Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:

“Операционные системы”

Лабораторная работа №3

“Оценка производительности. Параметры ядра.”

Выполнил:
ст. группы N3246 Цыдыпов А.О.

Проверил:
Ханов А.Р.

Санкт-Петербург

2022 г.

Задание:

Базовый вариант:

Найти и скомпилировать программу linpack для оценки производительности компьютера (Flops) и протестировать ее при различных режимах работы ОС:

1. С различными приоритетами задачи в планировщике
2. С наличием и отсутствием привязки к процессору
3. Провести несколько тестов, сравнить результаты по 3 сигма или другим статистическим критериям

Усиленный вариант:

То же самое, плюс изменить параметры на уровне ядра (выбрать одно):

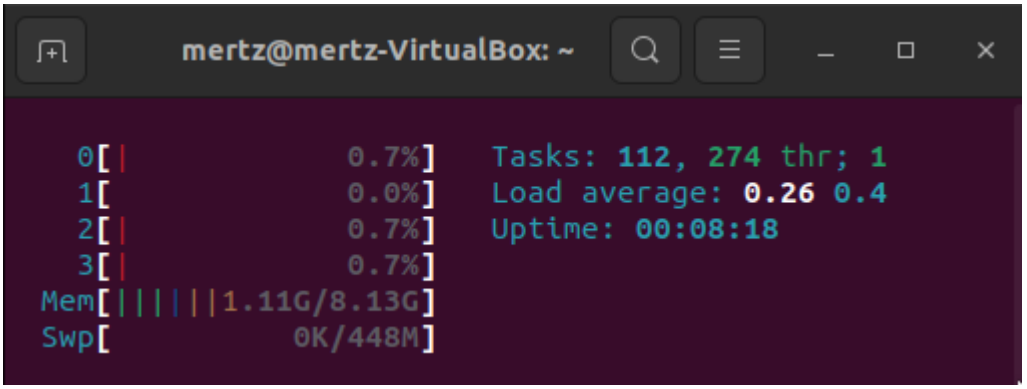
1. Запретить выполнение всех потоков кроме того, который тестируется (путем запрета
прерываний) (cli sti)
2. Найти другие планировщики процессов для Linux и сравнить результаты работы вычислительной задачи на них
3. Повлиять на настройки имеющегося планировщика
4. Вмешаться в работу планировщика на уровне ядра

Лабораторная работа выполнялась используя бенчмарк linpack.

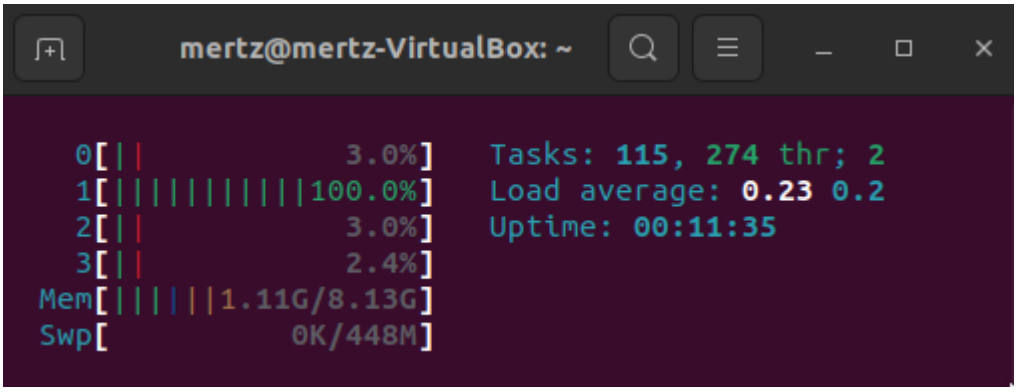
Github репозиторий

Ход работы:

Работа была начата на виртуальной машине.



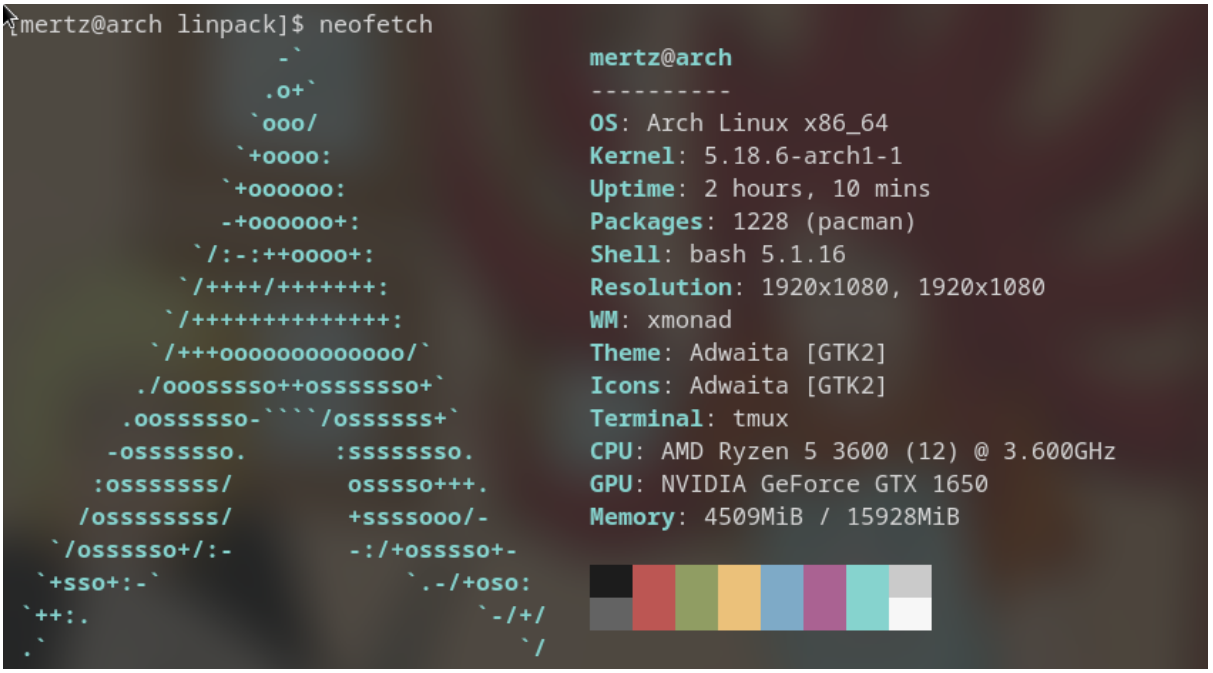
Скриншот загруженности процессора без выполнения каких либо юзер приложений.

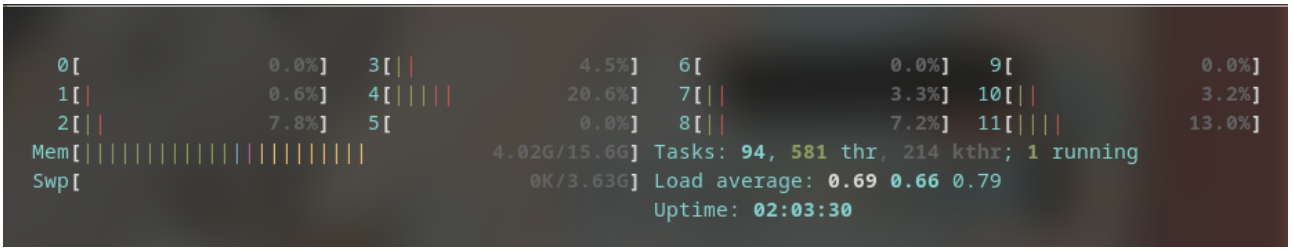


Скриншот загруженности процессора при запуске бенчмарка.

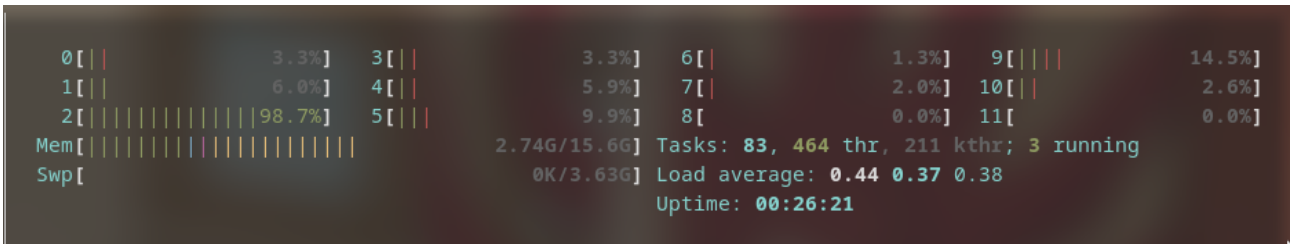
Как мы видим, запуская бенчмарк, он прикрепляется к одному ядру и там и выполняется. Во-первых потому что один прогон бенчмарка проводится довольно быстро, а во вторых потому что системе хватает оставшихся 3-х ядер. Исходя из этого, я считаю нецелесообразно прикреплять бенчмарк к одному из ядер (потоков), так как никакой разницы мы не получим. Единственное, что может помочь, так это сбросить все процессы с одного потока, и taskset'ом прикрепить бенчмарк к этому же потоку (но на виртуалке в этом крайне мало смысла, поэтому давайте сделаем это на настоящей машине).

Конфигурация:





Скриншот загруженности процессора без бенчмарка.



Скриншот загруженности процессора при запуске бенчмарка.

```
[mertz@arch linpack]$ ./linpack
Memory required:  315K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 200 X 200.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.78  71.30%   2.59%  26.11%  9717238.512
    8192   1.57  71.40%   2.58%  26.02%  9655598.973
   16384   3.18  71.46%   2.60%  25.95%  9567577.259
   32768   6.37  71.52%   2.61%  25.87%  9526415.548
   65536  12.45  71.32%   2.58%  26.09%  9784073.599
```

```
[mertz@arch linpack]$ sudo nice -n -10 ./linpack
Memory required:  315K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 200 X 200.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.77  71.23%   2.59%  26.18%  9906596.151
    8192   1.54  71.32%   2.60%  26.07%  9851252.136
   16384   3.08  71.30%   2.58%  26.11%  9887494.038
   32768   6.16  71.26%   2.58%  26.16%  9893719.807
   65536  12.42  71.34%   2.59%  26.07%  9804011.148
```

```
[mertz@arch linpack]$ sudo nice -n -20 ./linpack
Memory required:  315K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 200 X 200.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.79  71.52%   2.62%  25.87%  9653138.389
    8192   1.56  71.40%   2.60%  26.00%  9736201.775
   16384   3.09  71.26%   2.61%  26.13%  9856093.784
   32768   6.18  71.27%   2.60%  26.13%  9853206.346
   65536  12.38  71.28%   2.60%  26.12%  9839916.417
```

```
[mertz@arch linpack]$ sudo taskset -c 0 nice -n -10 ./linpack
Memory required:  315K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 200 X 200.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.77  71.22%   2.60%  26.19%  9872119.770
    8192   1.55  71.31%   2.60%  26.09%  9805513.986
   16384   3.11  71.28%   2.61%  26.11%  9796331.119
   32768   6.20  71.28%   2.58%  26.13%  9819489.879
   65536  12.39  71.28%   2.57%  26.14%  9836935.235
```

```
[mertz@arch linpack]$ sudo taskset -c 0 nice -n -20 ./linpack
Memory required:  315K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 200 X 200.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.78  71.42%   2.62%  25.96%  9686583.390
    8192   1.56  71.34%   2.62%  26.04%  9730660.728
   16384   3.13  71.44%   2.60%  25.96%  9713324.731
   32768   6.19  71.29%   2.59%  26.12%  9832949.168
   65536  12.49  71.44%   2.59%  25.97%  9737012.830
```

```
[mertz@arch linpack]$ sudo taskset -c 0 nice -n -20 ./linpack && sudo taskset -c 3 ./cle
an.sh
Memory required:  315K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 200 X 200.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.77  71.26%   2.60%  26.14%  9849303.012
    8192   1.55  71.33%   2.59%  26.08%  9789958.915
   16384   3.09  71.26%   2.58%  26.16%  9856866.644
   32768   6.20  71.33%   2.58%  26.09%  9819976.286
   65536  12.40  71.30%   2.58%  26.12%  9825412.530
```

Скрипт приведенный ниже переносит все процессы с первого ядра (1 и 2 потоки) на другие ядра, кроме самого бенчмарка.

./clean.sh

```
#!/bin/bash
while true
  PIDDD=$(ps -ef | awk '$NF~"linpack" {print $2}')
do
  for i in $(ps aux | tail -n +2 | awk '{print $2}')
  do
    if [ "$i" != "$PIDDD" ]; then
      sudo taskset -p 0xfc $i > /dev/null 2>&1
    fi
  done
done
```

Результаты работы:

Reps	default	nice -10	nice -20	taskset + nice -10	taskset + nice -20	очистка ядра
4096	9717238,512	9906596,151	9653138,389	9872119,77	9686583,39	9849303,012
8192	9655598,973	9851252,136	9736201,775	9805513,986	9730660,728	9789958,915
16384	9567577,259	9887494,038	9856093,784	9796331,119	9713324,731	9856866,644
32768	9526415,548	9893719,807	9853206,346	9819489,879	9832949,168	9819976,286
65536	9784073,599	9804011,148	9839916,417	9836935,235	9737012,83	9825412,53
СРЗНАЧ	9650180,778	9868614,656	9787711,342	9826077,998	9740106,169	9828303,477
СКО	21578460,22	22066893,21	21885987,91	21971778,36	21779539,5	21976754,68

Усложненный вариант:
Доступные параметры ядра (относящиеся к планировщику):

```
[mertz@arch ~]$ sudo sysctl -a | grep sched
kernel.sched_autogroup_enabled = 1
kernel.sched_cfs_bandwidth_slice_us = 5000
kernel.sched_child_runs_first = 0
kernel.sched_deadline_period_max_us = 4194304
kernel.sched_deadline_period_min_us = 100
kernel.sched_energy_aware = 1
kernel.sched_rr_timeslice_ms = 90
kernel.sched_rt_period_us = 1000000
kernel.sched_rt_runtime_us = 950000
kernel.sched_schedstats = 0
kernel.sched_util_clamp_max = 1024
kernel.sched_util_clamp_min = 1024
kernel.sched_util_clamp_min_rt_default = 1024
```

Изменим параметр отвечающий за планирование с учетом энергопотребления. Планирование с учетом энергопотребления (или EAS) дает планировщику возможность прогнозировать влияние его решений на энергию, потребляемую ЦП. EAS использует модель энергопотребления (EM) ЦП для выбора энергоэффективного ЦП для каждой задачи с минимальным влиянием на пропускную способность.

```
[mertz@arch linpack]$ su -c "echo 0 /proc/sys/kernel/sched_energy_aware"
Password:
0 /proc/sys/kernel/sched_energy_aware
```

```
[mertz@arch linpack]$ ./linpack
Memory required:  315K.


LINPACK benchmark, Double precision.
Machine precision:  15 digits.
Array size 200 X 200.
Average rolled and unrolled performance:

    Reps Time(s) DGEFA   DGESL  OVERHEAD    KFLOPS
----------------------------------------------------
    4096   0.77  71.27%   2.58%  26.15%  9864415.954
    8192   1.55  71.23%   2.60%  26.17%  9859980.900
   16384   3.11  71.33%   2.61%  26.06%  9782395.506
   32768   6.23  71.38%   2.60%  26.02%  9770773.145
   65536  12.31  71.21%   2.59%  26.20%  9909542.246
```

Как можем заметить, результат улучшился ~3%, что в целом неплохо.

Вывод: в этой работе я научился повышать производительность отдельных приложений на системной уровне, перенося сторонние процессы на другие ядра с помощью утилиты taskset. Научился повышать приоритет с помощью утилиты nice. Впервые поменял параметры ядра.