

**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»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
«Linpack»

Выполнил:

Суханкулиев Мухаммет,
студент группы N3246



(подпись)

Проверил:

Савков Сергей Витальевич,
инженер

(отметка о выполнении)

(подпись)

Санкт-Петербург
2024 г.

СОДЕРЖАНИЕ

Введение	3
1 linpack для Linux	4
1.1 Найти и скомпилировать программу linpack	4
Протестировать ее при различных режимах работы ОС:	4
1.2 С различными приоритетами задачи в планировщике	4
1.3 С наличием и отсутствием привязки к процессору.....	5
1.4 Запретить выполнение всех потоков кроме того, который тестируется (путем запрета прерываний)	5
1.5 Провести несколько тестов, сравнить результаты по 3 сигма	7
Заключение.....	9
Список использованных источников.....	10

ВВЕДЕНИЕ

Цель работы – оценить производительность компьютера, измерив количество операций с плавающей запятой в секунду (Flops) с помощью программы Linpack при различных режимах работы операционной системы.

Для достижения поставленной цели необходимо:

1. Найти и скомпилировать программу linpack;

Протестировать ее при различных режимах работы ОС:

2. С различными приоритетами задачи в планировщике;
3. С наличием и отсутствием привязки к процессору;

Плюс изменить параметры на уровне ядра:

4. Запретить выполнение всех потоков кроме того, который тестируется (путем запрета прерываний) (cli (Clear Interrupt Flag) и sti (Set Interrupt Flag)).
5. Провести несколько тестов, сравнить результаты по 3 сигма или другим статистическим критериям;

1 LINPACK ДЛЯ LINUX

Буду использовать виртуальную машину с 2 ГБ ОЗУ и 3 процессорами.

1.1 Найти и скомпилировать программу linpack

Скачиваем исходник netlib.org/benchmark/linpackc.new

```
(root@kali)-[/home/kali]
# cd /home/kali/Desktop/lab3/

(root@kali)-[/home/kali/Desktop/lab3]
# gcc -o linpack linpack.c -lm

(root@kali)-[/home/kali/Desktop/lab3]
# ./linpack
Enter array size (q to quit) [200]: 200
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
1024	1.04	88.83%	2.86%	8.31%	1477810.494
2048	2.11	88.76%	2.77%	8.47%	1459587.242
4096	4.27	88.54%	2.71%	8.75%	1443253.245
8192	8.52	88.29%	2.79%	8.92%	1449873.075
16384	17.14	88.38%	2.85%	8.77%	1438910.374

Рисунок 1 – Запуск при размере массива 200

Протестировать ее при различных режимах работы ОС:

1.2 С различными приоритетами задачи в планировщике

Изменение приоритета задачи с помощью **nice** (значения приоритета от -20 до 19)

```
(root@kali)-[/home/kali/Desktop/lab3]
# sudo nice -n -20 ./linpack

Enter array size (q to quit) [200]: 200
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
512	0.53	88.64%	2.81%	8.54%	1445108.960
1024	1.07	88.94%	2.69%	8.37%	1432330.232
2048	2.15	89.09%	2.70%	8.21%	1425767.897
4096	4.12	88.54%	2.79%	8.67%	1495079.369
8192	8.37	87.95%	3.38%	8.67%	1471660.828
16384	16.73	88.75%	2.78%	8.47%	1469023.866

```
(root@kali)-[/home/kali/Desktop/lab3]
# sudo nice -n 19 ./linpack

Enter array size (q to quit) [200]: 200
Memory required: 315K.

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

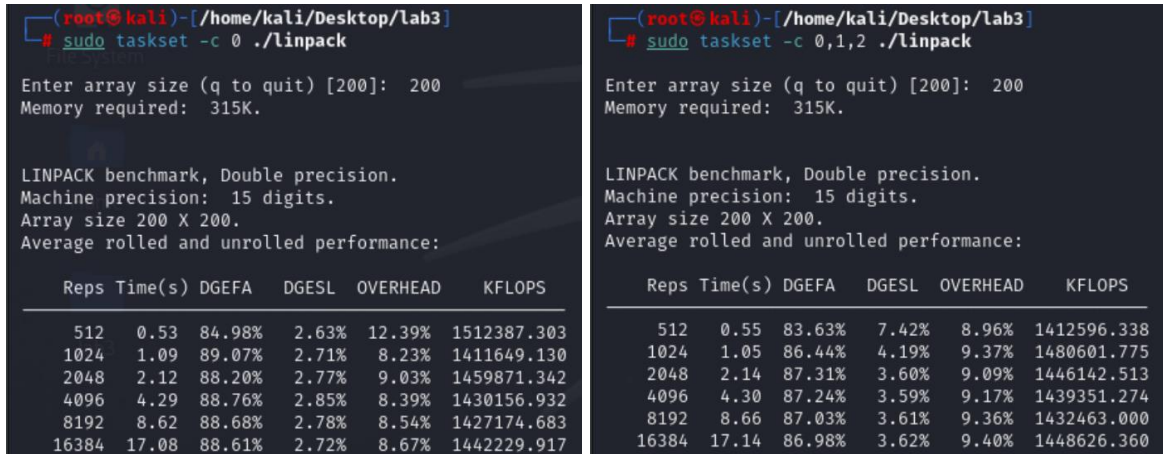
Reps	Time(s)	DGEFA	DGESL	OVERHEAD	KFLOPS
1024	0.96	88.52%	2.86%	8.62%	1610234.344
2048	1.88	88.55%	2.80%	8.65%	1638894.214
4096	3.77	88.53%	2.79%	8.68%	1635498.182
8192	7.53	88.55%	2.79%	8.66%	1634772.868
16384	15.08	88.57%	2.79%	8.65%	1633439.300

Рисунок 2 – Запуск с приоритетами -20 и 19

Вывод: Чем выше приоритет, тем больше вычислительных ресурсов доступно для выполнения теста (должно быть так).

1.3 С наличием и отсутствием привязки к процессору

Будем использовать команду `taskset` для установки привязки к конкретному процессору.



Reps	Time(s)	DGEFA	DGESL	OVERHEAD	KFLOPS
512	0.53	84.98%	2.63%	12.39%	1512387.303
1024	1.09	89.07%	2.71%	8.23%	1411649.130
2048	2.12	88.20%	2.77%	9.03%	1459871.342
4096	4.29	88.76%	2.85%	8.39%	1430156.932
8192	8.62	88.68%	2.78%	8.54%	1427174.683
16384	17.08	88.61%	2.72%	8.67%	1442229.917

Reps	Time(s)	DGEFA	DGESL	OVERHEAD	KFLOPS
512	0.55	83.63%	7.42%	8.96%	1412596.338
1024	1.05	86.44%	4.19%	9.37%	1480601.775
2048	2.14	87.31%	3.60%	9.09%	1446142.513
4096	4.30	87.24%	3.59%	9.17%	1439351.274
8192	8.66	87.03%	3.61%	9.36%	1432463.000
16384	17.14	86.98%	3.62%	9.40%	1448626.360

Рисунок 3 – Запуск с привязкой к первому процессору (0),

и с привязкой к трём процессорам (0,1,2)

Вывод: Иногда многопроцессорность увеличивает время на решение системы линейных уравнений (DGESL) из-за накладных расходов на синхронизацию, что делает её менее эффективной по сравнению с выполнением на одном процессоре.

1.4 Запретить выполнение всех потоков кроме того, который тестируется (путем запрета прерываний)

В С функции `cli()` и `sti()` не определены в пространстве пользовательского модуля ядра. Вместо них я использовал функции `local_irq_disable()` и `local_irq_enable()`, которые являются безопасными аналогами.

Для этого напишем модуль ядра.

```
sudo apt-get install build-essential linux-headers-$(uname -r).
```

Создадим файл `my_module.c`

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/interrupt.h>
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Muhammet");
```

```

MODULE_DESCRIPTION("Kernel module to disable interrupts");
static int __init my_module_init(void) {
    printk(KERN_INFO "Disabling interrupts...\n");
    local_irq_disable();
    return 0;
}
static void __exit my_module_exit(void) {
    printk(KERN_INFO "Enabling interrupts...\n");
    local_irq_enable();
}
module_init(my_module_init);
module_exit(my_module_exit);

```

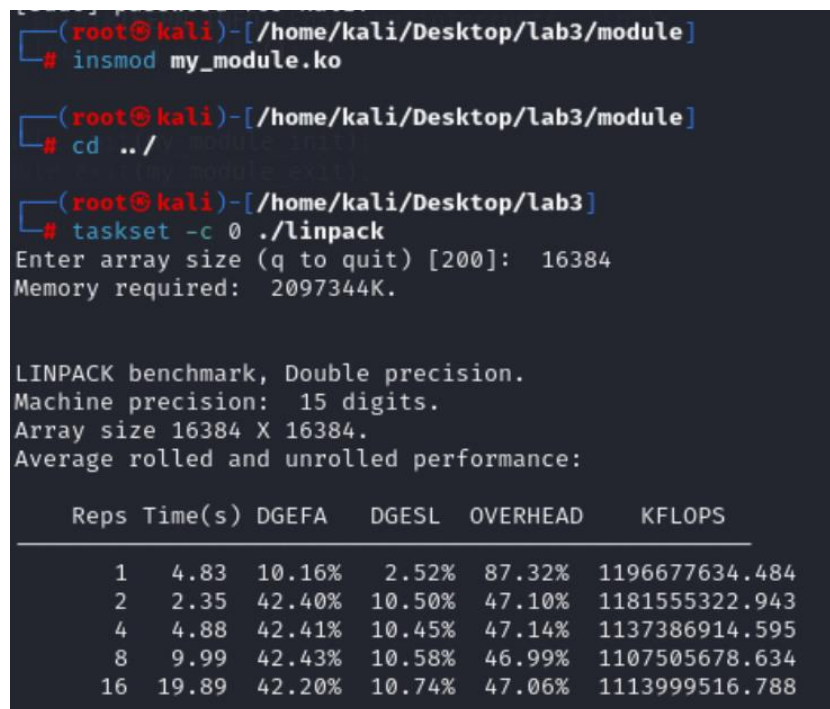
И Makefile

```

obj-m += my_module.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

```

Модуль отключает только локальные прерывания на одном процессоре, поэтому запустим linpack с **taskset**.



```

(root@kali)-[/home/kali/Desktop/lab3/module]
# insmod my_module.ko

(root@kali)-[/home/kali/Desktop/lab3/module]
# cd ../

(root@kali)-[/home/kali/Desktop/lab3]
# taskset -c 0 ./linpack
Enter array size (q to quit) [200]: 16384
Memory required: 2097344K.

LINPACK benchmark, Double precision.
Machine precision: 15 digits.
Array size 16384 X 16384.
Average rolled and unrolled performance:

```

	Reps	Time(s)	DGEFA	DGESL	OVERHEAD	KFLOPS
	1	4.83	10.16%	2.52%	87.32%	1196677634.484
	2	2.35	42.40%	10.50%	47.10%	1181555322.943
	4	4.88	42.41%	10.45%	47.14%	1137386914.595
	8	9.99	42.43%	10.58%	46.99%	1107505678.634
	16	19.89	42.20%	10.74%	47.06%	1113999516.788

Рисунок 4 – Запуск с запретом прерываний процессора

Вывод: Отключение прерываний должно немного повышать производительность с вычислительными задачами.

1.5 Провести несколько тестов, сравнить результаты по 3 сигма

Теперь запустим все с 1.5 ГБ ОЗУ и 2 процессорами. Запустим все по три раза и запишем данные KFLOPS. Размер массива будем определять 16384 (Записывать будем при Reps 8).

№	KFLOPS	Среднее	Стандартное отклонение
./linpack			
1.	997163387.095	994170145,003	36214086,793
2.	1018207754.254		
3.	967139293.660		
nice -n -20 ./linpack			
1.	907652025.272	893790872,644	12984933,513
2.	888921712.290		
3.	884798880.369		
nice -n 19 ./linpack			
1.	915977950.867	994672501,334	91133991,138
2.	994366839.856		
3.	1073672713.279		
taskset -c 0 ./linpack			
1.	852123193.854	923214292,229	70190861,109
2.	939850362.846		
3.	977669319.988		
taskset -c 0,1 ./linpack			
1.	866998452.206	878777495,812	41656094,290
2.	913124078.554		
3.	856209956.676		
taskset -c 0 ./linpack при insmod my_module.ko			
1.	870443116.059	889139577,403	72786052,468
2.	848455261.776		
3.	948520354.373		

Границы 3 сигма:

./linpack: $948,115 \pm 103,609$ Tflops, $\varepsilon = 10,928\%$

nice -n -20 ./linpack: $852,385 \pm 37,150$ Tflops, $\varepsilon = 4,358\%$ ↓

nice -n 19 ./linpack: $948,594 \pm 260,736$ Tflops, $\varepsilon = 27,487\%$ ↑

taskset -c 0 ./linpack: $880,446 \pm 200,818$ Tflops, $\varepsilon = 22,809\%$ ↓

taskset -c 0,1 ./linpack: $838,068 \pm 119,179$ Tflops, $\varepsilon = 14,221\%$ ↓

taskset -c 0 ./linpack при insmod my_module.ko: $847,950 \pm 208,243$ Tflops, $\varepsilon = 24,558\%$ ↓

Повторим тесты для 8 ГБ ОЗУ и 4 процессоров.

№	KFLOPS	Среднее	Стандартное отклонение
./linpack			
1.	1150092831.185	1112947477,177	35587217,026
2.	1086738191.676		
3.	1102011408.671		

nice -n -20 ./linpack			
1.	1158310123.884	1169450449,380	31673816,889
2.	1153809668.024		
3.	1196231556.231		
nice -n 19 ./linpack			
1.	1165240426.728	1148179087,409	33075507,829
2.	1119015739.761		
3.	1160281095.739		
taskset -c 0 ./linpack			
1.	1034350549.169	1099324863,666	61088397,337
2.	1142149839.889		
3.	1121474201.939		
taskset -c 0,1 ./linpack			
1.	978457604.651	1085215074,578	97455820,604
2.	1138539887.057		
3.	1138647732.026		
taskset -c 0,1,2,3 ./linpack			
1.	1221413215.497	1112788293,452	147153875,727
2.	1135357077.652		
3.	981594587.208		
taskset -c 0 ./linpack при insmod my_module.ko			
1.	980634661.348	1086625613,800	97173098,037
2.	1133262002.799		
3.	1145980177.254		

Границы 3 сигма:

./linpack: $1061,389 \pm 101,816$ Tflops, $\varepsilon = 9,593\%$

nice -n -20 ./linpack: $1115,275 \pm 90,620$ Tflops, $\varepsilon = 8,125\%$ ↑

nice -n 19 ./linpack: $1094,989 \pm 94,630$ Tflops, $\varepsilon = 8,642\%$ ↑

taskset -c 0 ./linpack: $1048,398 \pm 174,775$ Tflops, $\varepsilon = 16,671\%$ ↓

taskset -c 0,1 ./linpack: $1034,942 \pm 278,823$ Tflops, $\varepsilon = 26,941\%$ ↓

taskset -c 0,1,2,3 ./linpack: $1061,238 \pm 421,011$ Tflops, $\varepsilon = 39,672\%$ ≈

taskset -c 0 ./linpack при insmod my_module.ko: $1036,287 \pm 278,014$ Tflops, $\varepsilon = 26,828\%$ ↓

Запустим тесты по 50 раз. Более точные результаты:

./linpack: $1109,879 \pm 51,429$ Tflops, $\varepsilon = 4,634\%$

nice -n -20 ./linpack: $1108,961 \pm 33,906$ Tflops, $\varepsilon = 3,057\%$ ↓

nice -n 19 ./linpack: $1103,645 \pm 115,074$ Tflops, $\varepsilon = 10,427\%$ ↓

taskset -c 0 ./linpack: $1083,942 \pm 6,124$ Tflops, $\varepsilon = 0,565\%$ ↓

taskset -c 0,1 ./linpack: $1094,317 \pm 7,288$ Tflops, $\varepsilon = 0,666\%$ ↓

taskset -c 0,1,2,3 ./linpack: $1118,646 \pm 25,729$ Tflops, $\varepsilon = 2,300\%$ ↑

taskset -c 0 ./linpack при insmod my_module.ko: $1073,940 \pm 6,773$ Tflops, $\varepsilon = 0,631\%$ ↓

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы была проведена оценка производительности компьютера с использованием программы Linpack для измерения количества операций с плавающей запятой в секунду при различных режимах работы операционной системы.

Однозначные выводы на основе тестов на моей машине получить не удалось. Однако в теории, при **высоком приоритете производительность увеличивается**, при **использовании нескольких процессоров производительность увеличивается**, при условии что программа корректно использует многопоточность, при **отключении прерываний так же должна увеличиться производительность** в контексте задачи, но при этом могут появиться системные ошибки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. [GitHub - ereyes01/linpack](#)
2. netlib.org/benchmark/linpackc.new
3. [Презентация PowerPoint \(sibsutis.ru\)](#)
4. [How To Change Linux Process Priority | by The Anshuman | Medium](#)
5. [Как использовать команду taskset в Linux \(andreyex.ru\)](#)
6. [Unreliable Guide To Hacking The Linux Kernel — The Linux Kernel documentation](#)