САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-Механический факультет
Кафедра Системного Программирования

Пелогейко Макар Андреевич

Сравнение регуляторов состояния простоя в Android устройствах

Отчет по учебной практике

Научный руководитель:
ст. преп. Сартасов С. Ю.

Санкт-Петербург
2021

Содержание

Введение

В наше время смартфоны с каждым годом глубже внедряются в жизнь людей и их количество увеличивается [1]. Вместе с этим повышается уровень технических характеристик (например, с 2006 года в смартфонах начала появляться технология NFC [3]) и постоянно решается вопрос времени автономной работы данных устройств [4]. Смартфон не используется постоянно - есть длительные отрезки времени, когда он находится в состоянии ожидания или использует не все ресурсы. В это время для большего сохранения энергии возможно ввести весь ЦП (Central Processing Unit, CPU) или его отдельные ядра в состояние простоя (idle state). Поскольку самой широкоиспользуемой операционной системой для смартфонов является Android [2], в данной работе будет рассматриваться именно эта ОС. В Android есть CPUIdle - система управления простоем ЦП [5].
Эта система платформонезависима и состоит из трех частей:

- регуляторы (governors) выбирают, в какое состояние простоя перевести ЦП;
- драйверы (drivers) передаёт решение регуляторов в аппаратную часть смартфона;
- ядро (core) общая платформа, связывающая систему воедино;

В ОС Android есть несколько вариантов стандартных регуляторов, но и сторонними разработчиками созданы регуляторы состояния простоя. В данной работе мы сравниваем стандартные и нестандартные регуляторы между собой.

1 Цели и задачи

Цель: Сравнение актуальных регуляторов простоя для Android OS
Задачи:
1 Сделать обзор регуляторов простоя.
2 Сделать обзор на актуальные CPUIdle регуляторы.
3 Выбрать регуляторы простоя для сравнения.
4 Определить параметры, по которым будет проходить сравнение.
5 Провести сравнение.
6 Сделать выводы.

2 Обзор устройства регуляторов простоя

Для рассмотрения регуляторов простоя сначала рассмотрим, что такое состояния простоя. Современные процессоры обычно могут переходить в состояния простоя, в которых выполнение программы приостанавливается. Поскольку часть аппаратного обеспечения процессора не используется в состояниях простоя, использование этих состояний позволяет снизить мощность, потребляемую процессором, и дает возможность сэкономить энергию [5]. Эти состояния различаются по времени входа в него, времени выхода и потребляемой энергии CPU в нем.

Перечислим некоторые часто используемые состояния простоя на примере устройства Samsung Galaxy Nexus 3 с Android 4.1.1 Ice Cream Sandwich. Смартфон оснащен процессором ARM dual core 1.2 GHz Cortex-A9 [9] (состояния перечисляются в порядке увеличения глубины простоя):

- C1 (WFI) - большинство таймеров процессора отключены. Задержка выхода из этого состояния составляет 4 мкс.
- C2 ((CPUs OFF, MPU + CORE INA) - ЦП выключен, блок защиты памяти (MPU) включен для защиты критически важных данных, а ядро неактивно. Задержка выхода из этого состояния составляет 1100 мкс.
- C3 (CPUs OFF, MPU + CORE Closed Switched with Retention) - аналогично состоянию C2, но более глубокое. Задержка выхода для этого состояния это 1200 мкс.
- C4 (CPUs OFF, MPU CSWR + CORE Open Switched Retention) - аналогично состоянию C2, но еще более глубокое. Задержка выхода из этого состояния составляет 1500 мкс.

Далее опишем Android OS API для работы регуляторов состояния простоя [6]:

- struct cpuidle_governor
  {
  char name[CPUIDLE_NAME_LEN];
  struct list_head governor_list;
  unsigned int rating;
  int (*enable) (struct cpuidle_driver *drv, struct cpuidle_device *dev);
  void (*disable) (struct cpuidle_driver *drv, struct cpuidle_device *dev);
  int (*select) (struct cpuidle_driver *drv, struct cpuidle_device *dev, bool *stop_tick);
  void (*reflect) (struct cpuidle_device *dev, int index);
  }; - структура регулятора простоя.

  CPUidle регистрирует регулятор функцией cpuidle_register_governor() в качестве используемого может быть только 1 регулятор

  - int (*enable) (struct cpuidle_driver *drv, struct cpuidle_device *dev) - подготовка регулятора для обработки переданного ядра dev, в drv по мимо драйвера должен лежать список состояний, в которые можно перевести данное ядро.
  - void (*disable) (struct cpuidle_driver *drv, struct cpuidle_device *dev) - остановит обработку данного ядра (dev) и освободит всю память, которая выделена под него.
  - int (*select) (struct cpuidle_driver *drv, struct cpuidle_device *dev, bool *stop_tick) -
    * в drv находится драйвер и массив состояний;
    * в dev находится ядро, относительно которого мы сейчас принимаем выбор;
    * в bool *stop_tick - находится функция, определяющая, стоит ли остановить планировщик перед переходом CPU в выбранное состояние;

    Возвращаемое значение — индекс состояния из массива переданного в drv, в которое надо перевести данный CPU, или отрицательное число - код ошибки.
    Указатель на select в struct cpuidle_governor обязательно не равен NULL. Регулятор состояния простоя должен учитывать задержку пробуждения процессора (power management quality

of service, PM QoS) при выборе состояния простоя. Для получения текущей PM QoS задержки пробуждения регулятор простоя передает номер ядра в функцию
cpuidle_governor_latency_req(). После чего exit_latency состояния, индекс которого возвращает select, не может быть больше чем возвращаемое значение
cpuidle_governor_latency_req.

- void (*reflect) (struct cpuidle_device *dev, int index); оценивает точность выбора состояния, полученного вызовом select, на предмет использования этого состояния в дальнейшем.
- Поле name — строка, соответствующая названию регулятора простоя.
- Поле rating — рейтинг регулятора целое число. В качестве стандартного используется тот регулятор, у которого наибольший рейтинг[8].
- Поле governor_list — список всех регуляторов, который заполняется при передаче в функцию cpuidle_register_governor, описанной в файле cpuidle.h или governor.c

- Массив состояний должен быть отсортирован по “глубине” состояний - у состояния с индексом 0 минимальное значение
target_residency - самое неглубокое состояние. Данное поле содержится в
struct cpuidle_state, которая находится в
struct cpuidle_driver target_residency.
Поля в struct cpuidle_state, которые используются существующими регуляторами для вычислений, связанных с выбором состояния простоя:
  - target_residency - минимальное время с учетом времени входа в состояние, которое необходимо, чтобы сэкономить больше энергии нежели в менее глубоком состоянии
  - exit_latency - максимальное необходимое время для выхода ЦП из данного состояния и выполнения первой инструкции процессором.
- void (*enter) (struct cpuidle_device *dev, struct cpuidle_driver *drv, int index); указатель на функцию, вводящую CPU в выбранное состояние простоя. Находится в struct cpuidle_state и не может быть равен NULL.

3 Обзор регуляторов простоя

3.1 Регулятор Menu

Регулятор Menu является стандартным регулятором Android. Он пытается найти самое глубокое состояние ожидания, которое может быть введено в данных условиях. Он прогнозирует продолжительность следующего периода простоя на основе прошлой истории, а затем коррелирует наблюдаемые недавние продолжительности простоя с доступными состояниями ожидания, чтобы выбрать состояние ожидания, которое, скорее всего, будет соответствовать следующему интервалу ожидания. Регулятор Menu применяет различные корректирующие факторы на время до следующего прогнозируемого пробуждения, включая загрузку системы и количество задач, ожидающих ввода-вывода. Корректирующие факторы ограничивают влияние на производительность при переходе в состояния ожидания.

Есть несколько проблем, которые, делают работу регулятора Menu не такой точной, как хотелось бы [5]. Первое наблюдение связано с созданием шаблона истории прерываний. Регулятор Menu использует все прерывания, включая таймеры, чтобы предсказать, когда произойдет следующее. С другой стороны, он уже имеет информацию, когда произойдет следующий тик таймера, но не связывает их друг с другом. В результате может случиться так, что регулятор предсказывает пробуждение по таймеру, когда он уже должен знать, что следующее событие таймера на самом деле произойдет позже.

Другая проблема заключается в том, что регулятор использует количество процессов, ожидающих ввода-вывода, в качестве корректирующего фактора. Причиной тому стало желание снизить влияние состояний простоя на высоконагруженные системы. Переход в более глубокие состояния простоя в таких системах может иметь более заметное влияние на производительность, поэтому коррекция направляет регулятор в сторону менее глубоких состояний простоя. Количество процессов, ожидающих ввода-вывода, не влияет на доступные состояния простоя и не должно приниматься во внимание. Обнаружение шаблонов, используемое регулятором Menu, иногда учитывает значения, которые слишком велики, чтобы иметь смысл на практике. Эти значения можно опустить, и тогда для анализа потребуется меньше ресурсов. Данный регулятор часто используется в системах без тиков.

3.2 Регулятор Ladder

Регулятор Ladder тоже является стандартным регулятором Android OS. Он сначала выбирает самый неглубокий режим ожидания, а затем переходит к следующему более глубокому режиму, если наблюдаемое время ожидания достаточно велико. Считается, что это лучший выбор

при работе с системами без возможности отключения тиков планировщика и когда энергопотребление не является важным фактором.

Недостатком лестничного регулятора является то, что для выхода в режим глубокого состояния простоя может потребоваться длительное время.

3.3 Регулятор TEO

Это новый регулятор, созданный в 2019 году. Он ориентирован на события таймера (TEO) и является альтернативным регулятором CPUIdle для систем без тиков. Он следует той же базовой стратегии, что и Menu: он всегда пытается найти наиболее глубокое состояние ожидания, подходящее для данных условий. Однако он применяет другой подход к этой проблеме.

Во-первых, он не использует поправочные коэффициенты продолжительности сна, а вместо этого пытается сопоставить наблюдаемые значения продолжительности простоя с доступными состояниями ожидания и использовать эту информацию для определения состояния ожидания, которое с наибольшей вероятностью «соответствует» предстоящему интервалу простоя CPU.

Во-вторых, он не берет на себя задачи, которые выполнялись на данном CPU в прошлом и ожидают завершения некоторых операций ввода-вывода сейчас (нет гарантии, что они будут работать на том же CPU, когда они снова станут работоспособными), а код обнаружения паттернов в нем не учитывает срабатывания таймера. Он также использует для этой цели только значения продолжительности простоя, меньшие, чем текущее время, до ближайшего таймера (исключая отметку планировщика).

Как и в случае с регулятором Menu, первым шагом является получение продолжительности сна, то есть времени до ближайшего события таймера с предположением, что такт планировщика будет остановлен (это также верхняя граница времени до следующей активации CPU). Это значение затем используется для предварительного выбора состояния ожидания на основе трех метрик, поддерживаемых для каждого состояния ожидания, предоставляемого драйвером CPUIdle driver: hits, misses и early_hits.

3.4 Регулятор HaltPoll

Данный регулятор тоже является новым (изобретен в 2019 году), но подробно рассматриваться не будет из-за того, что он предназначен для работы на виртуальной машине и ограничений тестового стенда. Драйвер cpuidle_haltpoll с регулятором haltpoll позволяет гостевым виртуальным процессорам опрашивать в течение указанного периода времени перед остановкой.

Основная логика следующая: глобальное значение, guest_halt_poll_ns, настраивается пользователем — максимальное количество время опроса. Это значение фиксировано. У каждого виртуального процессора есть настраиваемый guest_halt_poll_ns ("per-cpu guest_halt_poll_ns"), который регулируется алгоритмом в ответ на события в системе. Таким образом, haltpoll управляет vCPU, чтобы избежать частого сна и пробуждений.

4 Ход работы

4.1 Подготовка тестового стенда

В качестве тестового стенда был взят смартфон Samsung Galaxy s7 SM-G930FD. Сначала была поставлена прошивка Android 8.0 - SM-G930FXXU2ERD5 BTU - прошивка от производителя. Потом через Odin (программа для прошивки смартфонов) установлена lineage-17.1-20210409 (android 10), после чего взято open source ядро linage 17.1 на основе kernaluniversall с внесением необходимымых изменений[1] и для создания запускаемого zip используется проект lazyflasher ветки no-verity-opt-encrypt[2].

Регулятор teo был создан в linux 5.1, в то время как Android базируется на linux 3. Поэтому для обеспечения работы регулятора пришлось изменить функции из регулятора на аналогичные для linux 3 и по возможности добавить функции из linux 5.1 в тестируемое ядро. Когда регулятор добавлен было создано 3 Image-файла - скомпилированные проекты Android ядер с регуляторами teo, menu, ladder путем повышения приоритета соответственного регулятора. Для сбора данных о энергопотреблении написан скрипт get_battery_stats[3], который должен находиться в памяти устройства в папке Downloads/IDLE_TEST.

4.2 Сценарии тестирования

Для созданий тестов были выявлены частые сценарии использования смартфона в наше время. Из них приоритет отдавался сценариям, не требующим подключения к сети Интернет ввиду снижения неконтролируемых действий смартфона. Таким образом, получились следующие сценарии тестирования:

1. Заметки;

[1] https://github.com/makar-pelogeiko/android_kernel_samsung_universal8890/tree/lineage-17.1-teo

[2] https://github.com/jcadduono/lazyflasher/tree/no-verity-opt-encrypt

[3] https://github.com/makar-pelogeiko/Android-idle-state-course-work/tree/main/Galaxy_s7_onBoard/Test_Android_IDLE

2. Чтение;
3. Видео;
4. Игра;
5. YouTube;

Каждому сценарию теста соответствует фаил .bat, который необходимо запускать при подключенном смартфоне к компьютеру и с включенным adb, который уже имеет доступ к смартфону.
Общая подготовка к тестированию проходила так:
на смартфоне устанавливался режим блокировки "провести по экрану", выключался wifi, bluetooth и энергосбережение, включался режим полета.

1. Сценарий Заметки:
   Подготовка сценария теста:
   - Установить приложение Заметки (Turist).
   - Добавить ярлык заметок в левый верхний угол рабочего стола.
   - Запустить приложение заметок, отключить автосохранение, создать 1 текстовую заметку.
   - Переключить язык на английский.
   - Закрыть приложение заметок, выйти на рабочий стол.
   - Заблокировать смартфон и подключить к ПК в режиме отладки.

   Описание теста:
   - Сброс данных о энергопотреблении.
   - Разблокирование смартфона.
   - Включение приложения заметки.
   - Создание новой заметки (текстового поля).
   - Открытие окна печати.
   - Печать текста 2 способами в цикле и сохранение текста.
   - Выход из приложения.
   - Сбор данных о энергопотреблении.
   - Блокирование смартфона.
2. Сценарий Чтение:
   Подготовка сценария теста:

- Добавить ярлык pdf файла в левый верхний угол (на 2 позиции правее заметок из теста 1) рабочего стола.
- Проверить запуск drive pdf viewer 1 кликом.
- Выйти на рабочий стол.
- Заблокировать смартфон и подключить к ПК в режиме отладки.

Описание теста:

- Сброс данных о энергопотреблении.
- Разблокирование смартфона.
- Открытие pdf файла.
- Имитация чтения (протягивание текста).
- Выход из приложения.
- Сбор данных о энергопотреблении.
- Блокирование смартфона.

3. Сценарий Видео:
   Подготовка сценария теста:

   - Добавить ярлык файла mp4 в левый верхний угол (на 2 позиции ниже заметок из теста 1) рабочего стола.
   - Проверить запуск видеоплеера 1 кликом.
   - Выйти на рабочий стол.
   - Заблокировать смартфон и подключить к ПК в режиме отладки.

   Описание теста:

   - Сброс данных о энергопотреблении.
   - Разблокирование смартфона.
   - Открытие видеофайла.
   - Имитация просмотра (бездействие, постановка на паузу, снятие с паузы).
   - Выход из приложения.
   - Сбор данных о энергопотреблении.
   - Блокирование смартфона.

4. Сценарий Игра:
   Подготовка сценария теста:

   - Установить приложение HillClimb Racing.

- Добавить ярлык приложения в левый верхний угол рабочего стола (ниже приложения из теста 2).
- Запустить приложение, пройти 2 цикла игры.
- Закрыть приложение, выйти на рабочий стол.
- Заблокировать смартфон и подключить к ПК в режиме отладки.

Описание теста:

- Сброс данных о энергопотреблении.
- Разблокирование смартфона.
- Включение приложения HillClimb Racing.
- Имитация игры (запуск новых циклов игры, процесс игры).
- Выход из приложения.
- Сбор данных о энергопотреблении.
- Блокирование смартфона.

5. сценарий YouTube:
   Подготовка сценария теста:

   - установить приложение YouTube.
   - Добавить ярлык файла YouTube-test.html на в левый нижний угол рабочего стола.
   - Открыть файл, кликнуть по ссылке.
   - Закрыть все приложения, выйти на рабочий стол.
   - Заблокировать смартфон и подключить к ПК в режиме отладки.

   Описание теста:

   - Сброс данных о энергопотреблении.
   - Разблокирование смартфона.
   - Включение приложения YouTube по средством открытия файла YouTube-test.html и перехода по ссылке.
   - Имитация просмотра (проверка времени видео).
   - Выход из приложения.
   - Сбор данных о энергопотреблении.
   - Блокирование смартфона.

4.3 Сбор данных

Каждый сценарий тестирование исполнялся 25 раз для каждого регулятора. Для автоматизации представления был написан парсер на C#[4]. Данные тестирования загружены в таблицу[5]. В случае с имеющимся тестовым стендом для создания ярлыков на главном экране было использовано приложение File Manager (для его закрытия добавлено дополнительное нажатие в тест).

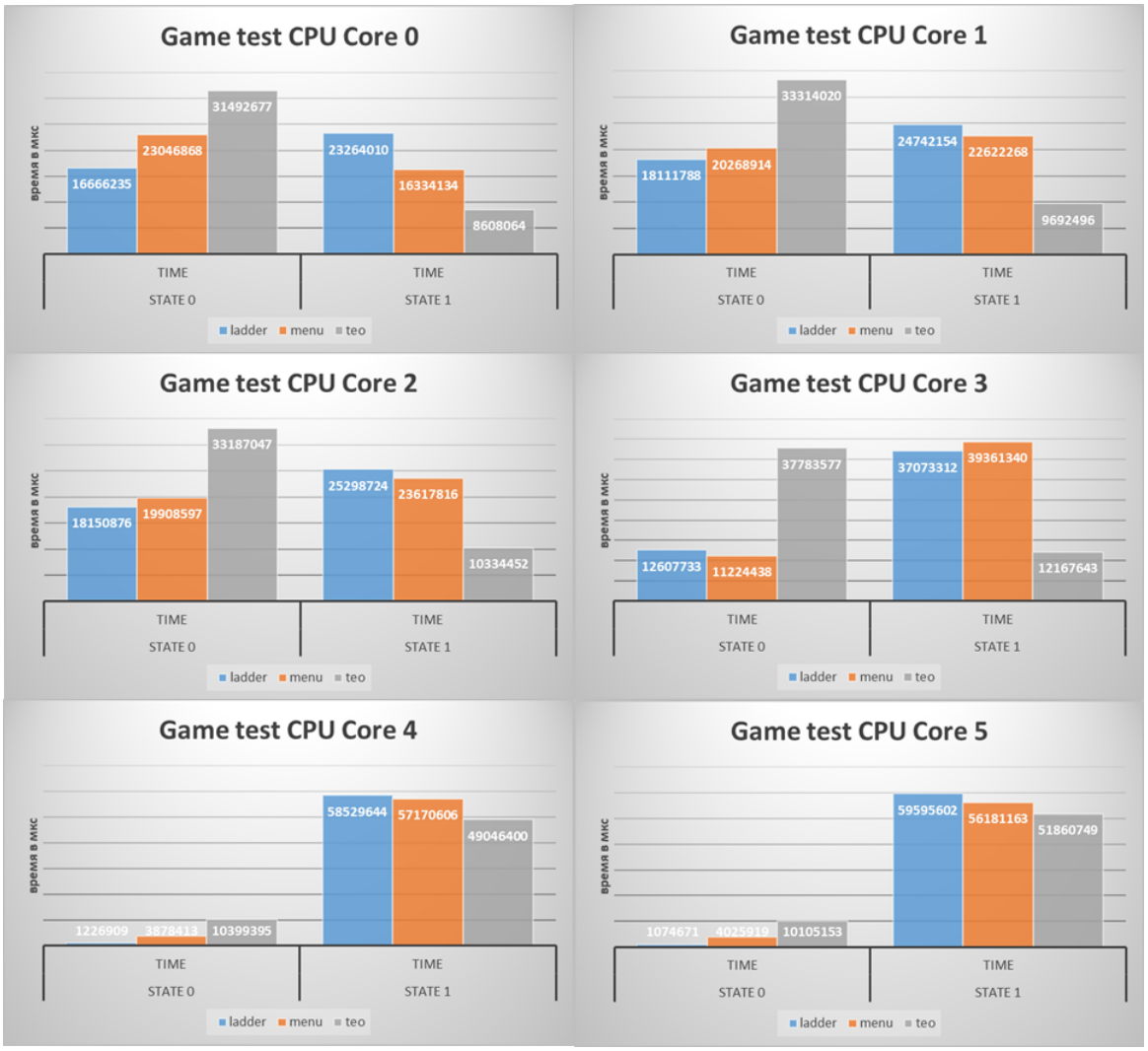
4.4 Анализ результатов

Поскольку в данном тестовом стенде имеются 2 состояния простоя: state 0 и state 1. И имеется возможность для отдельных ядер определять, сколько времений проведено в каждом состоянии простоя в микросекундах. С точки зрения энергоэффективности лучше тот регулятор, у которого суммарно ядра дольше находились в state 1.

Результат сравнения регуляторов в сценарии Игра:

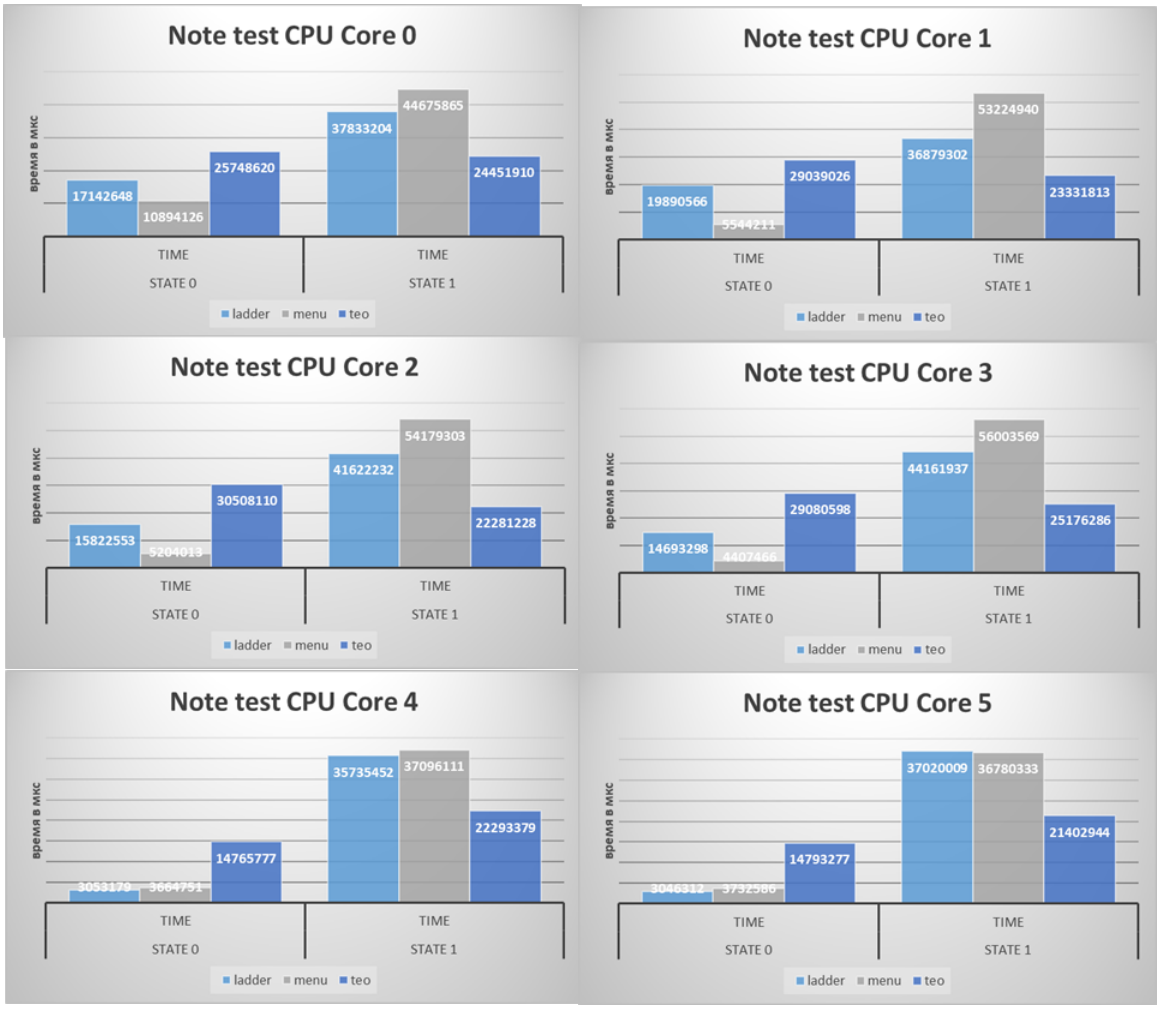
[4] `https://github.com/makar-pelogeiko/Android-idle-state-course-work/tree/main/parserDataTest`

[5] `https://github.com/makar-pelogeiko/Android-idle-state-course-work/tree/main/Results`

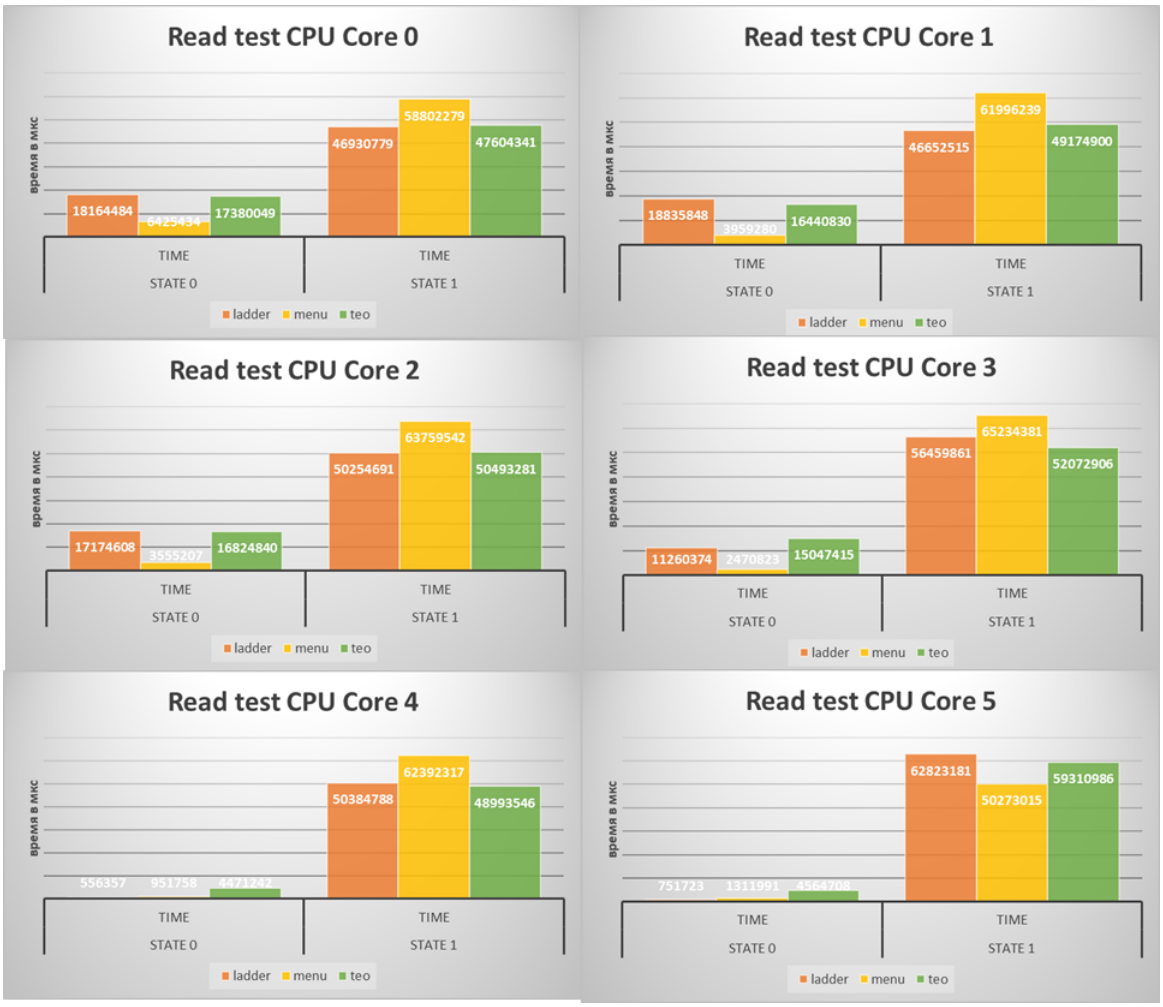


Не считая результатов 3 ядра, лучшую энергоэффективность показал регулятор ladder.

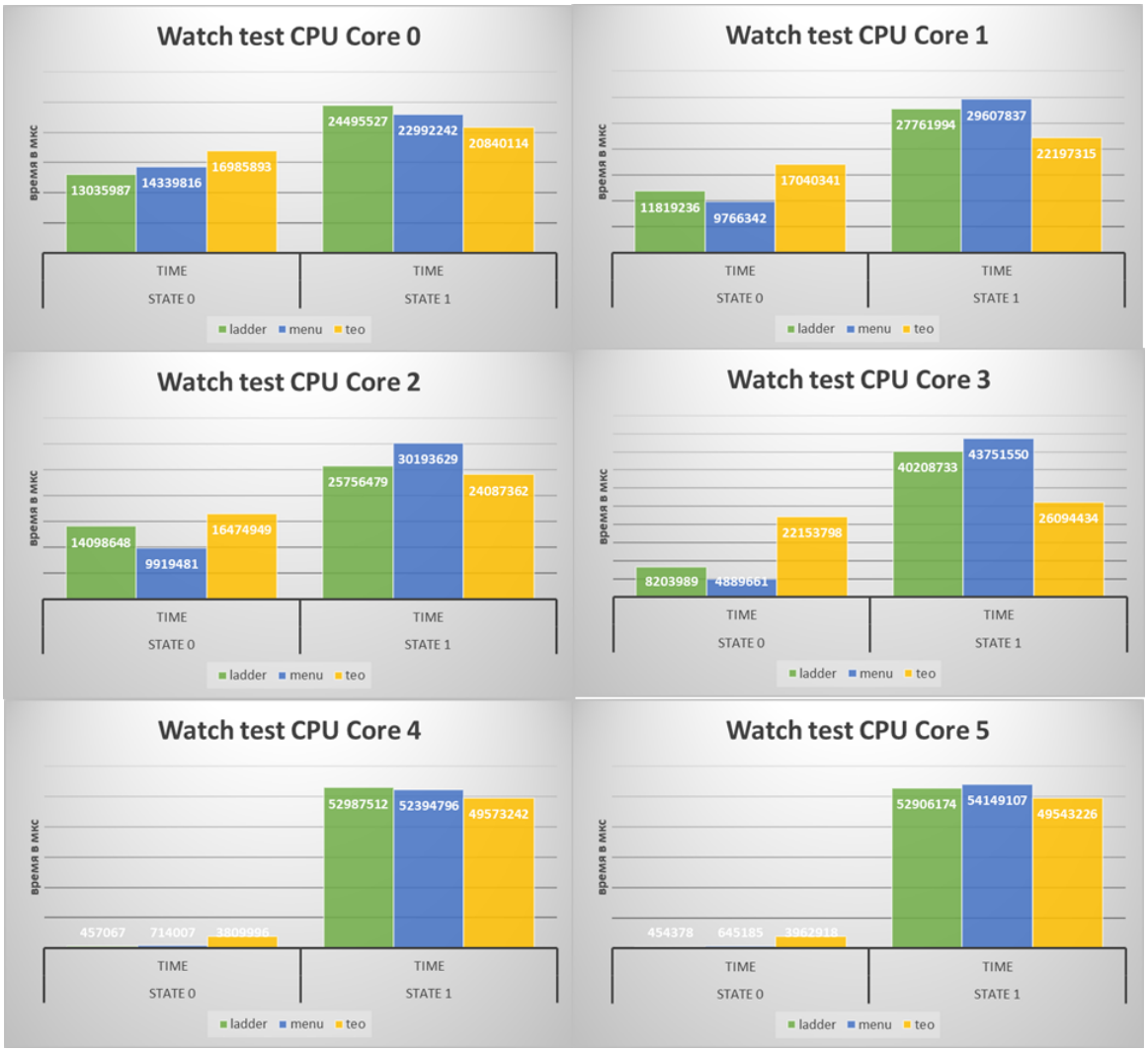
Результат сравнения регуляторов в сценарии Заметки:



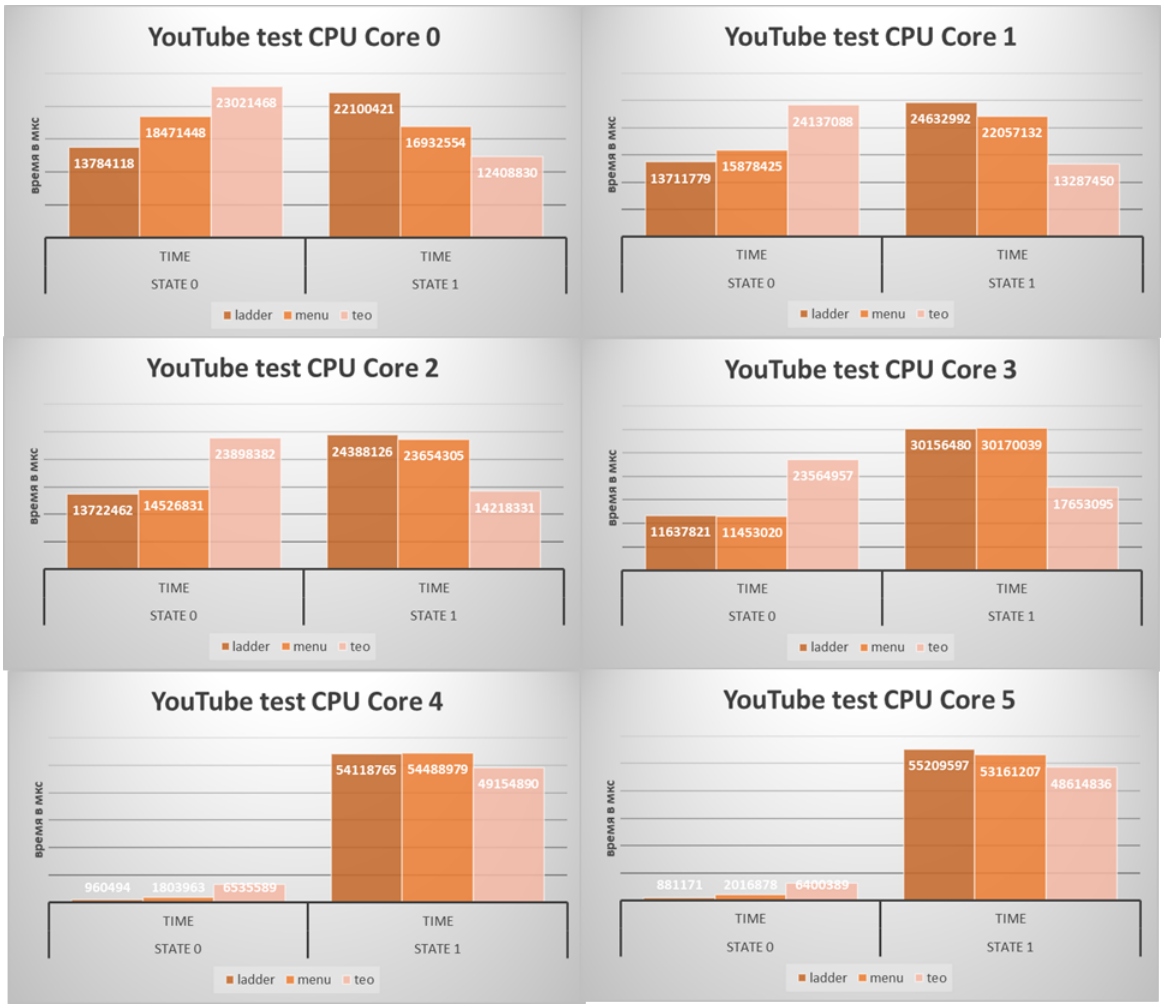
Результат сравнения регуляторов в сценарии Чтение:



Результат сравнения регуляторов в сценарии Видео:



Результат сравнения регуляторов в сценарии YouTube:



Не считая результатов 3 ядра, в котором результаты menu и ladder практически одинаковы, лучшую энергоэффективность показал регулятор ladder.

5 Вывод

Исходя из полученных результатов, можно сказать следующее: регуляторы ladder и menu показали близкие друг к другу результаты. В основном энергоэффективность регулятора menu выше, чем у ladder. Ladder показал себя лучшим в сценарии игра и YouTube. В это же время с регулятором ladder появляется ощутимое время отклика на действия пользователя. Регулятор teo с точки зрения энергоэффективности показал себя наихудшим образом во всех сценариях, но смартфон вместе с регулятором teo имел лучший отклик на действия пользователя.

Список литературы

[1] Number of mobile phone users in the U.S. from 2012 to 2020: https://www.statista.com/statistics/222306/forecast-of-smartphone-users-in-the-us/

[2] Worldwide Smartphone Shipment OS Market Share: https://www.idc.com/promo/smartphone-market-share/os

[3] First smartphone with NFC: https://compress.ru/article.aspx?id=23661

[4] Smartphone battery life over the years: https://www.androidauthority.com/smartphone-battery-capacity-887305/

[5] CPUIdle Time Management 2018: https://www.kernel.org/doc/html/latest/admin-guide/pm/cpuidle.html

[6] CPUIdle Time Management 2019: https://www.kernel.org/doc/html/latest/driver-api/pm/cpuidle.html

[7] Improving idle behavior in tickless systems: https://lwn.net/Articles/775618/

[8] The cpuidle subsystem: https://lwn.net/Articles/384146/

[9] G. Metri, A. Agrawal, R. Peri, M. Brockmeyer and Weisong Shi, "A simplistic way for power profiling of mobile devices" 2012: https://ieeexplore.ieee.org/abstract/document/6471020