**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ»**

***Изображение выглядит как черный, темнота

Автоматически созданное описание***

**Факультет Программной Инженерии и Компьютерной Техники**

**Дисциплина:**

**«*Операционные системы*»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

**Выполнил:**

Студент гр. P32151 *Соловьев Артемий Александрович*

**Проверил:**

*Осипов Святослав Владимирович*

Санкт-Петербург

2023г.

**Задание**

Основная цель лабораторной работы — знакомство с системными инструментами анализа производительности и поведения программ. В данной лабораторной работе Вам будет предложено произвести нагрузочное тестирование Вашей операционной системы при помощи инструмента stress-ng.

В качестве тестируемых подсистем использовать: cpu, cache, io, memory, network, pipe, scheduler.

Для работы со счетчиками ядра использовать все утилиты, которые были рассмотренны на лекции (раздел 1.9, кроме kdb)

Ниже приведены списки параметров для различных подсистем (Вам будет выдано 2 значения для каждой подсистемы согласно варианту в журнале). Подбирая числовые значения для выданных параметров, и используя средства мониторинга, добиться **максимальной** производительности системы (BOGOPS, FLOPS, Read/Write Speed, Network Speed).

**Выполнение**

Параметры для **cpu**:

1. sieve
2. int128decimal64

Параметры для **cache**:

1. cache-ways
2. cache-fence

Параметры для **io**:

1. iomix
2. io-uring

Параметры для **memory**:

1. misaligned-method
2. memfd-fds

Параметры для **network**:

1. netlink-task
2. netdev

Параметры для **pipe**:

1. pipe-data-size
2. pipeherd-yield

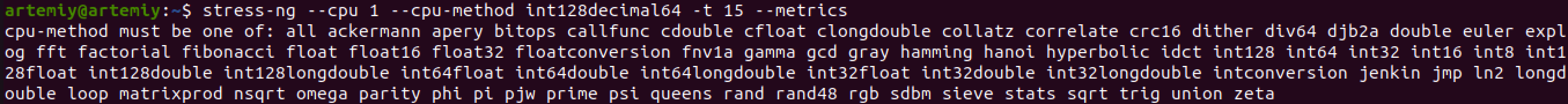
Параметры для **sched**:

1. yield
2. sched-deadline

Ссылка на расчеты:

[Ссылка](https://docs.google.com/spreadsheets/d/1xiHahX9sr_Pqtgv1-elGaitbli0hK1W2rkNLuli3eGk/edit?hl=ru#gid=0)

**Анализ производительности cpu:**

int128decimal64 ☹ 

В моей версии нет cpu-метода int128decimal64. Вместо него будет int128double.  
Для бенчмарка будут использоваться **sieve** и **int128double**.Изменяться будет только количество воркеров (параметр всего один). Мониторинг проведен с помощью sar и htop.

1. Sieve

stress-ng --cpu $I --cpu-method sieve -t 15 --metrics | sar 1 15 -u  
+htop +top

I = 1, 2, 3, 4, 5, 6

I = 1  
Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

1. int128double  
     
   stress-ng --cpu $I --cpu-method int128double -t 15 --metrics | sar 1 15 -u  
     
   I = 1, 2, 3, 4, 5, 6  
     
   I = 1  
   Изображение выглядит как текст, снимок экрана

   Автоматически созданное описание  
     
   Изображение выглядит как снимок экрана, линия, текст

   Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

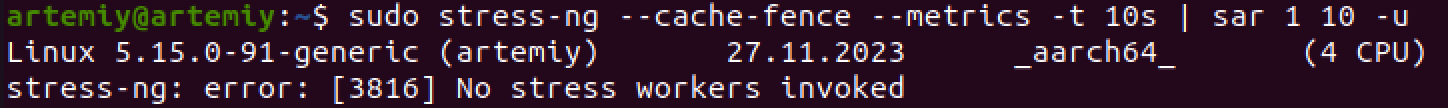
Графики:

Изображение выглядит как линия, диаграмма, График, число

Автоматически созданное описание

Оба метода ведут себя примерно одинаково: при увеличении количества воркеров до количества процессоров возрастает количество операций и нагрузка на процессор, но int128double намного сильнее нагружает процессор.

**Анализ производительности cache**

  
**cache-fence** не работает, вместо него возьмем stream-l3-size.  
Для бенчмарка будем использовать параметры **cache-ways** и **stream-l3-size**.

1. cache-ways  
     
   stress-ng --cache 0 --cache –cache-ways $i –t 10s –metrics  
     
   I=4,8,12,20  
   I = 4  
   Изображение выглядит как текст, снимок экрана

   Автоматически созданное описание

1. stream-l3-size  
   stress-ng --cache 0 --stream-l3-size $i –t 10s --metrics  
     
   I = 10k, 30k, 50k, 90k  
   Изображение выглядит как текст, снимок экрана

   Автоматически созданное описание

Графики:Изображение выглядит как линия, График, диаграмма, чек

Автоматически созданное описание

С увеличением числа путей l1-кеша увеличивается число операций. Увеличение размера l3-кеша не сильно влияет на уменьшение количества операций.   
Нагрузка на процессор в обоих случаях неизменна.

Замеры проводились с помощью sar, а в качестве мерки сравнение была взята bogoops. Perf на виртуальной машине не удалось запустить.

**Анализ производительности io**

Для бенчмарка будем использовать iomix и ioprio (так как io-uring не работает).

Для iomix число операций растет до достижения равенства между воркерами и процессорами, после количество записей уменьшается и держится на оптимальном для системы значении. Для ioprio растет число операций и скорость записи.

1. iomix

stress-ng --iomix %I –t 10s --metrics | iostat 10 1 -dhy sda  
  
I=8, 16, 32, 64, 128, 256  
  
I = 64  
Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

1. ioprio   
     
   stress-ng --ioprio $I –t 10s --metrics | iostat 10 1 -dhy sda  
     
   I = 8, 16, 32, 64, 128, 256  
     
   I = 8  
   Изображение выглядит как текст, снимок экрана, Шрифт

   Автоматически созданное описание

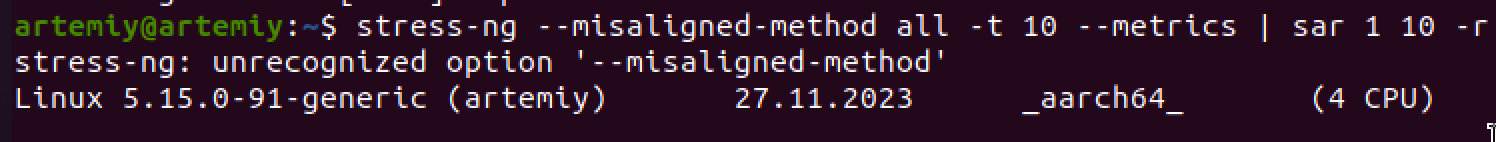
Графики:

Изображение выглядит как линия, диаграмма, График, число

Автоматически созданное описание

**Анализ производительности memory**

Невозможно запустить тестирование:



Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Были выбраны shm и lockbus.  
Для shm количество операций резко увеличивается при количестве воркеров равному 4 и 8, а после остается на том же уровне, кол-во используемой памяти растет с увеличением количества воркеров.

Для lockbus резко увеличивается количество операций, при количестве воркеров равному 4, после количество операций остается на одном уровне, а кол-во используемой памяти остаётся неизменным.

1. Shm  
     
   stress-ng --shm $I --metrics –t 10 | sar 1 10 –r  
     
   I = 2, 4, 8, 16, 32, 64  
     
   I = 2  
   Изображение выглядит как текст, снимок экрана

   Автоматически созданное описание
2. Lockbus  
     
   stress-ng --lockbus $I –t 10s --metrics | sar 1 10 -r  
   I = 2,4 ,8, 16, 32, 64  
   I = 2  
   Изображение выглядит как текст, снимок экрана, Шрифт

   Автоматически созданное описание

Графики:

Изображение выглядит как линия, График, диаграмма, Параллельный

Автоматически созданное описание

**Анализ производительности network**Для бенчмарка используем netdev и netlink-task.  
Для netlink-task резко увеличивается количество операций по мере увеличения числа воркеров до числа процессоров, после количество операций остается почти неизменным, а количество сокетов растет по мере увеличения числа воркеров.  
Для netdev анологично, но в netdev количество операций сильно меньше.

1. netlink-task  
   stress-ng --netlink-task $i --metrics –t 10s | sar 1 10 –n SOCK  
     
   Изображение выглядит как текст, снимок экрана

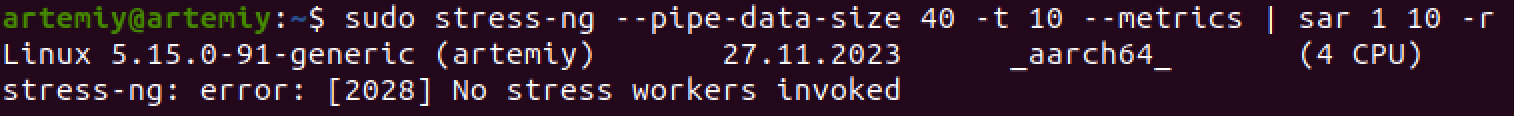
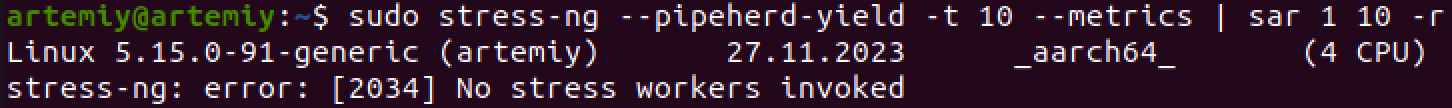
   Автоматически созданное описание
2. netdev  
     
   sudo stress-ng --netdev $I --metrics –t 10s | sar 1 10 -n SOCK  
     
   Изображение выглядит как текст, снимок экрана

   Автоматически созданное описание

Графики:

Изображение выглядит как линия, диаграмма, График, текст

Автоматически созданное описание

**Анализ производительности pipe******Не распознает pipe-data-size и pipeherd-yield, поэтому будем использовать pipe и pipeherd.  
Для pipe растет число операций по мере увеличения числа воркеров до числа процессоров, растет число операций, а также нагрузка на процессор. После достижения количество воркеров числа процессоров, значения почти не меняются.   
Для pipeherd по мере увеличения количества воркеров уменьшается число операций, но нагрузка на систему резковозрастает до достижения числа воркеров числа процессоров, после нагрузка остается примерно на одном уровне.

1. pipe   
   stress-ng –pipe $i --metrics –t 10s | sar 1 10 –u  
     
   I = 1, 3, 6, 16, 32, 64, 128  
   I = 1  
   Изображение выглядит как текст, снимок экрана

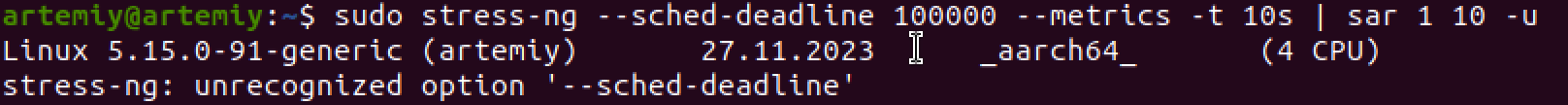
   Автоматически созданное описание
2. pipeherd   
     
   stress-ng –pipeherd $i --metrics –t 10s | sar 1 10 -u  
     
   I = 1, 3, 6, 16, 32, 64, 128  
   I = 1  
   Изображение выглядит как текст, снимок экрана

   Автоматически созданное описание

Графики:

**Изображение выглядит как текст, диаграмма, линия, График

Автоматически созданное описание**

**Анализ производительности sched**Из-за того, что –sched-deadline не работает, возьмём schedpolicy и yield.  
Для бенчмарка scheduler используем yield и schedpolicy. В yield количество операций и context-sw уменьшается. В schedpolicy кол-во операций увеличивается до определенного порога, затем уменьшается, то же самое происходит с context-sw.

1. yield  
     
   stress-ng --yield $i --metrics –t 10s | sar 1 10 –u  
   I = 1,3 ,6, 16, 32, 64,128  
     
   I = 1  
   Изображение выглядит как текст, снимок экрана, Шрифт

   Автоматически созданное описание
2. schedpolicy   
     
   stress-ng --schedpolicy $i --metrics –t 10s | sar 1 10 -u  
   I = 1,3 ,6, 16, 32, 64,128  
     
   I = 1  
   Изображение выглядит как текст, снимок экрана

   Автоматически созданное описание

Графики:  
Изображение выглядит как линия, График, диаграмма, число

Автоматически созданное описание

**Вывод:**  
В ходе выполнения лабораторной работы, я на практике познакомился с методами мониторинга различных частей системы, а также с методами их нагрузки.