Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-петербургский государственный политехнический университет»

Институт компьютерных наук и технологий

Высшая школа программной инженерии

**ОТЧЁТ**

По дисциплине «Системное программное обеспечение GNU/LINUX»

На тему «Конфигурация и установка ядра Linux».

|  |  |
| --- | --- |
|  | Выполнил студент группы 3530904/10002  И. С. Тампио  Принял к.э.н., доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_20\_\_ года |

Санкт-Петербург

2022

Оглавление

[Введение 3](#_Toc96615724)

[Основная часть 4](#_Toc96615725)

[1.1 Подготовка системы к сборке ядра и сборка данных о программной и аппаратных платформах 4](#_Toc96615726)

[1.2 Выбор ядра 4](#_Toc96615727)

[1.3 Загрузка исходного кода ядра 4](#_Toc96615728)

[1.4 Конфигурация ядра 4](#_Toc96615729)

[1.5 Сборка ядра (make) 4](#_Toc96615730)

[1.6 Написать скрипт для сборки ядра при разном количестве потоков 5](#_Toc96615731)

[1.7 Построить график времени сборки в зависимости от числа потоков сборки 7](#_Toc96615732)

[2.1 Установка ядра 8](#_Toc96615733)

[2.1.1 Сборка модулей ядра (make modules) 8](#_Toc96615734)

[2.1.2 Установка модулей ядра (make modules\_install) 8](#_Toc96615735)

[2.1.3 Добавить ядро в список доступных для загрузки: 9](#_Toc96615736)

[2.1.4 Копирование System.map 10](#_Toc96615737)

[2.1.5 Добавление в загрузчик 10](#_Toc96615738)

[2.2 Использование нового ядра с помощью kexec (Индивидуальное задание) 12](#_Toc96615739)

[2.3 Тестирование нового ядра 14](#_Toc96615740)

[Заключение 19](#_Toc96615741)

[Список использованных источников 20](#_Toc96615742)

# Введение

Целью работы является конфигурация и сборка ядра Linux, с попутным сбором данных о платформе и времени сборке ядра при различном числе потоков сборки.

Для выполнения работы требуется выполнить следующие задачи:

1. Подготовка системы к сборке ядра и сборка данных о программной и аппаратных платформах
2. Выбор ядра
3. Загрузка исходного кода ядра
4. Конфигурация ядра
5. Сборка ядра (make)
6. Написать скрипт для сборки ядра при разном количестве потоков
7. Построить график времени сборки в зависимости от числа потоков сборки
8. Установка ядра
   * Сборка модулей ядра (make modules)
   * Установка модулей ядра (make modules\_install)
9. Передача управления новому ядру с помощью kexec
10. Тестирование нового ядра

# Основная часть

## Подготовка системы к сборке ядра и сборка данных о программной и аппаратных платформах

Информация об аппаратной и программных платформах представлена в выводе команды neofetch:

OS: ArcoLinux

Host: HP Pavilion Notebook

Kernel: 5.16.0-pf5+

Uptime: 42 mins

Packages: 1719 (pacman)

Shell: bash 5.1.16

WM: awesome

Theme: Arc-Dark [GTK2/3]

Icons: Sardi-Arc [GTK2/3]

Terminal: alacritty

CPU: Intel i3-5010U (4) @ 2.000GHz

GPU: Intel HD Graphics 5500

GPU: AMD ATI Radeon R7 M260/M265 / M340/M360 / M440/M445 / 530/53

Memory: 736MiB / 3852MiB (19%)

GPU Driver: Hewlett-Packard Company Device [103c:8099]

Disk (/): 57G / 63G (97%)

Locale: en\_US.UTF-8

## Выбор ядра

В качестве устанавливаемого ядра было выбрано ядро linux-zen. Оно содержит патчи для оптимизации. С технической точки зрения интересно сравнить – насколько сильно по производительности оно отличается от pfkernel.

## Загрузка исходного кода ядра

С помощью команды git clone <https://github.com/zen-kernel/zen-kernel> скачиваем исходный код ядра последней версии (5.16.11-lqx1).

## Конфигурация ядра

Прежде чем начать сборку необходимо сконфигурировать новое ядро. Для этого создадим конфиг на основе старого с помощью make olddefconfig.

[quakumei@quakumei-lite zen-kernel]$ make olddefconfig

#

# configuration written to .config

#

## Сборка ядра (make)

Чтобы собрать ядро, выполним команду make -j$(nproc). Флаг -j позволяет использовать несколько потоков для сборки, тем самым ускоряя процесс.

Начался процесс сборки ядра:

[quakumei@quakumei-lite zen-kernel]$ time make -j$(nproc)

SYSHDR  arch/x86/include/generated/uapi/asm/unistd\_32.h

SYSHDR  arch/x86/include/generated/uapi/asm/unistd\_64.h

SYSHDR  arch/x86/include/generated/uapi/asm/unistd\_x32.h

SYSTBL  arch/x86/include/generated/asm/syscalls\_32.h

SYSHDR  arch/x86/include/generated/asm/unistd\_32\_ia32.h

SYSHDR  arch/x86/include/generated/asm/unistd\_64\_x32.h

WRAP    arch/x86/include/generated/uapi/asm/bpf\_perf\_event.h

Закончился:

<…>

Kernel: arch/x86/boot/bzImage is ready  (#1)

real  18m47.637s

user  62m27.927s

sys  4m21.592s

## Написать скрипт для сборки ядра при разном количестве потоков

build\_many.sh:

#!/usr/bin/env bash

count=$(($(nproc --all) \* 2 + 1))

for(( i = 1; i <= $count; i++ ))

do

time=$(time make -j $i > /dev/null)

make clean > /dev/null

echo $time

done

Результат исполнения:

[quakumei@quakumei-lite zen-kernel]$ time ../\_Scripts/build\_many.sh

real  95m24.530s

user  88m26.205s

sys  7m1.947s

real  50m27.948s

user  92m39.722s

sys  7m20.329s

real  46m16.571s

user  125m20.816s

sys  9m6.664s

real  43m12.324s

user  148m20.158s

sys  10m23.194s

real  43m28.853s

user  149m21.336s

sys  10m19.711s

real  43m47.234s

user  149m48.908s

sys  10m18.941s

real  44m3.060s

user  150m59.963s

sys  10m19.291s

real  44m13.475s

user  151m44.073s

sys  10m29.025s

real  44m19.688s

user  152m11.316s

sys  10m25.950s

real  458m9.338s

user  1210m55.343s

sys  86m45.719s

## Построить график времени сборки в зависимости от числа потоков сборки

Внесем данные в MS Excel и построим диаграмму:

Из данных видно, что наилучшее время сборки достигается при использовании такого же количества потоков, как и количества ядер процессора, т.е. 4. Стоит отметить, что при отдельной компиляции на четырех потоках (если перезапустить систему и начать компиляцию, не компилируя предварительно на 1 потоке) процесс занимает в более чем 2 раза меньше времени (было 18 минут минимум).

## Установка ядра

Чтобы добавить ядро в список доступных для загрузки, воспользуемся руководством с сайта дистрибутива: <https://wiki.archlinux.org/title/Kernel/Traditional_compilation> (секции 4-5)

### Сборка модулей ядра (make modules)

Производим сборку модулей ядра, также засекаем с помощью time:

[quakumei@quakumei-lite zen-kernel]$ time make modules

CALL    scripts/checksyscalls.sh

CALL    scripts/atomic/check-atomics.sh

DESCEND objtool

^Cmake[1]: \*\*\* [scripts/Makefile.build:549: drivers/media] Interrupt

make: \*\*\* [Makefile:1841: drivers] Interrupt

real  0m22.139s

user  0m16.750s

sys  0m5.801s

[quakumei@quakumei-lite zen-kernel]$ time make modules -j$(nproc)

DESCEND objtool

CALL    scripts/atomic/check-atomics.sh

CALL    scripts/checksyscalls.sh

real  0m16.143s

user  0m38.077s

sys  0m11.506s

Я сначала забыл включить все ядра, но вероятно модули успели скомпилироваться частично, поэтому привожу полный лог.

### Установка модулей ядра (make modules\_install)

Для установки модулей ядра, пишем make modules\_install:

[quakumei@quakumei-lite zen-kernel]$ time sudo  make modules\_install -j$(nproc)

INSTALL /lib/modules/5.16.11-zen+/kernel/drivers/thermal/intel/x86\_pkg\_temp\_thermal.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/fs/efivarfs/efivarfs.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/net/ipv4/netfilter/iptable\_nat.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/net/netfilter/nf\_log\_syslog.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/net/netfilter/xt\_LOG.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/net/netfilter/xt\_MASQUERADE.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/net/netfilter/xt\_addrtype.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/net/netfilter/xt\_mark.ko

INSTALL /lib/modules/5.16.11-zen+/kernel/net/netfilter/xt\_nat.ko

DEPMOD  /lib/modules/5.16.11-zen+

real  0m2.689s

user  0m1.549s

sys  0m1.474s

### Добавить ядро в список доступных для загрузки

Для этого необходимо сначала скопировать образ ядра в /boot. После этого необходимо создать iniramfs для загрузки ядра с помощью mkinitcpio. Для этого мы копируем стандартный конфиг из /etc/mkinitcpio.d/linux.preset, а после редактируем его, чтобы он соответствовал нашему ядру. После этого запускаем mkinitcpio с названием нового ядра в качестве аргумента.

[quakumei@quakumei-lite zen-kernel]$ sudo cp -v arch/x86/boot/bzImage /boot/vmlinuz-linux-zen-5.16.11-lqx1

'arch/x86/boot/bzImage' -> '/boot/vmlinuz-linux-zen-5.16.11-lqx1'

[quakumei@quakumei-lite zen-kernel]$ sudo cp /etc/mkinitcpio.d/linux.preset /etc/mkinitcpio.d/linux-zen-5.16.11-lqx1.preset

[quakumei@quakumei-lite zen-kernel]$ sudo nvim /etc/mkinitcpio.d/linux-zen-5.16.11-lqx1.preset

[quakumei@quakumei-lite zen-kernel]$ mkinitcpio -p linux-zen-5.16.11-lqx1

==> Building image from preset: /etc/mkinitcpio.d/linux-zen-5.16.11-lqx1.preset: 'default'

-> -k /boot/vmlinuz-linux-zen-5.16.11-lqx1 -c /etc/mkinitcpio.conf -g /boot/initramfs-linuxi-zen-5.16.11-lqx1.img

==> ERROR: Unable to write to /boot/initramfs-linuxi-zen-5.16.11-lqx1.img

==> Building image from preset: /etc/mkinitcpio.d/linux-zen-5.16.11-lqx1.preset: 'fallback'

-> -k /boot/vmlinuz-linux-zen-5.16.11-lqx1 -c /etc/mkinitcpio.conf -g /boot/initramfs-linux-zen-5.16.11-lqx1-fallback.img -S autodetect

==> ERROR: Unable to write to /boot/initramfs-linux-zen-5.16.11-lqx1-fallback.img

[quakumei@quakumei-lite zen-kernel]$ sudo mkinitcpio -p linux-zen-5.16.11-lqx1

==> Building image from preset: /etc/mkinitcpio.d/linux-zen-5.16.11-lqx1.preset: 'default'

-> -k /boot/vmlinuz-linux-zen-5.16.11-lqx1 -c /etc/mkinitcpio.conf -g /boot/initramfs-linuxi-zen-5.16.11-lqx1.img

==> Starting build: 5.16.11-zen+

-> Running build hook: [base]

-> Running build hook: [udev]

-> Running build hook: [autodetect]

-> Running build hook: [modconf]

-> Running build hook: [block]

-> Running build hook: [keyboard]

-> Running build hook: [keymap]

-> Running build hook: [consolefont]

-> Running build hook: [filesystems]

-> Running build hook: [fsck]

==> WARNING: No modules were added to the image. This is probably not what you want.

==> Creating zstd-compressed initcpio image: /boot/initramfs-linuxi-zen-5.16.11-lqx1.img

==> Image generation successful

==> Building image from preset: /etc/mkinitcpio.d/linux-zen-5.16.11-lqx1.preset: 'fallback'

-> -k /boot/vmlinuz-linux-zen-5.16.11-lqx1 -c /etc/mkinitcpio.conf -g /boot/initramfs-linux-zen-5.16.11-lqx1-fallback.img -S autodetect

==> Starting build: 5.16.11-zen+

-> Running build hook: [base]

-> Running build hook: [udev]

-> Running build hook: [modconf]

-> Running build hook: [block]

-> Running build hook: [keyboard]

-> Running build hook: [keymap]

-> Running build hook: [consolefont]

-> Running build hook: [filesystems]

-> Running build hook: [fsck]

==> Generating module dependencies

==> Creating zstd-compressed initcpio image: /boot/initramfs-linux-zen-5.16.11-lqx1-fallback.img

==> Image generation successful

[quakumei@quakumei-lite zen-kernel]$

### Копирование System.map

Несмотря на то, что System.map не требуется для запуска системы, его стоит добавить так как он содержит символьную информацию о ядре. Это значит, что в случае возникновения ошибок ядра, оно сможет вывести информацию о том, что сломалось.

Для этого поместим System.map в /boot и поменяем символьную ссылку.

[quakumei@quakumei-lite zen-kernel]$ sudo cp System.map /boot/System.map-linux-zen-5.16.11-lqx1

[quakumei@quakumei-lite zen-kernel]$ sudo ln -sf /boot/System.map-linux-zen-5.16.11-lqx1 /boot/System.map

### Добавление в загрузчик

Чтобы новое ядро появилось в загрузчике GRUB, пропишем update-grub

[quakumei@quakumei-lite zen-kernel]$ update-grub

Generating grub configuration file ...

Found theme: /boot/grub/themes/Vimix/theme.txt

Found linux image: /boot/vmlinuz-pfkernel1

Found initrd image: /boot/intel-ucode.img /boot/initramfs-pfkernel1.img

Found fallback initrd image(s) in /boot:  intel-ucode.img initramfs-pfkernel1-fallback.img

Found linux image: /boot/vmlinuz-linux-zen-5.16.11-lqx1

Found initrd image: /boot/intel-ucode.img

Found fallback initrd image(s) in /boot:  intel-ucode.img initramfs-linux-zen-5.16.11-lqx1-fallback.img

Found linux image: /boot/vmlinuz-linux

Found initrd image: /boot/intel-ucode.img /boot/initramfs-linux.img

Found fallback initrd image(s) in /boot:  intel-ucode.img initramfs-linux-fallback.img

Warning: os-prober will be executed to detect other bootable partitions.

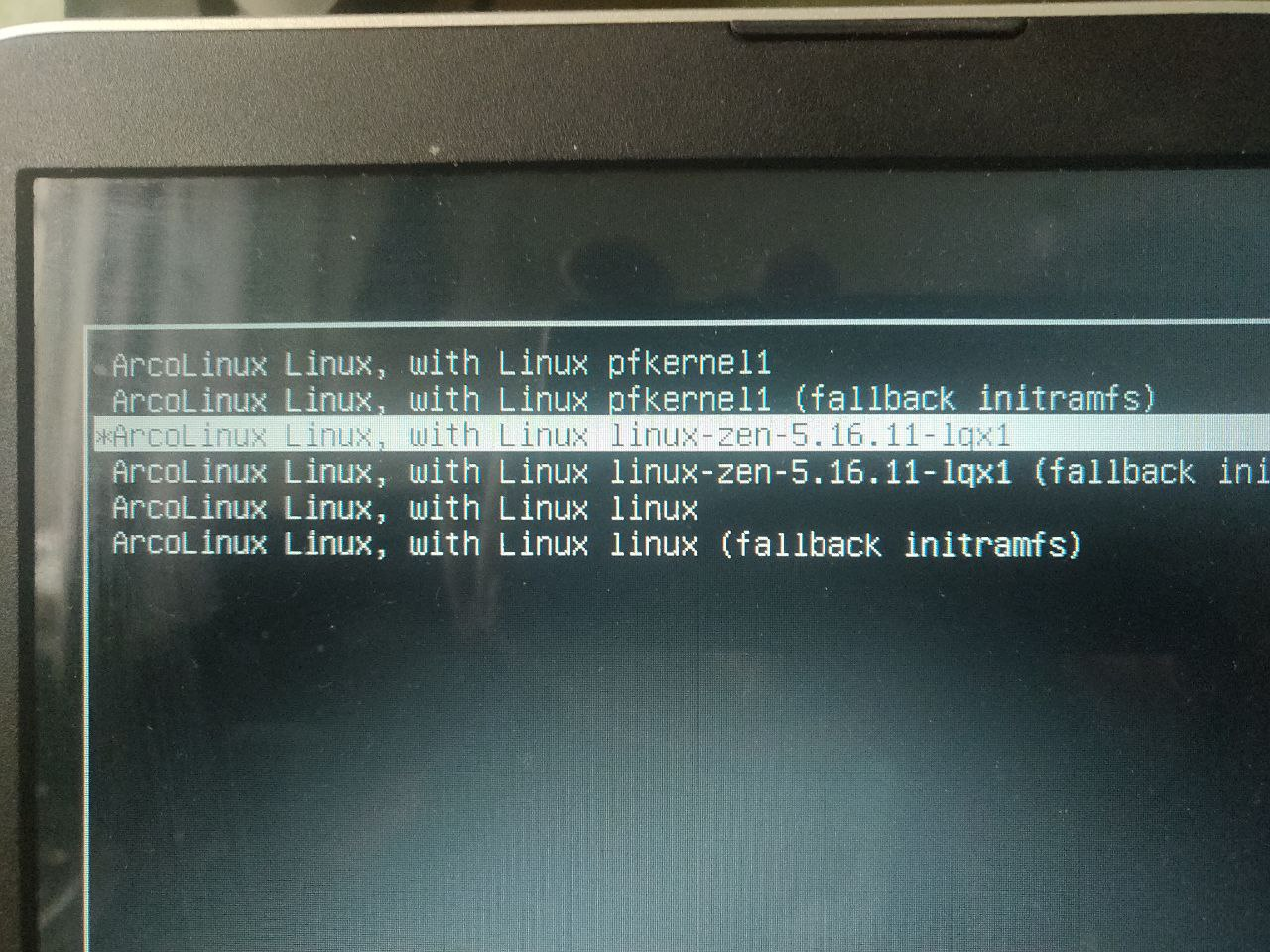
Its output will be used to detect bootable binaries on them and create new boot entries.

Found Windows Boot Manager on /dev/sda2@/EFI/Microsoft/Boot/bootmgfw.efi

Adding boot menu entry for UEFI Firmware Settings ...

done

[quakumei@quakumei-lite zen-kernel]$

На этом установку ядра можно считать завершенной. Теперь можно загрузиться в систему с использованием этого ядра.

## Использование нового ядра с помощью kexec (Индивидуальное задание)

Перед использованием kexec, необходимо установить пакет, включающий эту программу. В моём случае это выполнялось с помощью pacman -Syu kexec-tools

Для того чтобы передать управление новому ядру сначала загрузим новое ядро в kexec:

[quakumei@quakumei-lite zen-kernel]$ sudo kexec -l /boot/vmlinuz-linux-zen-5.16.11-lqx1 --append=/proc/cmdline

А потом передадим управление:

[quakumei@quakumei-lite zen-kernel]$ kexec -e

В результате выполнения этой операции у меня погас экран компьютера, а индикатор CAPS LOCK начал активно мигать. Мигает он непрерывно.

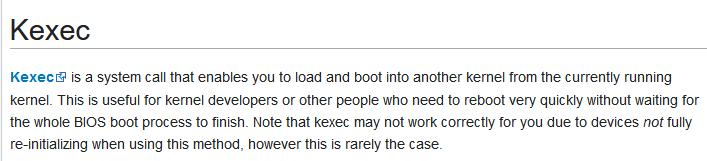
На сайте HP говорится, что в ноутбуках HP Pavilion caps-lock служит заменой звуковым сигналам системы. Изучив таблицу проблем, возникло несколько предположений. Либо это одно постоянное моргание – неисправность центрального процессора. Однако при перезапуске компьютера можно без проблем загрузиться в ядро, так что этот вариант отпадает. Остальные случаи ошибок описывают переменное мигание на ноутбуках, в связи с чем можно с уверенностью сказать, что мигание вызвано не ошибкой аппаратной платформы.

Моё предположение заключается в том, что при попытке передать управление с помощью kexec, возникает так называемый bootloop. При попытке инициализировать части компьютера возникает ошибка, процесс начинается заново, в связи с чем появляется мигание от включения\выключения клавиатуры.

С другой стороны – caps lock по стандарту отключен и при нажатии на загоревшийся caps lock он не выключается, что может говорить о том, что проблема заключается в другом.

Подождав около 10 минут, я всё же решил перезапустить ноутбук.

Посмотрим описание kexec на сайте ArchLinux: <https://wiki.archlinux.org/title/kexec>



В данном отрывке говорится, что kexec может работать неправильно, из-за того что устройства не полностью переинициализируются. Несмотря на то, что здесь говорится о том, что это происходит довольно редко, у меня уже были проблемы с переинициализацией устройств. При вхождении в режим гибернации на компьютере отключался wifi, при этом не как компонент системы, а на уровне аппаратуры – при попытке подключиться к wifi по usb адаптеру, система говорит о том, что wifi отключен на уровне аппаратуры и не давал его использовать, вне зависимости от linux дистрибутива или ядра.

## Тестирование нового ядра

Запустившись в систему с использованием нового ядра, проверим, действительно ли это так.

[quakumei@quakumei-lite ~]$ neofetch --off

quakumei@quakumei-lite

**----------------------**

OS: ArcoLinux

Host: HP Pavilion Notebook

Kernel: 5.16.11-zen+

Uptime: 1 min

Packages: 1724 (pacman)

Shell: bash 5.1.16

WM: awesome

Theme: Arc-Dark [GTK2/3]

Icons: Sardi-Arc [GTK2/3]

Terminal: alacritty

CPU: Intel i3-5010U (4) @ 2.000GHz

GPU: Intel HD Graphics 5500

GPU: AMD ATI Radeon R7 M260/M265 / M340/M360 / M440/M445 / 530/535 / 620/625 Mobile

Memory: 413MiB / 3852MiB (10%)

GPU Driver: Hewlett-Packard Company Device [103c:8099]

Disk (/): 41G / 63G (70%)

Locale: en\_US.UTF-8

Стоит отметить, что графическая оболочка загрузилась без проблем, а ядро соответствует устанавливаемому. Следовательно, можно приступать к тестированию.

Для тестирования будем использовать утилиту unixbench. Тесты для ядер linux и pfkernel1 я проводил заранее.

Утилита unixbench доступна после клонирования и сборки:

[quakumei@quakumei-lite Repos]$ git clone https://github.com/kdlucas/byte-unixbench

fatal: destination path 'byte-unixbench' already exists and is not an empty directory.

[quakumei@quakumei-lite Repos]$ cd byte-unixbench/

[quakumei@quakumei-lite byte-unixbench]$ git pull

remote: Enumerating objects: 18, done.

remote: Counting objects: 100% (18/18), done.

remote: Compressing objects: 100% (7/7), done.

remote: Total 18 (delta 11), reused 18 (delta 11), pack-reused 0

Unpacking objects: 100% (18/18), 2.91 KiB | 99.00 KiB/s, done.

From https://github.com/kdlucas/byte-unixbench

070030e..e477bc0  master     -> origin/master

Updating 070030e..e477bc0

Fast-forward

UnixBench/Run           | 2 +-

UnixBench/src/big.c     | 2 +-

UnixBench/src/syscall.c | 5 +++--

3 files changed, 5 insertions(+), 4 deletions(-)

[quakumei@quakumei-lite byte-unixbench]$ ls

LICENSE.txt  README.md  UnixBench

[quakumei@quakumei-lite byte-unixbench]$ cd UnixBench/

[quakumei@quakumei-lite UnixBench]$ make

make distr

make[1]: Entering directory '/home/quakumei/Repos/byte-unixbench/UnixBench'

Checking distribution of files

./pgms  exists

./src  exists

./testdir  exists

./tmp  exists

./results  exists

make[1]: Leaving directory '/home/quakumei/Repos/byte-unixbench/UnixBench'

make programs

make[1]: Entering directory '/home/quakumei/Repos/byte-unixbench/UnixBench'

gcc -o pgms/syscall -Wall -pedantic -O3 -ffast-math -march=native -mtune=native -I ./src -DTIME src/syscall.c

gcc -o pgms/execl -Wall -pedantic -O3 -ffast-math -march=native -mtune=native -I ./src -DTIME src/execl.c

make[1]: Leaving directory '/home/quakumei/Repos/byte-unixbench/UnixBench'

[quakumei@quakumei-lite UnixBench]$ ls

Makefile  README   Run  testdir  USAGE

pgms      results  src  tmp      WRITING\_TESTS

[quakumei@quakumei-lite UnixBench]$

Для запуска теста пропишем ./Run и получаем следующие результаты. Для краткости приведу только сведения о процессоре и тесты при использовании нескольких потоков.

[quakumei@quakumei-lite UnixBench]$ sudo ./Run

<. . . >

========================================================================

BYTE UNIX Benchmarks (Version 5.1.3)

System: quakumei-lite: GNU/Linux

OS: GNU/Linux -- 5.16.11-zen+ -- #12 ZEN SMP PREEMPT Thu Feb 24 12:15:31 MSK 2022

Machine: x86\_64 (unknown)

Language: en\_US.utf8 (charmap="UTF-8", collate="UTF-8")

CPU 0: Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz (4190.4 bogomips)

Hyper-Threading, x86-64, MMX, Physical Address Ext, SYSENTER/SYSEXIT, SYSCALL/SYSRET, Intel virtualization

CPU 1: Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz (4190.4 bogomips)

Hyper-Threading, x86-64, MMX, Physical Address Ext, SYSENTER/SYSEXIT, SYSCALL/SYSRET, Intel virtualization

CPU 2: Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz (4190.4 bogomips)

Hyper-Threading, x86-64, MMX, Physical Address Ext, SYSENTER/SYSEXIT, SYSCALL/SYSRET, Intel virtualization

CPU 3: Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz (4190.4 bogomips)

Hyper-Threading, x86-64, MMX, Physical Address Ext, SYSENTER/SYSEXIT, SYSCALL/SYSRET, Intel virtualization

14:20:28 up 9 min,  1 user,  load average: 0.01, 0.14, 0.11; runlevel

------------------------------------------------------------------------

<…>

------------------------------------------------------------------------

Benchmark Run: Thu Feb 24 2022 14:48:28 - 15:16:34

4 CPUs in system; running 4 parallel copies of tests

Dhrystone 2 using register variables       23736773.7 lps   (10.0 s, 7 samples)

Double-Precision Whetstone                     5842.5 MWIPS (10.2 s, 7 samples)

Execl Throughput                               2790.7 lps   (30.0 s, 2 samples)

File Copy 1024 bufsize 2000 maxblocks        366782.2 KBps  (30.0 s, 2 samples)

File Copy 256 bufsize 500 maxblocks           98761.1 KBps  (30.0 s, 2 samples)

File Copy 4096 bufsize 8000 maxblocks       1010086.0 KBps  (30.0 s, 2 samples)

Pipe Throughput                              470147.7 lps   (10.0 s, 7 samples)

Pipe-based Context Switching                 126792.0 lps   (10.0 s, 7 samples)

Process Creation                               8072.5 lps   (30.0 s, 2 samples)

Shell Scripts (1 concurrent)                   2416.0 lpm   (60.1 s, 2 samples)

Shell Scripts (8 concurrent)                    346.9 lpm   (60.2 s, 2 samples)

System Call Overhead                         266202.0 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX

Dhrystone 2 using register variables         116700.0   23736773.7   2034.0

Double-Precision Whetstone                       55.0       5842.5   1062.3

Execl Throughput                                 43.0       2790.7    649.0

File Copy 1024 bufsize 2000 maxblocks          3960.0     366782.2    926.2

File Copy 256 bufsize 500 maxblocks            1655.0      98761.1    596.7

File Copy 4096 bufsize 8000 maxblocks          5800.0    1010086.0   1741.5

Pipe Throughput                               12440.0     470147.7    377.9

Pipe-based Context Switching                   4000.0     126792.0    317.0

Process Creation                                126.0       8072.5    640.7

Shell Scripts (1 concurrent)                     42.4       2416.0    569.8

Shell Scripts (8 concurrent)                      6.0        346.9    578.2

System Call Overhead                          15000.0     266202.0    177.5

========

System Benchmarks Index Score                                         653.4

Если сравнить с ядром, поставляемым с системой (5.16.8-arch1-1):

BYTE UNIX Benchmarks (Version 5.1.3)

System: quakumei-lite: GNU/Linux

OS: GNU/Linux -- 5.16.8-arch1-1 -- #1 SMP PREEMPT Tue, 08 Feb 2022 21:21:08 +0000

Machine: x86\_64 (unknown)

Language: en\_US.utf8 (charmap="UTF-8", collate="UTF-8")

<...>

**------------------------------------------------------------------------**

Benchmark Run: Sun Feb 13 2022 04:24:30 - 04:52:43

4 CPUs in system; running 4 parallel copies of tests

Dhrystone 2 using register variables       61480911.2 lps   (10.0 s, 7 samples)

Double-Precision Whetstone                    14732.8 MWIPS (9.9 s, 7 samples)

Execl Throughput                               6412.3 lps   (29.9 s, 2 samples)

File Copy 1024 bufsize 2000 maxblocks        883722.9 KBps  (30.0 s, 2 samples)

File Copy 256 bufsize 500 maxblocks          246128.4 KBps  (30.0 s, 2 samples)

File Copy 4096 bufsize 8000 maxblocks       2149476.3 KBps  (30.0 s, 2 samples)

Pipe Throughput                             1294793.6 lps   (10.0 s, 7 samples)

Pipe-based Context Switching                 312592.4 lps   (10.0 s, 7 samples)

Process Creation                              17515.4 lps   (30.0 s, 2 samples)

Shell Scripts (1 concurrent)                   5755.2 lpm   (60.0 s, 2 samples)

Shell Scripts (8 concurrent)                    820.4 lpm   (60.1 s, 2 samples)

System Call Overhead                         694260.9 lps   (10.0 s, 7 samples)

System Benchmarks Index Values               BASELINE       RESULT    INDEX

Dhrystone 2 using register variables         116700.0   61480911.2   5268.3

Double-Precision Whetstone                       55.0      14732.8   2678.7

Execl Throughput                                 43.0       6412.3   1491.2

File Copy 1024 bufsize 2000 maxblocks          3960.0     883722.9   2231.6

File Copy 256 bufsize 500 maxblocks            1655.0     246128.4   1487.2

File Copy 4096 bufsize 8000 maxblocks          5800.0    2149476.3   3706.0

Pipe Throughput                               12440.0    1294793.6   1040.8

Pipe-based Context Switching                   4000.0     312592.4    781.5

Process Creation                                126.0      17515.4   1390.1

Shell Scripts (1 concurrent)                     42.4       5755.2   1357.4

Shell Scripts (8 concurrent)                      6.0        820.4   1367.3

System Call Overhead                          15000.0     694260.9    462.8

========

System Benchmarks Index Score                                        1585.0

Можно увидеть, что индекс производительности у нового ядра меньше, более чем в два раза.

# Заключение

В ходе работы мы многократно собрали ядро системы и проследили зависимость времени сборки от количества потоков сборки. Оптимальным числом потоков сборки является количество ядер в процессоре. В нашем случае это число равно 4. Так процессор сможет оптимально использовать все преимущества многопоточности.

Мы смогли без проблем запустить систему используя новое ядро.

Несмотря на то, что в качестве собираемого ядра мы выбрали linux-zen, редакцию ядра, рассчитанную на увеличение производительности, мы получили ухудшение производительности. Почему так произошло? Если внимательно прочитать вывод mkinitcpio в пункте 8.3 можно заметить предупреждение:

«Warning: No modules were added to the image. This is probably not what you want. »

Собственно, это говорит о том, что несмотря на make modules и make install\_modules, модули не добавились в ядро. Скорей всего, с помощью этих модулей и получился бы прирост к производительности.

Почему могли не установиться модули? Вероятно, было ошибочно использовать make olddefconfig для конфигурации ядра. Стоило потратить время и настроить ядро вручную через make menuconfig.

К сожалению, использовать kexec для передачи управления новому ядру не удалось в связи с особенностями аппаратной платформы.

Более того, почитав материалы о ядре, обнаружилось, что linux-zen-lqx каким-то образом заточен под Debian архитектуру. Я сомневаюсь, что это сильно повлияло на результат работы, но это стоит учесть при последующих сборах ядра.

# Список использованных источников

# Kernel/Traditional Compilation // ArchLinux URL: https://wiki.archlinux.org/title/Kernel/Traditional\_compilation (дата обращения: 24.02.2022).

# kexec // ArchLinux URL: https://wiki.archlinux.org/title/kexec (дата обращения: 24.02.2022).

# Есть ли польза от кастомных ядер // Хабр URL: https://habr.com/ru/post/131263/ (дата обращения: 24.02.2022).