ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

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

Факультет безопасности информационных технологий

Кафедра проектирования и безопасности компьютерных систем

Дисциплина:

«Операционные системы»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Выполнили:

Студент группы N3248

Назаров Максим Вячеславович

Проверил:

Савков Сергей Витальевич

Санкт-Петербург 2022г.

Простой вариант

Найти и скомпилировать программу linpack(https://github.com/ereyes01/linpack) для оценки производительности компьютера (Flops) и протестировать ее при различных режимах работы ОС:

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

усложненный вариант

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

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

1. Для начала скачаем linpack:
`git clone https://github.com/ereyes01/linpack`
После этого соберем программу:
`make all`
Для различных приоритетов(nice у процесса)
запустим linpack. Для того чтобы можно было
оценивать результаты проведем ряд измерений:
`nice -n -20 ./linpack` Аналогично запишем
вывод linpack для -20,-10, 0,10,19. Вывод:

```
root@kali:~/linpack# 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
----------------------------------------------------
    2048   0.56  65.58%   2.12%  32.30%  7376579.960
    4096   1.11  65.11%   2.14%  32.75%  7525617.528
    8192   2.22  65.15%   2.13%  32.72%  7527289.207
   16384   4.45  65.15%   2.13%  32.72%  7517830.352
   32768   8.91  65.17%   2.12%  32.70%  7508502.979
   65536  17.77  65.14%   2.12%  32.74%  7529842.216

root@kali:~/linpack# 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
----------------------------------------------------
    2048   0.55  64.55%   2.21%  33.24%  7683151.584
    4096   1.08  63.96%   2.22%  33.83%  7884589.059
    8192   2.16  64.01%   2.20%  33.78%  7876121.642
   16384   4.31  63.90%   2.19%  33.90%  7891601.958
   32768   8.60  63.99%   2.20%  33.81%  7909218.097
   65536  17.22  63.98%   2.20%  33.83%  7896513.010
```

```
root@kali:~/linpack# nice -n 0 ./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
----------------------------------------------------
    2048   0.54  63.97%   2.19%  33.84%  7923337.324
    4096   1.07  63.99%   2.19%  33.83%  7919500.000
    8192   2.15  64.01%   2.20%  33.80%  7909072.141
   16384   4.31  63.95%   2.20%  33.85%  7894179.619
   32768   8.62  63.95%   2.19%  33.86%  7896458.279
   65536  17.21  64.00%   2.19%  33.80%  7900414.077

root@kali:~/linpack# 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
----------------------------------------------------
    2048   0.54  63.98%   2.22%  33.81%  7891923.035
    4096   1.09  64.11%   2.22%  33.67%  7806148.882
    8192   2.15  63.99%   2.20%  33.81%  7912120.268
   16384   4.31  64.03%   2.20%  33.77%  7882277.205
   32768   8.60  64.01%   2.19%  33.80%  7901242.892
   65536  17.24  64.03%   2.20%  33.77%  7884030.305
```

```
root@kali:~/linpack# nice -n 19 ./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
----------------------------------------------------
    2048   0.54  63.99%   2.20%  33.81%  7918296.026
    4096   1.08  64.08%   2.22%  33.70%  7864658.409
    8192   2.16  64.08%   2.22%  33.70%  7865901.122
   16384   4.31  64.02%   2.20%  33.78%  7889927.795
   32768   8.60  63.93%   2.20%  33.87%  7911359.407
   65536  17.23  64.04%   2.20%  33.75%  7885081.573
```

2. Запишем вывод для процесса, запущенному с привязкой к процессору, для nice = 19. И сравним значения с привязкой и без.

```
nice -n 19 taskset -c 0 ./linpack
```

```
root@kali:~/linpack# nice -n 19 taskset -c 0 ./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
----------------------------------------------------
    2048    0.54  64.09%   2.20%  33.72%  7880336.516
    4096    1.08  63.92%   2.21%  33.87%  7907221.050
    8192    2.15  63.93%   2.21%  33.86%  7917783.333
   16384    4.31  64.01%   2.20%  33.79%  7882163.995
   32768    8.61  63.91%   2.20%  33.89%  7905852.745
   65536   17.21  63.97%   2.20%  33.83%  7901592.502
```

Процесс привязан к одному из ядер. Как мы видим, в результате, по сравнению с процессом, запущенным без привязки, вычислительная мощность стала чуть больше(скорее всего связано это с многопоточностью).

3. Статистика.

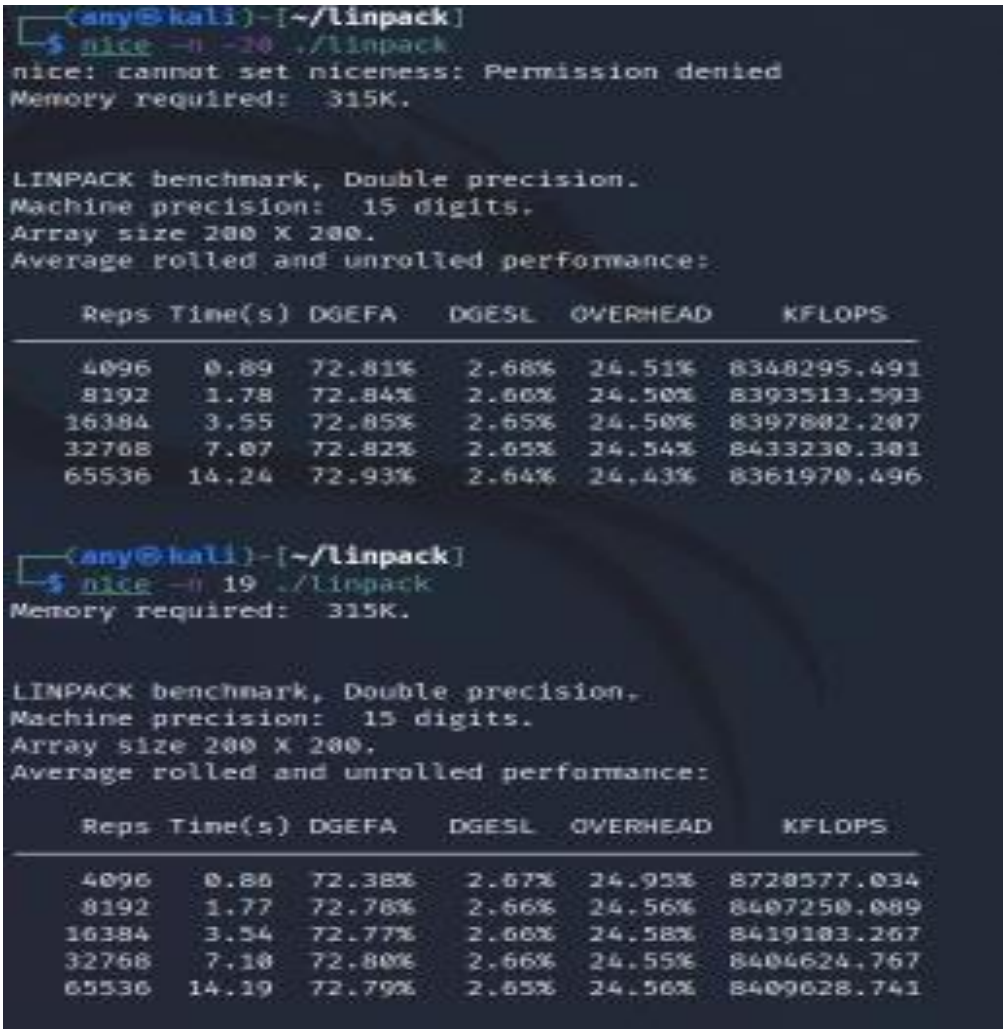
Вычислим матожидание(среднее по выборке) M и сигму - СКО sigm. Тогда результат измерений с высокой долей вероятности не выйдет за промежуток M+-3*sigm.

	A	B	C	D
1	nice	M	Sigm	y or no
2	-20	7500305	61346	+
3	-10	7907973	59313	+
4	0	7919342	57234	+
5	10	7899958	57134	+
6	19	7900009	58424	+
7	cpu19	7904531	53456	+

Разброс достаточно большой, однако средние значения мало отражают гипотезу того что то, при увеличении параметра nice вычислительная мощность растёт.

Interesting: при параллельном чтении 100 /dev/random и запуске игры на нормальном железе(второй оси на компьютере цифры становятся адекватнее и действительно отражается что при nice = -20 производительность выше, но в рамках "голой" виртуальной

машины такого не было замечено в связи с тем что это не высоконагруженная система и каких-то весомых помех для linpack'a не удаётся создать.



Результат работы Linpack'a для физической операционной системы с параллельным выполнением 10-ти выводов из /dev/random, на пользователе имеется CAP_SYS_NICE(kali linux last version)

Усложненный вариант:

Повлиять на настройки имеющегося планировщика.

Изменим несколько настроек и посмотрим как они влияют на вычислительную мощность системы. Для этого скомпилируем код linpack без изменений, введенных в первой части лабораторной работы.

Попробуем изменить параметр:

kernel.sched_min_granularity_ns

Задачи, привязанные к процессору, гарантированно выполняются в течение этого заданного(минимального) времени, прежде чем они будут вытеснены другими. Задача

считается связанной с процессором, когда время, необходимое для ее выполнения, зависит только от скорости процессора(т.е сетевые операции к ним не относятся!). Как правило, увеличение этого значения увеличивает пропускную способность системы. Этот параметр принимает значения в наносекундах(ns говорит об этом).
Установим в него первое значение времени и запустим linpack:

```
root@kali:~/linpack# sysctl -A | grep "sched" | grep -v "domain"
kernel.sched_autogroup_enabled = 0
kernel.sched_cfs_bandwidth_slice_us = 5000
kernel.sched_child_runs_first = 0
kernel.sched_latency_ns = 12000000
kernel.sched_migration_cost_ns = 500000
kernel.sched_min_granularity_ns = 1500000
kernel.sched_nr_migrate = 32
kernel.sched_rr_timeslice_ms = 100
kernel.sched_rt_period_us = 1000000
kernel.sched_rt_runtime_us = 950000
kernel.sched_schedstats = 0
kernel.sched_tunable_scaling = 1
kernel.sched_wakeup_granularity_ns = 2000000
root@kali:~/linpack# sysctl -w kernel.sched_min_granularity_ns=1000000
kernel.sched_min_granularity_ns = 1000000
root@kali:~/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
----------------------------------------------------
    2048   0.54  63.93%   2.17%  33.90%  7852154.192
    4096   1.08  64.27%   2.17%  33.56%  7813543.799
    8192   2.15  63.93%   2.19%  33.87%  7908727.428
   16384   4.30  63.98%   2.20%  33.81%  7900316.296
   32768   8.62  64.03%   2.19%  33.77%  7881136.969
   65536  17.21  64.02%   2.19%  33.78%  7895918.623
```

После этого уменьшим допустимое время в 10 раз(т.е 100000 наносекунд), то есть по сути дадим системе меньше времени на выполнение процессов. Тогда вычислительная мощность измеренная linpack'ом должна увеличиться. Проверим это:

```
root@kali:~/linpack# sysctl -w kernel.sched_min_granularity_ns=100000
kernel.sched_min_granularity_ns = 100000
root@kali:~/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
----------------------------------------------------
    2048   0.54  63.98%   2.22%  33.79%  7904098.951
    4096   1.09  64.21%   2.21%  33.58%  7793991.860
    8192   2.15  64.00%   2.18%  33.82%  7913572.848
   16384   4.30  63.98%   2.19%  33.83%  7911027.011
   32768   8.61  64.00%   2.19%  33.81%  7893426.358
   65536  17.23  64.01%   2.19%  33.80%  7890053.678
```

Наше предположение оказалось верным, для большинства тестов вычислительная сложность оказалась больше. А значит нам удалось повлиять на работу планировщика на уровне ядра.