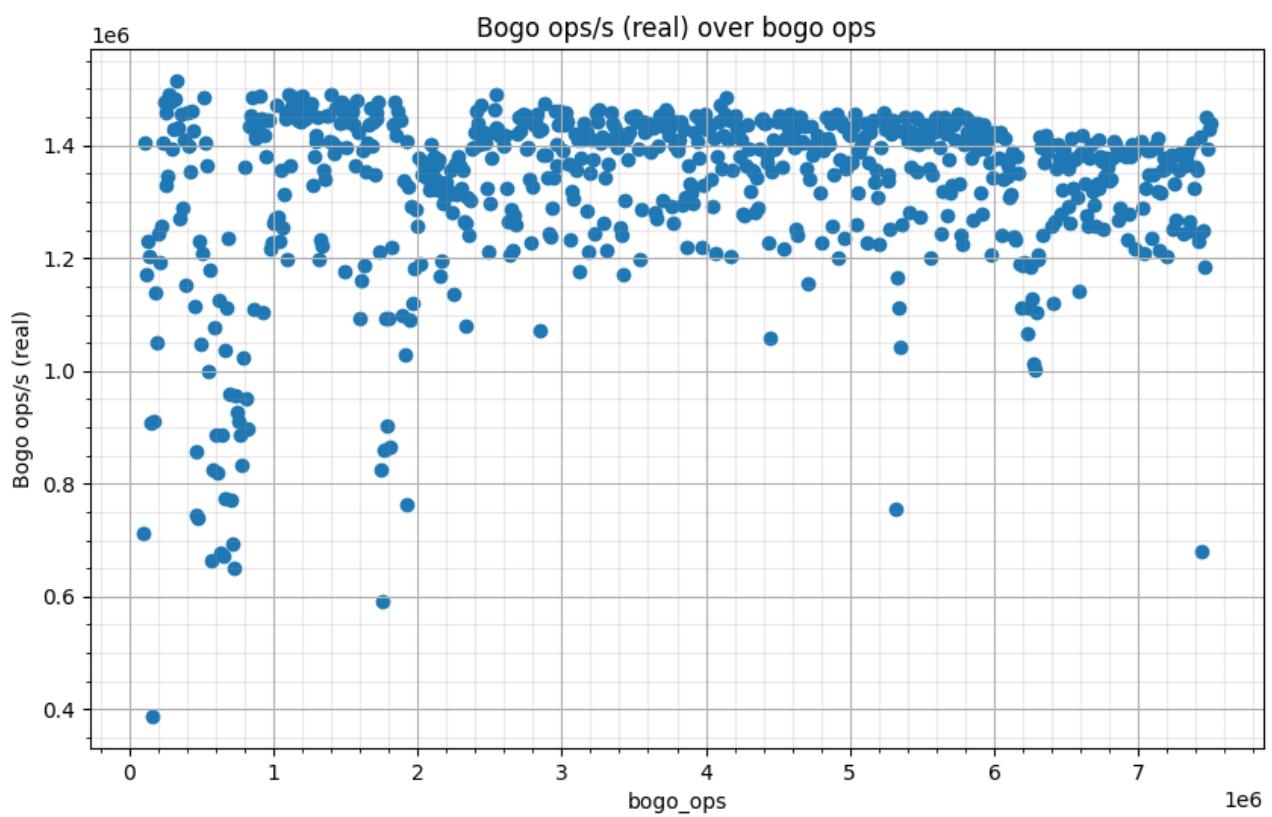
Максимум bogo ops достигается при 4.

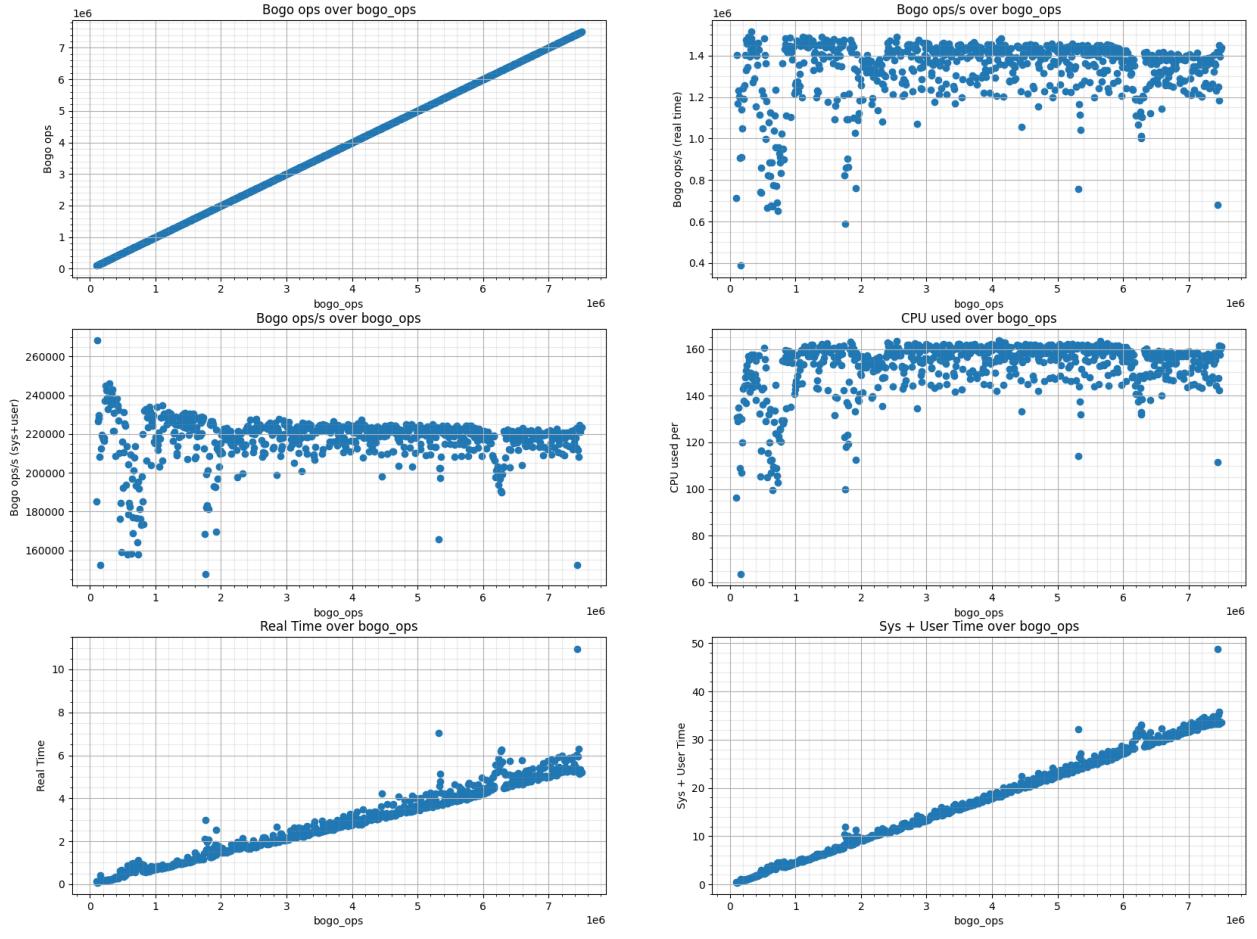
6.2 pipe-ops

—**pipe-ops N:** stop pipe stress workers after N bogo pipe write operations.

Попробуем оценить производительность по *bogo ops/s*

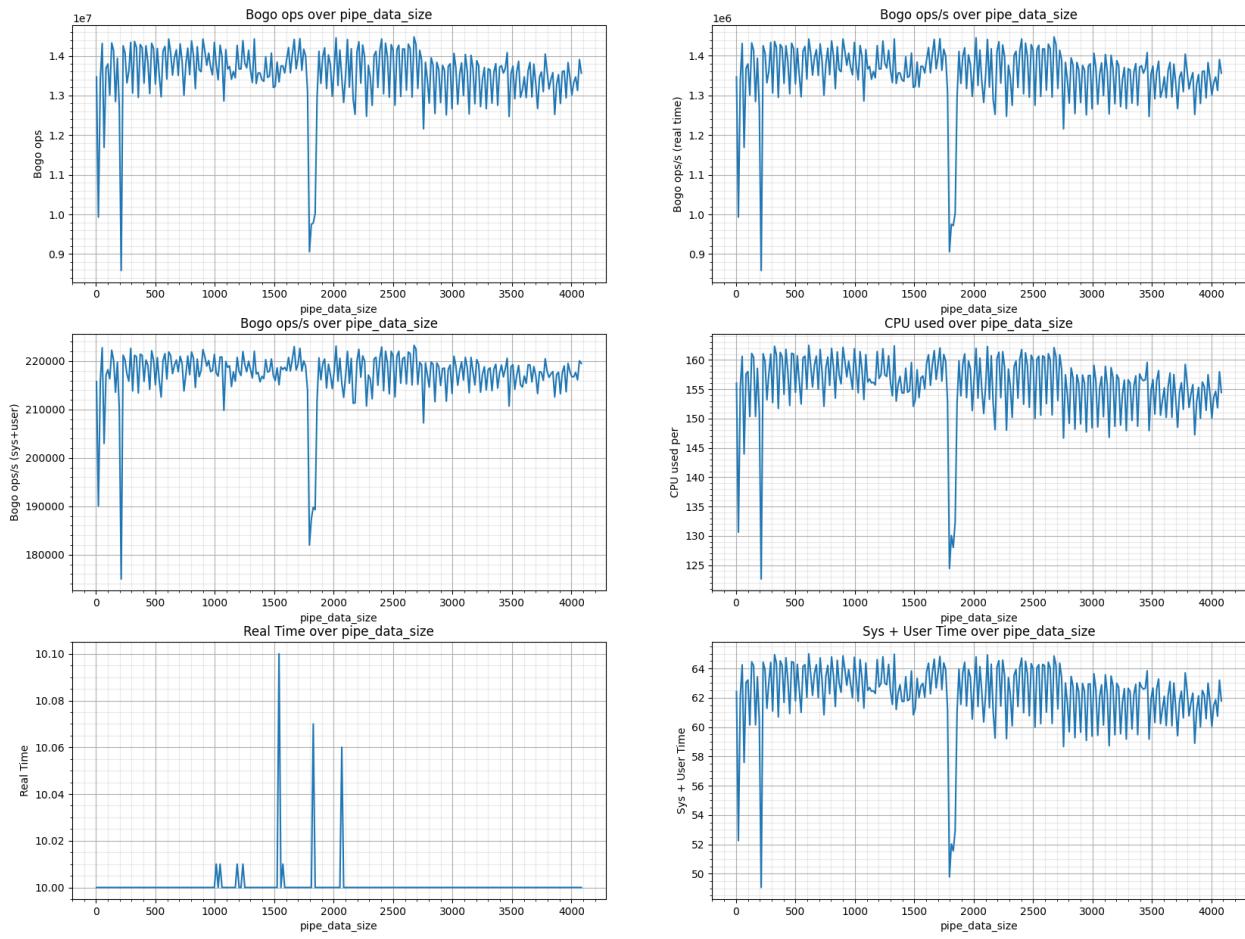
```
#!/bin/zsh
echo "pipe; bogo_ops; real_time; usr_time; sys_time; bogo_ops/s_real; bogo_ops/s_user+sys; CPU_used_per"
for i in {100000..100000000000..10000}; do
    sudo stress-ng --pipe 4 --pipe-ops $i --metrics --stdout |
        awk -v var="$i" '/[\[\]\d]+[ ]+pipe/{printf var;"$5";"$6";"$7";"$8";"$9";"$10";"$11"\n}' '
done
```





6.3 pipe-data-size

pipe-data-size N: specifies the size in bytes of each write to the pipe (range from 4 bytes to 4096 bytes). Setting a small data size will cause more writes to be buffered in the pipe, hence reducing the context switch rate between the pipe writer and pipe reader processes. Default size is the page size.



7 Sched testing

Выданные параметры: [yield,schedpolicy]

-y N, --yield N: start N workers that call `sched_yield(2)`. This stressor ensures that at least 2 child processes per CPU exercise `shield_yield(2)` no matter how many workers are specified, thus always ensuring rapid context switching.

```
/media/psf/Home/ITMO/operating-systems/Lab1/sched/scripts main !3 ?1
sudo stress-ng --yield 2 --metrics --timeout 3m
stress-ng: info: [240802] setting to a 180 second (3 mins, 0.00 secs) run per stressor
stress-ng: info: [240802] dispatching hogs: 2 yield
stress-ng: info: [240803] stress-ng-yield: limiting to 8 child yielders (instance 0)
stress-ng: info: [240804] stress-ng-yield: limiting to 8 child yielders (instance 1)
stress-ng: info: [240802] successful run completed in 180.05s (3 mins, 0.05 secs)
stress-ng: info: [240802] stressor      bogo ops real time  usr time  sys time   bogo ops/s      bogo ops/s CPU used per
stress-ng: info: [240802]                      (secs)    (secs)    (secs)    (real time) (usr+sys time) instance (%)
stress-ng: info: [240802] yield          408747842    180.04    510.39    866.28    2270296.83    296910.55      382.32
```

Запустим `pidstat` с параметром `-w`:

cswch/s: Total number of voluntary context switches the task made per second. A voluntary context switch occurs when a task blocks because it requires a resource that is unavailable.

nvcswch/s: Total number of non voluntary context switches the task made per second. A involuntary context switch takes place when a task executes for the duration of its time slice and then is forced to relinquish the processor.

	UID	PID	cswch/s	nvcswch/s	Command
00:07:21					
00:07:22	0	239293	10,00	0,00	stress-ng
00:07:22	0	239294	9,00	0,00	stress-ng
00:07:22	0	239295	0,00	61461,00	stress-ng
00:07:22	0	239296	0,00	49082,00	stress-ng
00:07:22	0	239297	0,00	63297,00	stress-ng
00:07:22	0	239298	0,00	60482,00	stress-ng
00:07:22	0	239299	0,00	58600,00	stress-ng
00:07:22	0	239300	0,00	57,00	stress-ng
00:07:22	0	239301	0,00	52600,00	stress-ng
00:07:22	0	239302	0,00	58579,00	stress-ng
00:07:22	0	239303	0,00	62563,00	stress-ng
00:07:22	0	239304	0,00	32637,00	stress-ng
00:07:22	0	239305	0,00	55949,00	stress-ng
00:07:22	0	239306	0,00	55856,00	stress-ng
00:07:22	0	239307	0,00	61420,00	stress-ng
00:07:22	0	239308	0,00	35061,00	stress-ng
00:07:22	0	239309	0,00	60805,00	stress-ng
00:07:22	0	239310	0,00	35199,00	stress-ng

Видим, что исполняются только два процесса, остальные же постоянно меняют контекст.

--schedpolicy N: start *N* workers that work set the worker to various available scheduling policies out of SCHED_OTHER, SCHED_BATCH, SCHED_IDLE, SCHE_FIFO, SCHED_RR and SCHED_DEADLINE. For the real time scheduling policies a random sched priority is selected between the minimum and maximum scheduling priority settings.

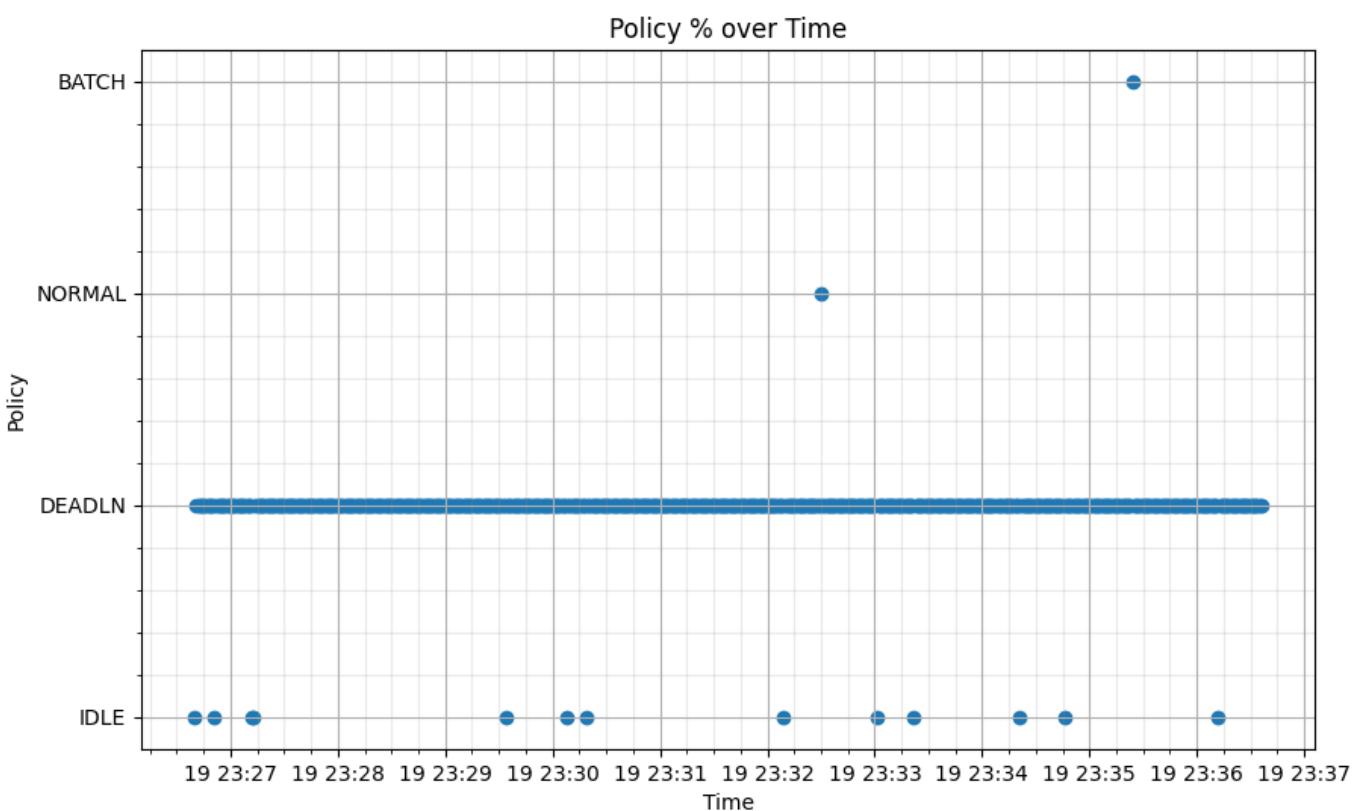
stress-ng: info: [233910] setting to a 600 second (10 mins, 0.00 secs) run per stressor						
stress-ng: info: [233910] dispatching hogs: 100 schedpolicy						
stress-ng: info: [233910] successful run completed in 600.25s (10 mins, 0.25 secs)						
stressor	bogo ops	real time	usr time	sys time	bogo ops/s	bogo ops/s CPU used per instance (%)
	(secs)	(secs)	(secs)	(real time)	(usr+sys time)	
stress-ng: info: [233910] schedpolicy	90153626	600.01	1502.70	3124.97	150254.28	19481.43 7.71

С помощью команды *top* мы видим, что политики планирования меняются:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
233979	root	20	0	46056	2140	1408	R	86,5	0,1	0:40.64	stress-ng
233966	root	-93	0	46056	2140	1408	R	62,0	0,1	0:46.84	stress-ng
233997	root	20	0	46056	2140	1408	R	53,5	0,1	0:38.72	stress-ng
233970	root	20	0	46056	2140	1408	R	48,8	0,1	0:37.52	stress-ng
233925	root	20	0	46056	2140	1408	R	40,9	0,1	0:35.81	stress-ng
233935	root	20	0	46056	2140	1408	R	23,8	0,1	0:34.58	stress-ng
234008	root	20	0	46056	2140	1408	R	17,5	0,1	0:41.25	stress-ng
233932	root	20	0	46056	2140	1408	R	14,5	0,1	0:44.13	stress-ng
234009	root	20	0	46056	2140	1408	R	13,5	0,1	0:40.53	stress-ng
233971	root	20	0	46056	2140	1408	R	13,2	0,1	0:36.59	stress-ng
233986	root	20	0	46056	2140	1408	R	13,2	0,1	0:31.93	stress-ng
233943	root	20	0	46056	2140	1408	R	12,2	0,1	0:40.87	stress-ng
233996	root	20	0	46056	2140	1408	R	10,6	0,1	0:32.77	stress-ng
234001	root	20	0	46056	2140	1408	R	10,2	0,1	0:38.60	stress-ng
233949	root	20	0	46056	2140	1408	R	8,9	0,1	0:33.70	stress-ng
233931	root	20	0	46056	2140	1408	R	8,6	0,1	0:38.24	stress-ng
233948	root	20	0	46056	2140	1408	R	7,6	0,1	0:34.65	stress-ng
233974	root	20	0	46056	2140	1408	R	7,6	0,1	0:31.42	stress-ng
233922	root	-42	0	46056	2140	1408	R	7,3	0,1	0:38.76	stress-ng
233995	root	20	0	46056	2140	1408	R	6,6	0,1	0:37.04	stress-ng
233926	root	20	0	46056	2140	1408	R	6,3	0,1	0:39.03	stress-ng
233930	root	20	0	46056	2140	1408	R	6,3	0,1	0:37.77	stress-ng
233936	root	20	0	46056	2140	1408	R	6,3	0,1	0:31.85	stress-ng
233946	root	20	0	46056	2140	1408	R	6,3	0,1	0:42.24	stress-ng
233960	root	20	0	46056	2140	1408	R	6,3	0,1	0:38.76	stress-ng
233956	root	20	0	46056	2140	1408	R	5,9	0,1	0:33.06	stress-ng
233964	root	20	0	46056	2140	1408	R	5,9	0,1	0:40.88	stress-ng
233983	root	20	0	46056	2140	1408	R	5,9	0,1	0:35.74	stress-ng
233921	root	20	0	46056	2140	1408	R	5,6	0,1	0:38.96	stress-ng

Выберем рандомный процесс и посмотрим как его политики планирования изменяются со временем:

```
#!/bin/zsh
2 sudo stress-ng --schedpolicy 100 --metrics --timeout 10m &
sleep 30
4 ps aux | awk '/stress-ng-schedpolicy \[run\]/ {print $2}' | head -n1 | xargs -I{} pidstat -R -p {} 1 > ./pidstat.txt
```



Видим, что он большую часть работал с политикой DEADLINE, немного с IDLE, и совсем чуть-чуть с NORMAL и BATCH.

8 Вывод:

В ходе выполнения лабораторной работы я познакомился с разными параметрами системы, научился проводить их мониторинг, настройку и нагружочное тестирование.