Arch Linux Optimization Guide (RU)

Выпуск 2024.07.21

Arch Linux Optimization Guide (RU)

апр. 17, 2025

Содержание

Это помощник по настройке вашей системы Arch Linux с целью получить максимальную производительность и настроить систему для комфортной игры. Проект не претендует на замену Arch Wiki, он лишь является практическим руководством, написанным на основе личного опыта его авторов.

Данный репозиторий является зеркалом, а ныне и основным хранилищем одноименного руководства из Google Docs.

Привет, неизвестный мне чувак из интернета, раз ты тут, то возможно жаждешь настроить свою систему на максимальный выхлоп, но прежде чем начать - знай: **Все манипуляции на вашей совести и авторы не несут никакой ответственности**, но если вам нужна помощь или что-то не понятно - создайте тему на Codeberg

На текущий момент руководство находится в активной переработке, поэтому если вы нашли ошибку или просто хотите внести свой вклад в проект - отправьте на рассмотрение Pull Request в наш Codeberg репозиторий. Мы будем признательны за ваше участие.

ГЛАВА 1

Содержание

1.1 Настройка pacman

1.1.1 Основные команды для управления пакетами

Ниже в таблице перечислены все основные команды pacman и другие, которые часто будут встречаться на протяжении всего руководства. Их лучше запомнить.

Таблица 1: Список часто используемых команд

Команда	Описание
`sudo pacman -S название_пакета`	Устанавливает *пакет* официальных из репозиториев.
`sudo pacman -Syu`	Выполнить полное обновление системы вместе с синхронизацией баз данных репозиториев.
`sudo pacman -R название_пакета`	Удаляет названный пакет из системы
`sudo pacman -Rsn название_пакета`	Удаляет названный пакет из системы, а также все его зависимости, если они не требуются другими установленными пакетами.
`sudo pacman -Scc`	Выполняет полную очистку кэша пакетов. Подробнее см. *Профилактика диска*.
`sudo pacman -Rscn $(pacman -Qtdq)`	Удаляет из системы все так называемые пакеты "сироты", то есть такие, которые не были явно установлены и не являются при этом ничьими зависимостями.
`git clone`	Команда для клонирования git репозитория. Активно используется при установке AUR пакетов нижеуказанным способом.
`makepkg -sric`	Команда для сборки пакета из соответствующего файла-сценария PKGBUILD. Ключ *-i* запросит установку пакета после его сборки, *-s* запросит установку сборочных зависимостей, *-r* выполнит их автоматическое удаление после окончания сборки пакета, *-c* выполнит удаление временных файлов.

1.1.1.1 Пару слов об AUR помощниках

Далее в руководстве все пакеты из AUR (Arch Linux User Repository) будут устанавливаться и собираться, если так можно выразиться, "дедовским" способом, т.е. через стандартные утилиты git и makepkg, без применения так называемых "AUR Помощников". Это сделано по причине их быстрой сменяемости, и тот помощник который был актуален раньше, может стать устаревшим и никому не нужным. Для примера, так было с AUR-помощником yaourt.

Кроме того, согласно Arch Wiki, AUR-помощники официально не поддерживаются дистрибутивом. А "старый" метод, через обычное клонирование git репозитория из AUR командой `git clone` и сборка пакета через makepkg, будет работать всегда. Тем не менее, обращаем ваше внимание, что возможность установки пакетов через AUR помощник возможна, и вы можете её использовать для всех AUR пакетов, о которых пойдет речь далее. Подробнее об этом можно почитать здесь.

1.1.2 Обновление ключей Arch Linux

Обновление ключей необходимо во избежание дальнейших проблем с установкой пакетов:

```
sudo pacman-key --init                # Инициализация
sudo pacman-key --populate archlinux # Получить ключи из репозитория
sudo pacman-key --refresh-keys       # Проверить текущие ключи на актуальность
sudo pacman -Sy archlinux-keyring    # Обновить пакет archlinux-keyring
```

Данная операция может занять продолжительное время. Для дальнейшего их автоматического обновления нужно включить службу-таймер, которая оптимизирует процесс при помощи команды `archlinux-keyring-wkd-sync`:

```
sudo systemctl enable --now archlinux-keyring-wkd-sync.timer
```

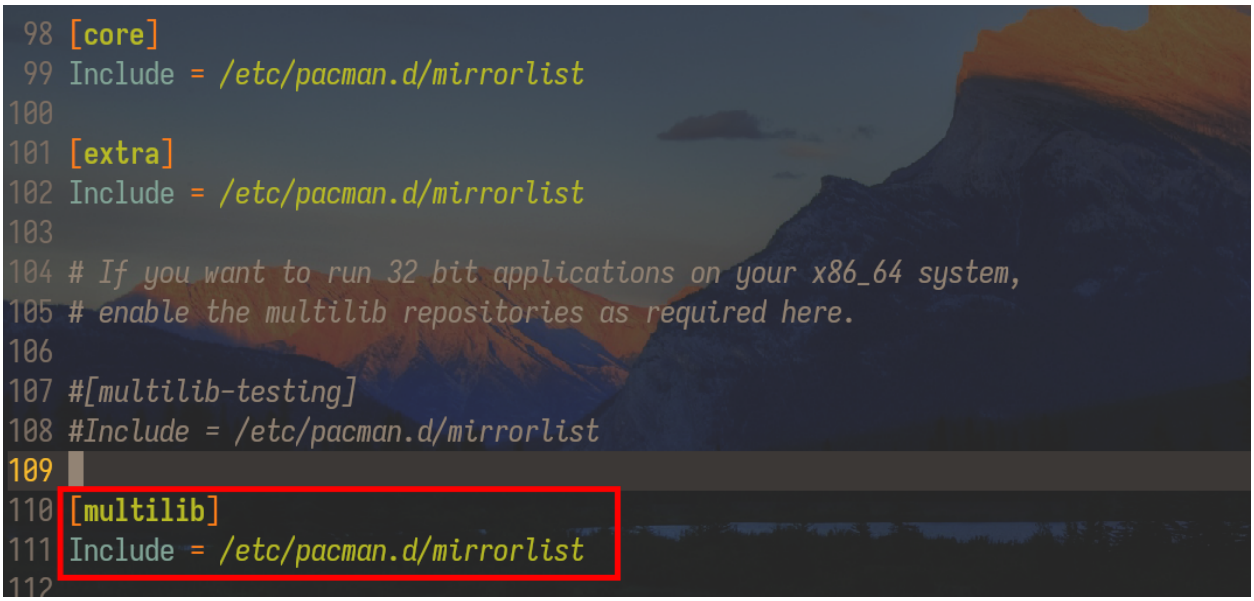
Таким образом ключи будут обновляться раз в неделю.

1.1.3 Включение 32-битного репозитория

Убедимся, что конфигурация пакетного менеджера Pacman настроена для получения доступа к 32-битным зависимостям, нужным в частности для установки Wine и Steam.

Для этого раскомментируем так называемый *multilib* репозиторий:

```
sudo nano /etc/pacman.conf           # Раскоментируйте последние две строчки как на␣
↪скриншоте
```




```
sudo pacman -Suy                    # Обновление репозиториев и всех программ␣
↪(пакетов)
```

Примечание

Если вы не используете Steam или Wine, то можете пропустить данный шаг. Также стоит отметить возможность работы Wine без установки 32-битных зависимостей, подробнее см. *Установка wine-pure*.

1.1.4 Ускорение обновления системы

Утилита Reflector отсортирует доступные репозитории по скорости:

```
sudo pacman -S reflector rsync
```

Если вы из Европейской части России, то советуем всегда использовать зеркала Германии, так как их больше всего и они имеют оптимальную свежесть/скорость:

```
sudo reflector --verbose --country 'Germany' -l 25 --sort rate --save /etc/pacman.d/
↪mirrorlist
```

Если вы проживаете не на территории Европейской части РФ или в иной стране, то просто измените *Germany* на *Russia* или ваше государство.

Можно также вручную отредактировать список зеркал, добавив туда зеркала из постоянно обновляющегося перечня на сайте Arch Linux (https://archlinux.org/mirrorlist/):

```
sudo nano /etc/pacman.d/mirrorlist # Рекомендуем прописывать зеркала от Яндекса
```

1.1.5 Параллельная загрузка пакетов

Начиная с шестой версии pacman поддерживает параллельную загрузку пакетов. Чтобы её включить отредактируйте `/etc/pacman.conf`:

Список 1: `sudo nano /etc/pacman.conf` # Раскомментируйте строчку внутри файла

```
# Где 4 - количество пакетов для одновременной загрузки
ParallelDownloads = 4
```

1.1.6 Отключение таймаутов при загрузке пакетов

Если вы имеете плохое качество соединения или слабый уровень сигнала, то при загрузке пакетов при помощи pacman вы могли сталкиваться с ошибкой превышания лимитов ожидания (таймаутов). Чтобы этого избежать нужно добавить параметр `DisableDownloadTimeout` в `pacman.conf` как мы это уже делали ранее с `ParallelDownloads`:

Список 2: `sudo nano /etc/pacman.conf` # Добавьте строку в секцию [options]

```
DisableDownloadTimeout
```

1.2 Выбор ПО

Говоря об оптимизации системы невозможно не затронуть такую тему, как выбор используемых программ. Несмотря на то, что это относится к субъективным предпочтениям каждого пользователя, это оказывает прямое влияние на производительность системы, так как все программы отличаются друг от друга такими объективными показателями как потребление памяти, использование процессора, диска и других ресурсов системы. Иными словами, от того насколько оптимизированы ваши программы зависит общая производительность всей системы, поэтому в данном разделе автор предложит ряд рекомендаций о том, какое ПО следует выбрать для тех или иных задач с учётом их нетребовательности к ресурсам вашего ПК.

1.2.1 Обязательные к установке пакеты

Эта группа пакетов **обязательна** к установке. Она содержит ряд полезных инструментов без которых вы не сможете установить множество пакетов из AUR, включая все те, о которых пойдет речь далее в руководстве. И так как в начале руководства мы условились пользоваться "дедовским" методом установки AUR пакетов, дополнительно установим `git` для скачивания PKGBUILD и текстовый редактор `nano` для редактирования конфигурационных файлов (вы так же в праве использовать любой другой):

```
sudo pacman -S base-devel git nano
```

1.2.2 Wayland vs X11

Часто задаваемый вопрос, это на базе какого графического протокола следует выбирать рабочее окружение - Wayland или X11? Обычно можно услышать лишь различные субъективные оценки и мнения, но автор попытается провести объективное их сравнение, после чего читатель сможет сам решить, что лучше выбрать.

Прежде всего следует понимать, что так называемый графический протокол X11 берёт свое начало 40 лет назад, в эпоху когда не существовало современных GPU и мультимониторных конфигураций. Конечно, протокол за всё это время получал достаточно много обновлений в рамках развития к нашему времени уже единственной (кроме Xwayland) реализации - Xorg сервера, в частности добавлялись новые расширения, которые в том числе таки позволили работать с несколькими мониторами (расширение xinerama и xrandr) и улучшить производительность (частичные обновления экрана, расширение X Damage). Однако фундаментальные ограничения самого протокола из-за требований к сохранению обратной совместимости не позволяют говорить о полном решении данных проблем.

Так как само понятие монитора в X11 как самодостаточной единицы отсутствует, то до сих пор вывод со всех дисплеев объединяется в некий единый логический "экран" (Screen), и все обновления содержимого окон воспроизводится только с учётом обновления всего такого экрана, то есть фактически в X11 нельзя обновить содержимое одного монитора отдельно от содержимого другого, что порождает проблему разницы частоты обновления. Если один монитор настроен на обновление с частотой меньшей, чем у другого, то так как в парадигме протокола обновляться может сразу только весь абстрактный экран, второй монитор так же вынужден перерисовывать свое содержимое с частотой первого. У этой проблемы есть решения, такие как выполнение обновления логического экрана с самой большой частотой из доступных у всех мониторов, но она не является единственной. Ещё одно вытекающее следствие из этого - невозможность правильного осуществления масштабирования выводимого изображения на экран, да и в целом невозможность правильной его обработки с учетом особенностей каждого отдельно взятого монитора, сюда же относится поддержка VRR и HDR.

Другая проблема протокола X11 состоит в ограниченности способов отрисовки содержимого на этот самый экран. Приложение не может рисовать содержимое своего окна самостоятельно, оно обязательно должно делать это опосредованно и в соответствии с форматом необходимым для использования API библиотеки libx11 или libxcb, которые изначально были разработаны без учёта многих цветовых форматов и необходимости в 3D-ускорении средствами GPU. Другими словами, исторически у приложений не было возможности выполнять прямую отрисовку содержимого окна через подсистему DRM в ядре, все изменения и обновления могли быть совершены только путем общения с X сервером с принудительными преобразованиями на уровне приложения всех операций в двумерное пространство, что стало большой проблемой, которая делала невозможной правильную отрисовку 3D графики или делала её очень низкопроизводительной. Впоследствии эта проблема была решена введением так называемого DRI - инфаструктуры прямого рендеринга, однако решение это по сути прикручено сбоку от самого графического протокола, то есть не является его расширением, а просто попыткой обойти указанные ограничения в рамках графического стэка развиваемого Mesa.

Ещё одна проблема протокола X11 - безопасность. Любое приложение может знать о содержимом не только своего окна, но и других окон, а также перехватывать по сути все события ввода, которые даже не были адресованы конкретно данному окну. Таким образом для написания любого зловредного ПО даже не нужно искать никакие уязвимости, протокол X11 сам вполне спокойно позволяет беспрепятственно получать любую информацию. Отчасти этому способствует также "сетевая" архитектура протокола. Написание простейшего "кейлоггера" для X11 может занимать около 150 строк кода и не требует вообще никаких особых прав доступа. Если вы думаете, что и на это появилось какое-то решение, то, увы нет. Так называемое расширение безопасности X11 не получило широкого и повсеместного распространения, многие приложения не работают правильно с его использованием, так как изначально подразумевают себя привилегированными для осуществления многих операций, что в итоге приводит к их неработоспособности. Наиболее рабочими можно считать только решения с запуском приложений в отдельном, изолированном X сервере, что конечно нельзя назвать хорошим компромиссом.

С учётом всех вышеуказанных фундаментальных проблем протокола X11, было решено развивать новый графический протокол - Wayland, который призван был решить проблемы предшественника, и надо сказать вполне успешно. Во-первых, в Wayland введено понятие "поверхностей" (`wl_surface`) - не следует приравнивать поверхность к понятию окна, поверхность представляет собой по сути "холст", в который приложение может выполнять отрисовку по сути абсолютно любыми средствами, включая различные API, такие как OpenGL, Vulkan и другие. Здесь нет никаких ограничений, главное, чтобы в конечном счете появился ассоциированный с поверхностью буфер, в котором находился набор пикселей определенного цветого формата, которые затем будут отображены композитором.

Во-вторых, отображение поверхностей осуществляется через так называемые `wl_output` объекты, которые описывают некоторую область для вывода пикселей соответствующих поверхностей на экран, как правило все такие объекты напрямую связаны с конкретным монитором у вас в системе, и при этом управление над такими объектами осуществляется полностью независимо, то есть без оглядки на другие такие объекты.

Наконец, в третьих, приложения ничего не могут знать о содержимом других окон (поверхностей), кроме тех, которые управляются непосредственно ими же, они также ничего не знают о событиях ввода, которые адресованы не для них. Такие жесткие рамки хоть и создают определенные сложности для разработчиков приложений, так как теперь для взаимодействия с другими окнами приходится использовать средства IPC, но это позволяет

гарантировать безопасность.

Несмотря на все очевидные преимущества Wayland и работе над ошибками X11, из-за сравнительно небольшого возраста он страдает от проблем с "организационной" фрагментацией. У протокола нет единой рабочей реализации (хотя есть эталонная - Weston), так как в его парадигме любой Wayland композитор представляет собой по сути одновременно графический сервер, оконный менеджер и композитор в узком смысле, для объединения содержимого всех поверхностей в единое изображение на вашем мониторе с возможным добавлением вертикальной синхронизации и некоторых эффектов, таких как тени, прозрачность и т. д. Учитывая столь широкие полномочия, которые в рамках X11 как правило реализовывались отдельными сущностями, вполне логично, что большинство рабочих окружений имеют свои собственные Wayland композиторы, которые отвечают их собственным убеждениям о том, как должна выглядить организация управления окнами в их окружении. На текущий момент можно выделить четыре большие группы всех Wayland композиторов:

- mutter - композитор используемый в GNOME, применяется также в Budgie.
- KWin - композитор используемый в KDE Plasma.
- Композиторы на базе библиотеки wlroots (sway, river, labwc и другие).
- Композиторы на базе библиотеки smithray (Niri, COSMIC).

Кроме отличий в организации окнами, Wayland композиторы также отличаются составом поддерживаемых расширений протокола Wayland, которые тоже называются протоколами. Данные различия вносят небольшую неразбериху в возможностях окружений на базе Wayland при их сравнении с графическими сессиями на базе X11, так как часть дополнительных протоколов могут быть реализованы в одном композиторе, но не в другом, а кроме того любой композитор также может иметь свой собственный набор протоколов, расширяющий его возможности. В контексте данного руководство отдельно хотелось бы остановиться на данных дополнительных протоколах Wayland:

- `tearing-control` - позволяет композиторам контролировать, для каких окон разрешено прямое асинхронное отображение кадров, что сопряжено с появлением так называемого "тиринга", то есть визуального разрыва между кадрами. Данный протокол полезен в первую очередь для полноэкранных видеоигр, предотвращая высокие задержки ввода вызванные синхронизацией кадров на стороне композитора (хочу отметить, что это не обязательно должна быть именно вертикальная синхронизация, но и в целом любая их обработка вроде простейших FIFO очередей). На текущий момент реализован во всех композиторах, кроме GNOME и композиторах на базе библиотеки Smithray.
- `presentation-time` - протокол, позволяющий приложениям, использующим Wayland, указывать определенную временную "подсказку" для отображения кадров с привязкой к некоторому событию или временному интервалу. Используется в основном в видеоплеерах для синхронизации потоков видео с аудиодорожкой. Поддерживается во всех мейнстримных композиторах Wayland. Рекомендуется использовать видеоплееры, которые полагаются на использование данного протокола.
- `fifo` - очень молодой протокол, который позволяет приложениям, использующим простую FIFO очередь для рендеринга кадров, говорить композитору о том, чтобы их отображение выполнялось сразу же с привязкой к частоте обновления монитора, а не через ожидание некоторого обратного вызова со стороны самого приложения. Пока ещё не реализован ни в одном Wayland композиторе, но уже является жёстким требованием для нативных игр, использующих библиотеку SDL3.

Использование Wayland композиторов, которые реализуют все или хотя бы часть из указанных протоколов, желательно в случае если вы активно играете в игры и задержка ввода для вас не пустой звук. В остальном же выбор композитора является субъективным делом.

Надеюсь, что эта небольшая (а может и большая?) заметка помогла вам определиться в вопросе, который будоражит тысячи пользователей различных форумов по Linux. Так или иначе, но вектор развития большинства рабочих окружений сейчас направлен в сторону Wayland, а сервер Xorg хоть и по прежнему остается рабочей лошадкой, постепенно отходит на второй план и находится в состоянии полуактивного сопровождения.

1.2.2.1 Пакеты для работы с архивами

В Linux есть поддержка целого зоопарка различных архивов и алгоритмов их сжатия, но чтобы все они работали правильно, необходима установка дополнительных пакетов:

```
# zip, rar, ace, rzip/lcma/lzo, iso
sudo pacman -S lrzip unrar unzip unace p7zip squashfs-tools
```

Но они предоставляют только интерфейс командной строки для работы с архивами, потому стоит так же поставить графическую обертку с минимальным набором зависимостей:

```
sudo pacman -S file-roller
```

1.2.2.2 Набор прикладного ПО

Далее мы установим набор джентельмена в виде браузера (chromium), плеера (VLC) и торрент-клиента (qbittorrent):

```
sudo pacman -S qbittorrent chromium vlc
```

Банально, но всё же.

Вдобавок можно отметить легковесный файловый менеджер PCManFM:

```
sudo pacman -S pcmanfm-gtk3 gvfs gvfs-mtp
```

Предупреждение

Пакеты начинающиеся с `gvfs` нужны для автомонтирования различных устройств (например Android смартфонов) и интеграции с различными сетевыми хранилищами (Google Disk/SAMBA и т.д.)

Итак, мы установили набор джентльмена и парочку программ, что понадобятся нам в дальнейшем. Но если вас не устраивает тот или иной компонент, вы всегда можете найти любой нужный вам пакет по адресу https://www.archlinux.org/packages/. Если вы не смогли найти нужную вам программу в официальных репозиториях, вы всегда можете найти всё что душе угодно в AUR (по адресу https://aur.archlinux.org/packages/).

1.2.2.3 Установка Steam

Если в предыдущем разделе вы активировали в настройках pacman Multilib репозиторий, то из него можно установить официальный клиент Steam. Однако, здесь стоит упомянуть о сразу нескольких доступных версиях.

Существует просто steam - он содержит в себе клиент Steam с собственными копиями библиотек к нему.

```
sudo pacman -S steam ttf-liberation
```

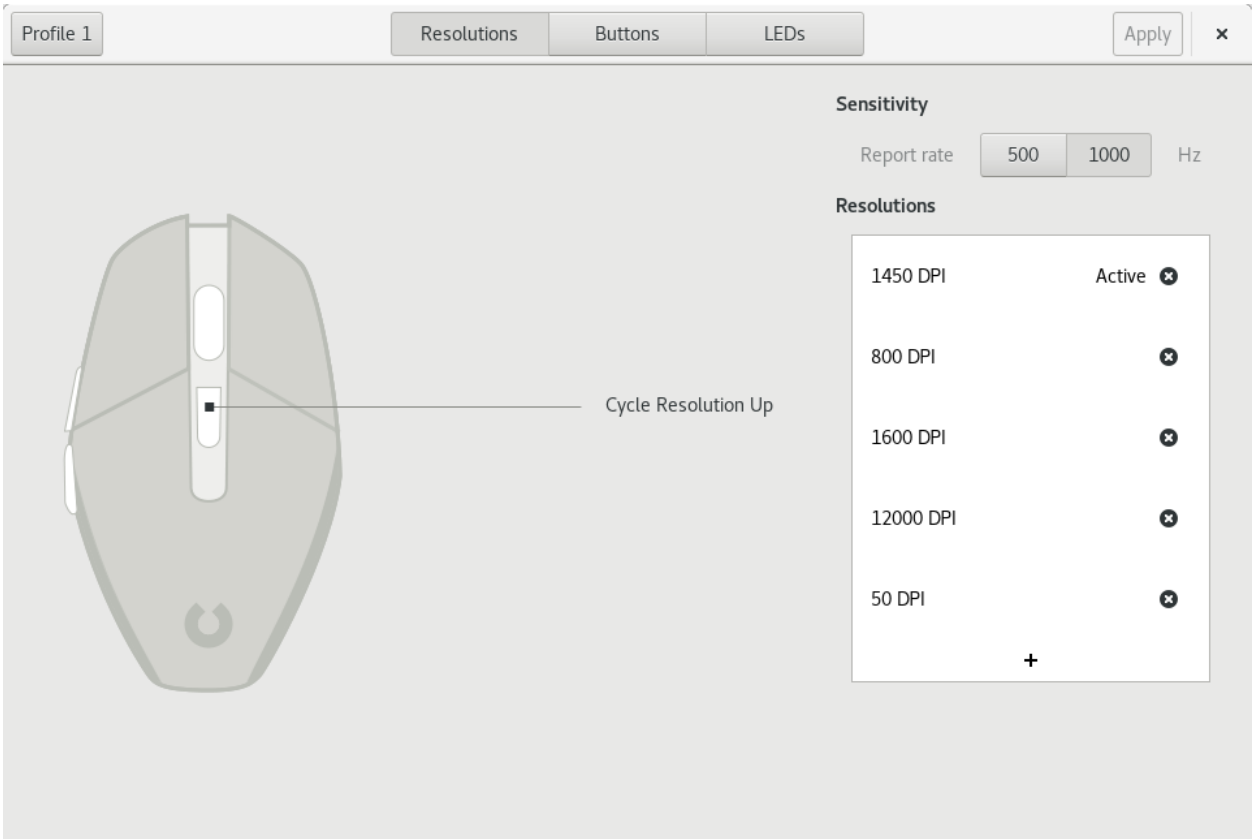
А есть steam-native-runtime, который использует локально установленные в систему библиотеки. Разница в том, что последний будет использовать всегда более свежие версии библиотек, из-за чего опыт работы клиента и нативных игр Linux может как улучшиться, так и наоборот сломать какой-либо функционал. Вдобавок, steam-native-runtime требует большее количество 32-битных зависимостей.

```
sudo pacman -S steam-native-runtime ttf-liberation
```

Автор рекомендует пользоваться обычной версией, хотя и при использовании steam-native-runtime каких-либо серьёзных проблем замечено не было.

1.2.2.4 Piper

Позволяет выполнить более тонкую настройку вашей мышки, в том числе переназначить DPI, настроить подсветку и собственные действия на дополнительные кнопки.



Установка

```
sudo pacman -S piper
```

> **Внимание**
>
> Поддерживаются только некоторые из моделей мышек от Logitech/Razer/Steelseries. Полный список поддерживаемых устройств вы можете найти по ссылке:
>
> https://github.com/libratbag/libratbag/wiki/Devices

1.2.2.5 Установка актуальных драйверов для видеокарты

В установке драйверов для Linux-систем нет ничего сложного, главное просто учитывать, что от свежести ядра и версии драйвера, будет зависеть получите ли вы чёрный экран смерти или нет (Шутка).

И да, **устанавливайте драйвера ТОЛЬКО через пакетный менеджер вашего дистрибутива!**

Забудьте про скачивание драйвера с сайта NVIDIA/AMD, это поможет вам избежать кучу проблем в дальнейшем.

1.2.2.5.1 NVIDIA

Рекомендуется использовать модули драйвера из пакета `nvidia-dkms`, которые при помощи системы динамических модулей DKMS автоматически собируться под нужное ядро:

```
sudo pacman -S nvidia-dkms nvidia-utils lib32-nvidia-utils nvidia-settings lib32-
↪opencl-nvidia opencl-nvidia libxnvctrl vulkan-icd-loader lib32-vulkan-icd-loader␣
↪libva-nvidia-driver
```

Предупреждение

Для правильной работы DKMS требуется также установить заголовки текущей версии ядра. Например, для стандартного ядра `linux` заголовки требуемые для сборки модулей находится внутри пакета `linux-headers`.

С недавних пор помимо закрытых модулей драйвера NVIDIA также предоставляет версию модулей с открытым исходным кодом, которые рекомендуются к использованию начиная с `560` ветки драйвера. Их установка практически ничем не отличается от закрытого варианта кроме как заменой пакета `nvidia-dkms` на `nvidia-open-dkms`:

```
sudo pacman -S nvidia-open-dkms nvidia-utils lib32-nvidia-utils nvidia-settings lib32-
↪opencl-nvidia opencl-nvidia libxnvctrl lib32-vulkan-icd-loader libva-nvidia-driver
```

Перед установкой драйвера рекомендуется отключить *"Secure Boot"* в UEFI, ибо из-за этого модули драйвера могут не загрузиться.

1.2.2.5.2 NVIDIA (470xx)

Драйвер NVIDIA для Linux имеет несколько веток с долгосрочной поддержкой, часть из которых, как например nvidia-470xx-dkms, оставлены для сохранения совместимости со старыми видеокартами, в данном случае с поколением GPU Kepler. Если ваша видеокарта относится именно к этому поколению, то вам нужно установить не последний драйвер выше, а данную версию из AUR:

```
git clone https://aur.archlinux.org/nvidia-470xx-utils
cd nvidia-470xx-utils
makepkg -sric

sudo pacman -S lib32-vulkan-icd-loader

# 32-битные библиотеки (необходимо для запуска игр через Wine/Steam)
git clone https://aur.archlinux.org/lib32-nvidia-470xx-utils
cd lib32-nvidia-470xx-utils
makepkg -sric
```

1.2.2.5.3 Nouveau (*Только для старых видеокарт*)

Для старых видеокарт Nvidia (ниже GeForce 600) рекомендуется использовать свободную альтернативу драйвера NVIDIA — Nouveau, входящую в состав Mesa. Она имеет официальную поддержку и обновления в отличии от старых версий закрытого драйвера NVIDIA (340, 390) и отлично справляется с 2D ускорением. Вдобавок, Nouveau хорошо работает с Wayland:

```
sudo pacman -S mesa lib32-mesa vulkan-nouveau lib32-vulkan-nouveau opencl-rusticl-
↪mesa lib32-opencl-rusticl-mesa
```

1.2.2.5.4 AMD

```
sudo pacman -S mesa lib32-mesa vulkan-radeon lib32-vulkan-radeon vulkan-mesa-layers␣
↪opencl-rusticl-mesa lib32-opencl-rusticl-mesa
```

1.2.2.5.5 Intel

```
sudo pacman -S mesa lib32-mesa vulkan-intel lib32-vulkan-intel opencl-rusticl-mesa␣
↪lib32-opencl-rusticl-mesa
```

Предупреждение

Автор не рекомендует выполнять установку морально устаревших DDX драйверов, как например `xf86-video-intel`, так как они в большинстве своем заброшены и не получают никаких исправлений. Вместо этого используйте DDX драйвер `modesetting`, который поставляется вместе с пакетом `xorg-server`. Он использует аппартное ускорение на базе glamor и Mesa. Обратите внимание, что последние исправления и новые возможности (Как, например, опция `"Tearfree"`) доступны только в Git версии, поэтому имеет смысл установить `xorg-server-git` из AUR.

1.2.3 Удаление лишних пакетов

К сожалению, если во время установки системы вы выполняли установку KDE Plasma или GNOME при помощи одноименных групп пакетов, то скорее всего вы установили себе в систему некоторое количество лишних пакетов, таких как например `gnome-software` или `discover`, которые крайне не рекомендуется использовать в Arch Linux взамен простого использования `pacman`. Чтобы не выполнять переустановку всех пакетов, связанных с рабочим окружением, можно выполнить удаление лишних пакетов при помощи следующих команд в зависимости от используемого окружения:

GNOME

```
sudo pacman -D --asdeps $(pacman -Qqg gnome)
sudo pacman -D --asexplicit gnome-shell mutter gdm gnome-control-center gnome-console␣
↪nautilus gnome-session gnome-settings-daemon gvfs gvfs-mtp
sudo pacman -Rsn $(pacman -Qqgdtt gnome)
```

KDE Plasma

```
sudo pacman -D --asdeps $(pacman -Qqg plasma)
sudo pacman -D --asexplicit plasma-desktop breeze-gtk kde-gtk-config plasma-pa␣
↪bluedevil sddm sddm-kcm plasma-nm
sudo pacman -Rsn $(pacman -Qqgdtt plasma)
```

Если вас пугает большой набор непонятных команд - не переживайте, все что здесь происходит, это помечание всех пакетов из группы пакетов `gnome` или `plasma` соответственно как неявно установленных, то есть подтянутых в качестве зависимостей, после чего идет изменение причины установки базовых пакетов окружения уже как явно установленных, что позволяет разделить действительно нужные и мусорные пакеты по причине их установки и удалить все лишние пакеты. Конечно, всегда думайте головой и проверяйте не подтянулось ли что-то для вас нужное, однако данный способ гарантирует, что базовые пакеты, необходимые для работы окружения, не будут удалены, поэтому вы всегда сможете доустановить нужные вам программы в соответствии со своими предпочтениями.

Если вы не используйте GNOME или KDE Plasma, то вы можете пропустить данный шаг, так как для всех остальных рабочих окружений, таких как Xfce, MATE и LXQt, в соответствующей им группе пакетов есть лишь предельный минимум того, что действительно нужно.

1.2.4 Установка микрокода

Микрокод - программа реализующая набор инструкций процессора. Она уже встроена в материнскую плату вашего компьютера, но скорее всего вы её либо не обновляли вовсе, либо делаете это не часто вместе с обновлением BIOS (UEFI).

Однако у ядра Linux есть возможность применять обновления микрокода прямо во время загрузки системы. Они содержат множественные исправления ошибок и улучшения стабильности, поэтому настоятельно рекомендуется их периодически устанавливать.

Осуществляется это следующими командами в зависимости от используемого процессора:

Intel

```
sudo pacman -S intel-ucode
sudo mkinitcpio -P
```

AMD

```
sudo pacman -S amd-ucode
sudo mkinitcpio -P
```

1.2.5 Установка дополнительных прошивок

В Arch Linux и основанных на нем дистрибутивах большинство прошивок устройств как правило поставляются с пакетом linux-firmware и всех связанных с ним пакетов (linux-firmware-whence, linux-firmware-bnx2x, linux-firmware-liquidio, linux-firmware-marvell, linux-firmware-mellanox, linux-firmware-nfp, linux-firmware-qcom, linux-firmware-qlogic). Тем не менее вы можете столкнуться с предупреждением во время пересборки initramfs образов через команду `sudo mkinitcpio -P` подобного формата:

```
==> WARNING: Possibly missing firmware for module: XXXXXXXX
```

Такие предупреждения не являются критическими, однако некоторые устройства у вас в системе могут работать не полностью или вообще не работать без требуемых прошивок. Поэтому в первую очередь рекомендуется попробовать установить все вышеуказанные пакеты linux-firmware (некоторые из них можно пропустить в силу отсутствия соответствующих устройств, например linux-firmware-marvell).

Но некоторых прошивок нет в официальных репозиториях дистрибутива, поэтому их требуется установить отдельно из AUR (все пакеты содержащие файлы прошивок имеют окончание "-firmware"). Рассмотрим на примере прошивки для модуля aic94xx:

```
git clone https://aur.archlinux.org/aic94xx-firmware
cd aic94xx-firmware
makepkg -sric
```

После этого повторите команду `sudo mkinitcpio -P`. Предупреждение о пропуске прошивок для модуля aic94xx должно пропасть.

1.3 Настройка драйверов GPU

1.3.1 Настройка закрытого драйвера NVIDIA

Драйвер NVIDIA - одно из самых болезненных мест с которым сталкиваются пользователи при переходе на Linux. В данном разделе мы рассмотрим как минимизировать количество испытываемой головной боли если вы являетесь несчастным обладателем видеокарты NVIDIA под Linux.

1.3.1.1 Распространенные мифы о настройке драйвера

Но прежде чем перейти к исправлению насущных проблем, поговорим о том, чего делать определенно не стоит, то есть о различных вредных советах, которые раньше могли иметь смысл, но сейчас уже нет.

Во-первых, автор категорически не рекомендует выполнять настройку ваших мониторов и генерацию конфига `xorg.conf` в целом через nvidia-settings или nvidia-xconfig как советовалось делать ранее. В первую очередь потому, что это просто не нужно, так как современные версии графического сервера Xorg сами выполняют автонастройку и определение рабочих мониторов, кроме того большинство рабочих окружений (DE) в своих настройках уже позволяют вам выставить требуемую частоту обновления нескольких мониторов и их компановку, перекрывая при этом все изменения сделанные в файле `xorg.conf`, который статичен и не может подстраиваться под изменения вашей конфигурации. Например, подключение второго монитора на лету вызовет проблемы, так как он не указан в `xorg.conf`, а автоопределение при наличии конфигурационного файла перестает работать. Вдобавок, программа nvidia-settings также является крайне ограниченной в конфигурациях с гибридной графикой (PRIME) или при использовании сессий на базе протокола Wayland.

Больше подробностей о проблемах которые могут возникнуть при использовании nvidia-settings в качестве конфигуратора для Xorg можно прочитать здесь:

https://unix.stackexchange.com/questions/697517/how-to-correlate-xorg-conf-config-for-nvidia-gpu-with-xrandr-detected-screens/697553#697553

Автор так же рекомендует полностью отказаться от использования морально устаревшей опции для `xorg.conf` как "Force composition pipeline", так как современных композиторов, поставляемых вместе с такими рабочими окружениями как GNOME/KDE/Cinnamon/Xfce и т.д., полностью достаточно чтобы предотвратить так называемую проблему тиринга (разрывов экрана). К тому же "Force composition Pipeline" имеет побочный эффект, создавая сильные задержки отклика, так называемый input lag. И вызывает проблемы при работе расширения Vulkan VK_KHR_present_wait, которое требуется для запуска многих игр при использовании VKD3D-Proton. Подробнее смотрите здесь: https://github.com/ValveSoftware/Proton/issues/6869.

Если же вы являетесь пользователем тайлинговых оконных менеджеров (WM), где нет удобных средств настройки из коробки, то автор рекомендует вам использовать такие средства как xrandr и композитор picom.

1.3.1.2 Повышение производительности CPU на ноутбках с графикой NVIDIA

Обмен кадрами между iGPU и dGPU в следствии их копирования из VRAM в ОЗУ может вызывать большую нагрузку на процессор, из-за чего производительность самого CPU может сильно падать на ноутбуках с гибридной графикой. Page Attribute Table (PAT) это более оптимизированный метод управления памятью. Метод PAT создает таблицу типов разделов по определенному адресу, отображенному в регистре, и использует архитектуру памяти и набор инструкций более эффективно и быстро. Для его активации в драйвере NVIDIA нужно указать параметр `NVreg_UsePageAttributeTable=1`. Для этого как обычно создаем файл в директории `/etc/modprobe.d/` или указываем через пробел в тот же файл, что мы создавали ранее:

Список 3: `sudo nano /etc/modprobe.d/nvidia-pat.conf`

```
options nvidia NVreg_UsePageAttributeTable=1
```

См. также: https://bbs.archlinux.org/viewtopic.php?id=242007

1.3.1.3 Повышение лимита TDP на ноутбках GPU Ampere и выше

К сожалению, в новых версиях драйверов NVIDIA невозможно зафиксировать определенное значение TDP вручную через nvidia-smi. Но для пользователей ноутбуков с GPU поколения Ampere (RTX 30xx) и выше, есть обходной путь, который частично решает проблему путем незначительного повышения лимита TDP. Для этого необходимо включить службу `nvidia-powerd`, которая включает технологию Dynamic Boost:

```
sudo systemctl enable nvidia-powerd
```

Например, на ноутбуке с 3050 Mobile это позволяет динамически повышать (т.е. в зависимости от нагрузки на систему) лимит TDP видеокарты с 35 Вт до 40 Вт, без значительного изменения температуры, но с повышением на 10 FPS в бенчмарке Furmark.

Примечание

Обратите внимание, что технология Dynamic Boost работает только тогда, когда ноутбук работает от сети и это также влияет на производительность процессора, изменяя максимальную частоту работы CPU.

1.3.1.4 Специальные переменные окружения

Здесь речь пойдет о переменных окружения, которые влияют на поведение драйвера при работе с приложениями которые используют API OpenGL или Vulkan. Указать вы их можете либо в Lutris для конкретных игр, либо в *"Параметрах Запуска"* игры в Steam (*"Свойства"* -> *"Параметры запуска"*. После указания всех переменных обязательно добавьте в конце "*%command%*", для того чтобы Steam понимал, что вы указали именно системные переменные окружения для запуска игры, а не параметры специфичные для этой самой игры).

`__GL_THREADED_OPTIMIZATIONS=1` **(По умолчанию выключено)** - Активируем многопоточную обработку OpenGL. Используете выборочно для нативных игр/приложений, ибо иногда может наоборот вызывать регрессию производительности. Некоторые игры и вовсе могут не запускаться с данной переменной (К примеру, некоторые нативно-запускаемые части Metro).

`__GL_MaxFramesAllowed=1` **(По умолчанию - 2)** - Задает тип буферизации кадров драйвером. Можете указать значение *"3"* (Тройная буферизация) для большего количества FPS и улучшения производительности в приложениях/играх с VSync. Мы рекомендуем задавать вовсе *"1"* (т.е. не использовать буферизацию, подавать кадры так как они есть). Это может заметно уменьшить значение FPS в играх, но взамен вы получите лучшие задержки отрисовки и реальный физический отклик, т.к. кадр будет отображаться вам сразу на экран без лишних этапов его обработки.

`__GL_YIELD="USLEEP"` **(По умолчанию без значения)** - Довольно специфичный параметр, *"USLEEP"* - снижает нагрузку на CPU и некоторым образом помогает в борьбе с тирингом, а *"NOTHING"* дает больше FPS при этом увеличивая нагрузку на процессор.

`__GL_SHADER_DISK_CACHE_SKIP_CLEANUP=1` **(По умолчанию 0)** - отключает ограничение кэша шейдеров OpenGL/Vulkan (по умолчанию располагается по пути `~/.cache/nvidia`). Рекомендуется для современных нативных игр и DXVK 2.0+, где размер кэша может достигать более гигабайта.

`__GL_SYNC_DISPLAY_DEVICE` - указывает монитор с частотой которого драйвер NVIDIA будет осуществлять синхронизацию. Это нужно для конфигураций с двумя и более мониторами для предотвращения заиканий на дисплеях с высокой частотой обновления на некоторых композиторах (например Muffin). Представим, что у вас есть два дисплея, один 144 Гц, другой 60. В переменную следует указать видеовыход, в который подключен монитор с наибольшей частотой обновления. Чтобы определить какой монитор, в какой выход подключен можно использовать утилиту `xrandr`. Данную переменную имеет смысл указывать глобально, то есть в файле `/etc/environment`.

Список 4: `sudo nano /etc/environment`

```
__GL_SYNC_DISPLAY_DEVICE=HDMI-0 # Это пример, указывайте свое имя выхода
```

1.3.1.5 Добавление драйверов GPU в образы initramfs

Иногда может понадобиться добавить модули для ядра в начальный загрузочный образ (initramfs), который содержит в себе всё необходимое для правильной загрузки ядра, включая необходимые драйверы и программы для монтирования корневого раздела. Добавление модулей в initramfs позволяет избежать некоторых проблем, когда загрузка модулей происходит позже, чем когда они уже фактически нужны для использования. Так происходит из-за параллельной загрузки сервисов (юнитов) в systemd. Например, при использовании графического менеджера входа в систему как SDDM, модули драйвера Nvidia могут быть загружены позже, чем будет запущен сам сервис SDDM. Из-за этого пользователь ловит так называемый "Чёрный экран".

За генерацию загрузочных образов могут отвечать различные программы, но в Arch Linux по умолчанию для этого используется `mkinitcpio`. Чтобы добавить необходимые модули нужно создать новый файл со следующим содержанием:

Список 5: `sudo nano /etc/mkinitcpio.conf.d/10-modules.conf`

```
MODULES+=(nvidia nvidia_modeset nvidia_uvm nvidia_drm)
```

Здесь мы добавляем модули драйвера Nvidia в загрузочный образ initramfs, но в массив (ограничен скобками) вы можете прописать любые модули ядра, которые вы считаете нужными для использования вовремя загрузки.

После добавления новых модулей обязательно нужно пересобрать уже существующие образы initramfs:

```
sudo mkinitcpio -P
```

Предупреждение

Обратите внимание, что добавление большого количества модулей может сильно раздуть размер образа initramfs, поэтому перед этим убедитесь, что у вас имеется достаточно места в `/boot` разделе (если таковой имеется).

1.3.2 Настройка драйверов Mesa

1.3.2.1 Форсирование использования AMD SAM *(Только для опытных пользователей).*

AMD Smart Acess Memory (или Resizble Bar) — это технология которая позволяет процессору получить доступ сразу ко всей видеопамяти GPU, а не по блокам в 256 мегабайт, что приводит к задержкам ввода/вывода при обмене между CPU и GPU. Несмотря на то, что данная технология заявлена только для оборудования AMD и требует новейших комплектующих для обеспечения своей работы, получить её работу можно и на гораздо более старом оборудовании, например таком как AIT Radeon HD 7700.

Предупреждение

Для включения данной технологии в настройках вашего BIOS (UEFI) должна быть включена опция *"Re-Size BAR Support"* и *"Above 4G Decoding"*. Если таких параметров в вашем BIOS (UEFI) нет - скорее всего технология не поддерживается вашей материнской платой и не стоит даже пытаться её включить.

К сожалению, после недавнего обновления драйверов Mesa, поддержка SAM была удалена из драйвера OpenGL - radeonsi, но вы по прежнему можете заставить Mesa использовать SAM при работе в приложениях использующих Vulkan.

Чтобы активировать SAM в Linux, нужно добавить переменные окружения:

Список 6: `sudo nano /etc/environment`

```
RADV_PERFTEST=sam # Только для Vulkan
```

Предупреждение

Учтите, что в некоторых играх с vkd3d вам может понадобиться также экспортировать переменную `VKD3D_CONFIG=no_upload_hvv` для избежания регрессий производительности при использовании вместе с SAM.

https://www.reddit.com/r/linux_gaming/comments/119hwmt/this_setting_may_help_vkd3d_games_that_have/

1.3.2.2 Решение проблем работы графики Vega 11 (Спасибо @Vochatrak-az-ezm)

На оборудовании со встроенным видеоядром Vega 11 может встретиться баг драйвера, при котором возникают случайные зависания графики. Проблема наиболее актуальна для *Ryzen 2XXXG* и чуть реже встречается на Ryzen серии *3XXXG*, но потенциально имеет место быть и на более новых видеоядрах Vega.

Решается через добавление следующих параметров ядра:

Список 7: `sudo nano /etc/modprobe.d/90-amdgpu.conf`

```
options amdgpu gttsize=8192 lockup_timeout=1000 gpu_recovery=1 noretry=0␣
↪ppfeaturemask=0xfffd3fff deep_color=1
```

На всякий случай можно дописать ещё одну переменную окружения:

Список 8: `sudo nano /etc/environment`

```
AMD_DEBUG=nodcc
```

Для подробностей можете ознакомиться со следующими темами:

https://www.linux.org.ru/forum/linux-hardware/16312119

https://www.linux.org.ru/forum/desktop/16257286

1.3.2.3 Многопоточная OpenGL обработка

У Mesa есть свой аналог переменной окружения `__GL_THREADED_OPTIMIZATIONS=1`, так же предназначенный для активирования многопоточной обработки OpenGL - `mesa_glthread=true`. В ряде игр и приложений это даёт сильное увеличение производительности, но в некоторых либо нет прироста, либо вовсе не может быть применено.

Чтобы включить его для всей системы нужно либо прописать переменную окружения в файл `/etc/environment`, либо используя adriconf, включив параметр во вкладке *"Performance" -> "Enable offloading GL driver work to a separate thread"*

1.3.2.4 Ускорение с помощью OpenCL

В 2021 году разработчики Mesa представили реализацию драйвера OpenCL, основанную на языке Rust, под не замысловатым названием - *Rusticl* Данная реализация призвана заменить старый драйвер *Clover*, и ныне совместима с OpenCL 3.0.

Для правильной работы аппаратного видео ускорения в таких приложениях как Handbrake, Darktable и DaVinci Resolve вам может понадобиться установить данный драйвер.

Чтобы установить данную реализацию драйвера, выполните данную команду в терминале:

```
sudo pacman -S opencl-rusticl-mesa
```

Теперь чтобы включить поддержку OpenCL на вашей видеокарте, укажите данные значения в переменную `RUSTICL_ENABLE` в зависимости от вашего производителя GPU:

AMD

Список 9: `sudo nano /etc/environment`

```
RUSTICL_ENABLE=radeonsi
```

Примечание

Данная переменная будет работать на видеокартах с драйвером AMDGPU.

Intel

Список 10: `sudo nano /etc/environment`

```
RUSTICL_ENABLE=iris
```

Примечание

Данный драйвер будет поддерживается на видеокартах, начиная с поколения Broadwell (Gen8) и новее.

1.3.2.5 Аппаратное ускорение видео на старых iGPU от Intel

К сожалению старые встраиваемые графические процессоры от Intel редко удостаиваются вниманием со стороны разработчиков драйверов, в следствии чего некоторые их возможности в Linux могут быть весьма ограничены. Так с недавних пор Intel прекратила разработку старого драйвера VAAPI, используемого для аппаратного ускорения видео в Linux, в пользу нового драйвера intel-media-driver. Вместе с этим повысились требования к минимально поддерживаемому поколению - теперь это Broadwell (8-е поколение графики Intel), из-за чего пользователи старых поколений, всё ещё имеющих возможность декодировать видео на аппаратном уровне, остались без поддержки. В частности, начиная с версии Chromium 116 аппаратное декодирование видео перестало работать для поколений Haswell/Ivy Bridge/Sandy Bridge/Nehalem (Ironlake).

Однако относительно недавно появился форк драйвера Intel VAAPI для старых поколений графики, так называемый IRQL форк, который решает многие из проблем и позволяет получить правильно работающее аппаратное видео ускорение в Chromium и основанных на нём браузерах. Если вы являетесь обладателем встроенной графики Intel автор настоятельно рекомендует установить вам данный вариант драйвера из AUR:

```
git clone https://aur.archlinux.org/libva-intel-driver-irql
cd libva-intel-driver-irql
sed -i 's/tag=.*/branch=ilk-fix/g' PKGBUILD
makepkg -sricCf
```

Обратите внимание, что на поколениях Ironlake и Sandybridge (Gen 5/Gen 6), для правильной работы аппаратного ускорения в Chromium требуется использовать со следующие флаги для запуска и окружение на базе Wayland:

Список 11: `nano ~/.config/chromium-flags.conf`

```
--enable-features=AcceleratedVideoDecodeLinuxZeroCopyGL,AcceleratedVideoDecodeLinuxGL
--ozone-platform-hint=wayland
```

Без использования Wayland сессии вместе с указанными флагами для Chromium будет наблюдатся чёрный (прозрачный) квадрат вместо видео. Хотя на остальных поколениях графики Intel использование данных параметров не обязательно, параметр `AcceleratedVideoDecodeLinuxZeroCopyGL` в окружениях на базе Wayland активирует использование подсистемы dma-buf для прямого доступа графического процессора к буферу для отрисовки видео, что может заметно улучшить производительность.

1.4 Гибридная графика в ноутбуках

Гибридная графика - это аппаратная конфигурация, в которой есть две и более видеокарт, которые могут работать в тандеме друг с другом. Такой подход встречается в основном в ноутбуках, где есть интегрированная графика (iGPU) от процессора и дискретная графика (dGPU). Его основное преимущество заключается в том, что интегрированная графика должна (но не обязательно) использоваться только для малозатратных задач, таких как серфинг в Интернете, просмотр видео и т. д. А дискретная графика используется для видеоигр, видеомонтажа, 3D-моделирования и так далее. Следовательно, если два GPU разделяют "большие" и "маленькие" задачи, и в данный момент выполняются только "маленькие" задачи, то нет никакого смысла в работе dGPU, поэтому его можно просто временно отключить (то есть усыпить), тем самым значительно снизив энергопотребление. Таким образом, когда dGPU снова понадобится (мы запустим приложение, которое его использует), он проснется и возобновит работу.

1.4.1 Что такое PRIME?

PRIME - это универсальная технология для работы с различными наборами гибридной графики в Linux, такими как NVIDIA Optimus/AMD Dynamic Switchable Graphics. PRIME Offload - это реализация идеи переноса выполнения рендеринга с одного GPU на другой, но вот отображением уже готовых кадров на экран заниматься будет тот GPU, который управляет дисплеем в данный момент, как правило это встроенная графика. Поддержка PRIME в закрытом драйвере NVIDIA фактически началась только с драйвера 435.17, поэтому если вы являетесь пользователем устаревшей ветки драйверов 390xx или даже 340xx, PRIME Offload не будет работать для вас.

Обратите внимание, что автор настоятельно не рекомендует вам использовать устаревшие способы использования гибридной графики, такие как nvidia-xrun или Bumblebee. Они устарели и не поддерживаются (Bumblebee не обновлялся уже более 8 лет), работают исключительно на убогих хаках и имеют низкую производительность. В то же время открытый драйвер Nouveau поддерживает PRIME Offload, что может стать неплохой альтернативой для старых dGPU.

Кроме того, пожалуйста, избегайте использования таких инструментов, как optimus-manager. Они могут показаться вам очень удобными, но поверьте, они могут вызвать множество проблем, и они вам на самом деле не нужны, если ваш графический процессор поддерживает PRIME Offload и динамическое управление питанием.

1.4.2 Настройка PRIME

В Arch Linux никакой дополнительной настройки системы, в частности графического сервера Xorg, не требуется. Для современных версий драйвера все должно работать из коробки при условии, что все модули NVIDIA загружаются правильно, а проверить это можно при помощи команды `lspci -k | grep "Kernel driver in use: nvidia"`, вы уже можете полноценно использовать вашу дискретную видеокарту. .. note:: Если модули драйвера NVIDIA по какой-то причине не загружаются, убедитесь, что вы отключили *Secure Boot* в настройках UEFI.

Для конфигураций, где обе видеокарты управляются открытыми драйверами Mesa (к примеру AMD+AMD, AMD+Intel или даже Intel+NVIDIA, где дискретная графика NVIDIA находится под управлением открытого драйвера Nouveau), ничего настраивать не нужно и для использования дискретной графики нужно лишь указать переменную окружения `DRI_PRIME=1` перед запуском приложений или игр по аналогии со всеми переменными, которые будут описаны далее для NVIDIA или использовать уже готовые графические способы.

1.4.2.1 Динамическое управление питанием

Поддержка динамического управления питанием вашей дискретной видеокарты позволяет драйверу полностью отключать её питание на время простоя, то есть когда в ней нет необходимости, она просто уходит в состоянии сна не потребляя никакой энергии. К сожалению, она доступна не для всех и работает только для мобильных видеокарт NVIDIA начиная с поколения Turing (16xx/20xx) и выше. Для видеокарт поколения Ampere её поддержка включена по умолчанию и никаких дополнительных действий производить не нужно, однако если вы являетесь обладателем графики именно поколения Turing, то вам так же понадобиться создать дополнительные файлы настройки и правила udev, которые будут выполнять настройку динамического управления питания при каждом запуске. Для этого просто создайте файл под названием `/etc/udev/rules.d/71-nvidia.rules` со следующим содержанием:

Список 12: `sudo nano /etc/udev/rules.d/71-nvidia.rules`

```
ACTION=="add|bind", SUBSYSTEM=="pci", DRIVERS=="nvidia", ATTR{vendor}=="0x10de", ATTR
↪{class}=="0x03[0-9]*", TEST=="power/control", ATTR{power/control}="auto"
ACTION=="remove|unbind", SUBSYSTEM=="pci", DRIVERS=="nvidia", ATTR{vendor}=="0x10de",␣
↪ATTR{class}=="0x03[0-9]*", TEST=="power/control", ATTR{power/control}="on"
```

Предупреждение

Динамическое управление питание на текущий момент не работает для видеокарт поколения Turing при использовании открытых модулей драйвера NVIDIA (`nvidia-open-dkms`)[1].

Вам также потребуется изменить значение параметра для модуля драйвера NVIDIA, который и активирует работу динамического управления питанием:

Список 13: `sudo nano /etc/modprobe.d/nvidia-dynamic-powermanagment.conf`

```
options nvidia NVreg_DynamicPowerManagement=0x02
```

После чего рекомендуется обновить образы initramfs через команду `sudo mkinitcpio -P` и перезагрузиться. Чтобы убедиться, что все работает правильно нужно проверить вывод команды `cat /proc/driver/nvidia/gpus/*/power`, он должен быть примерно следующим:

[1] https://forums.developer.nvidia.com/t/clarifying-560-series-drivers-open-sourcedness-vs-kernel-module-type-proprietary/292698/2

```
Runtime D3 status:          Enabled (fine-grained)
Video Memory:               Active

GPU Hardware Support:
 Video Memory Self Refresh: Supported
 Video Memory Off:          Supported

S0ix Power Management:
 Platform Support:          Supported
 Status:                    Disabled
```

1.4.3 Использование PRIME Offload

Чтобы указать, что вы хотите использовать дискретную графику вместо встроенной, перед запуском программы необходимо указать несколько переменных окружения:

```
__NV_PRIME_RENDER_OFFLOAD=1 __VK_LAYER_NV_optimus=NVIDIA_only __GLX_VENDOR_LIBRARY_
↪NAME=nvidia <program>
```

Такой набор переменных выглядит очень громоздким и его легко забыть, поэтому вы можете установить пакет `nvidia-prime` (`sudo pacman -S nvidia-prime`), который содержит скрипт-псевдоним `prime-run` для всех этих переменных. Тогда запуск приложения с его помощью будет выглядеть следующим образом:

```
prime-run  <program>
```

Где `<program>` - это имя команды, запускающей ваше приложение. Например, вы можете запустить команду `glxinfo`, чтобы проверить корректность работы PRIME Offload:

```
prime-run glxinfo | grep OpenGL
```

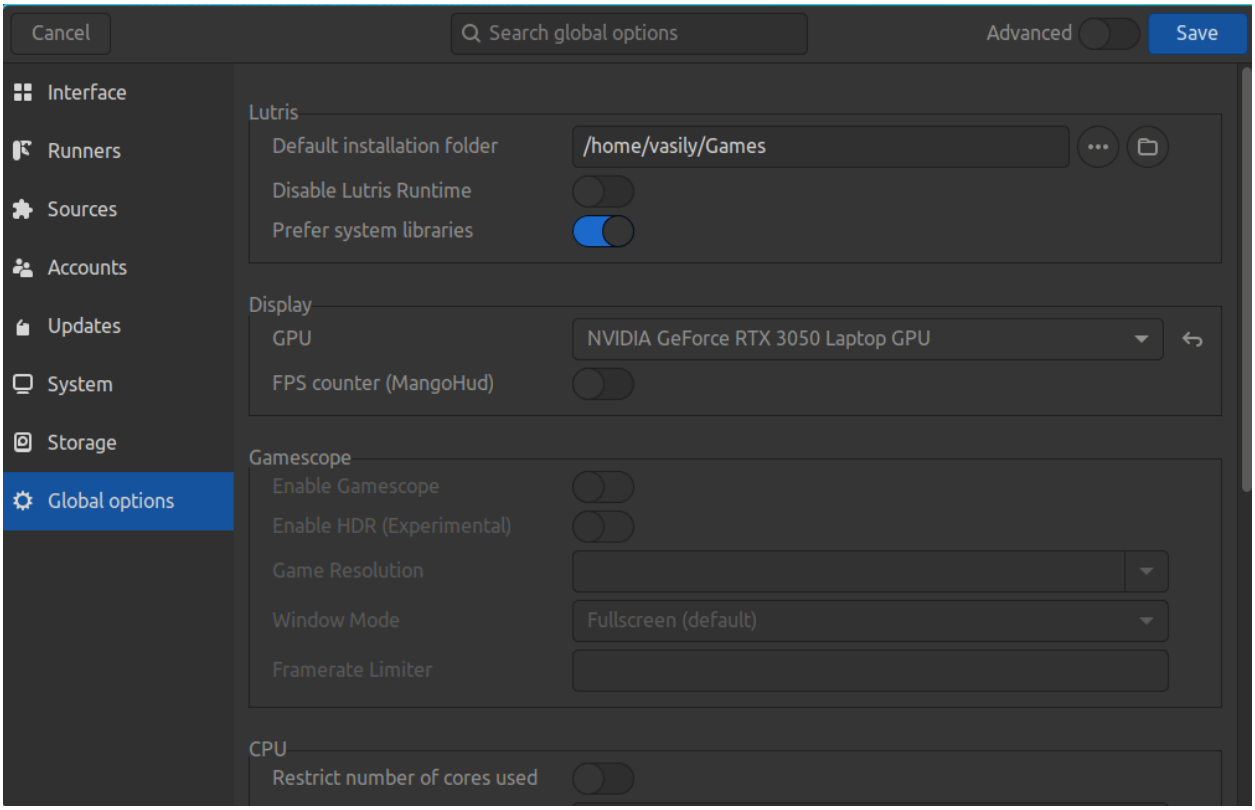
Если вывод данной команды даёт вам упоминание вашей дискретной видеокарты, значит вы всё сделали правильно. При возникновении проблем, советуем вам перепроверить правильность установки драйвера NVIDIA и загрузки всех модулей.

1.4.3.1 Графический способ

Возможно, запуск всех необходимых приложений через терминал с помощью `prime-run` покажется вам не слишком удобным, но некоторые приложения позволяют вам указывать запуск игр/других приложений с использованием дискретной графики через специальные настройки.

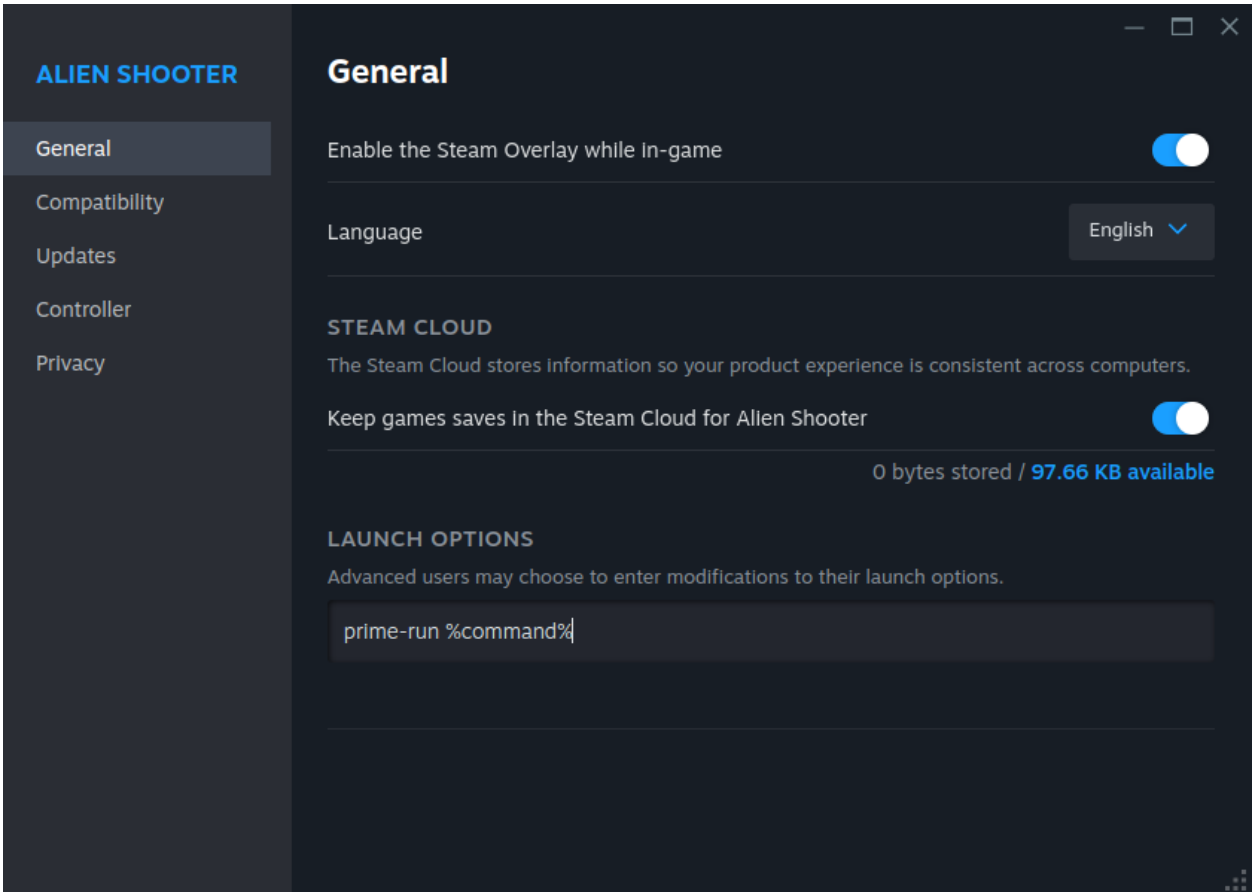
1.4.3.1.1 Lutris

Чтобы настроить запуск игр с использованием дискретной графики в Lutris, нужно зайти в настройки (три полоски в правом верхнем углу окна и кнопка *"Настройки"*). Далее перейдите в раздел *"Global options"* -> *"Display"* -> *GPU"*. Здесь вы выбираете графический процессор, с которым будет запускаться игра.



1.4.3.1.2 Steam

В Steam нет специальных настроек для запуска игр с дискретной графикой, но вы можете нажать на *"Шестеренку"* и перейти в *"Свойства"* перед запуском игры, после чего прописать команду `prime-run %command%` или указанные ранее переменные окружения, например:



1.4.3.1.3 Графические окружения

Ряд пользовательских окружений таких как KDE Plasma, Cinnamon и GNOME имеют встроенную интеграцию с PRIME при помощи специального инструмента как `switcheroo-control`. Чтобы получить заветную удобную кнопку для запуска приложений с использованием дискретной графики в данных окружениях и без необходимости лезть в терминал, вы должны установить данный пакет и включить соответствующую службу:

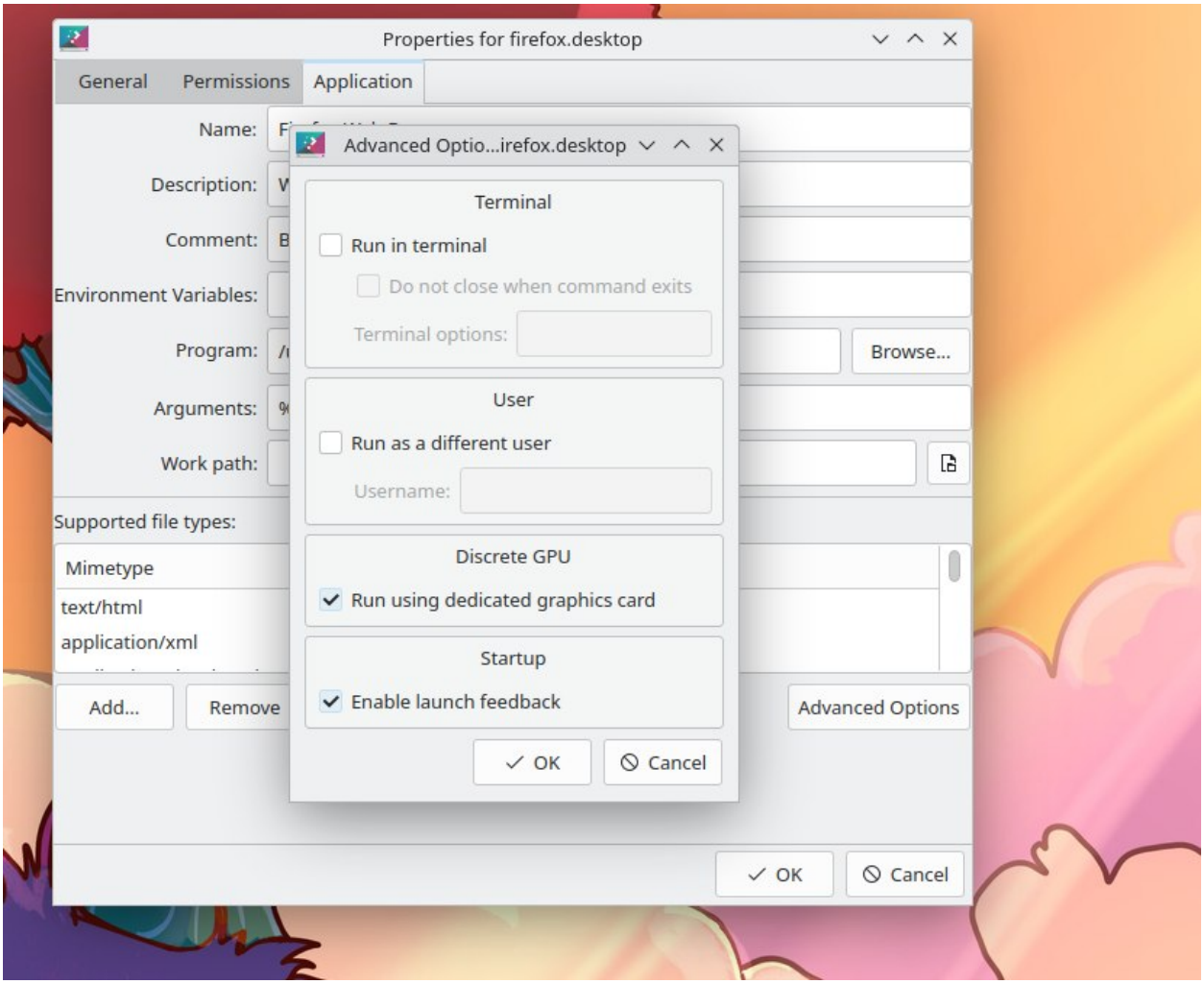
```
sudo pacman -S switcheroo-control
sudo systemctl enable --now switcheroo-control
```

Примечание

Использование switcheroo-control позволяет работать описанным ниже графическим способам на всех конфигурациях PRIME, в которых даже нет NVIDIA dGPU, например AMD+AMD.

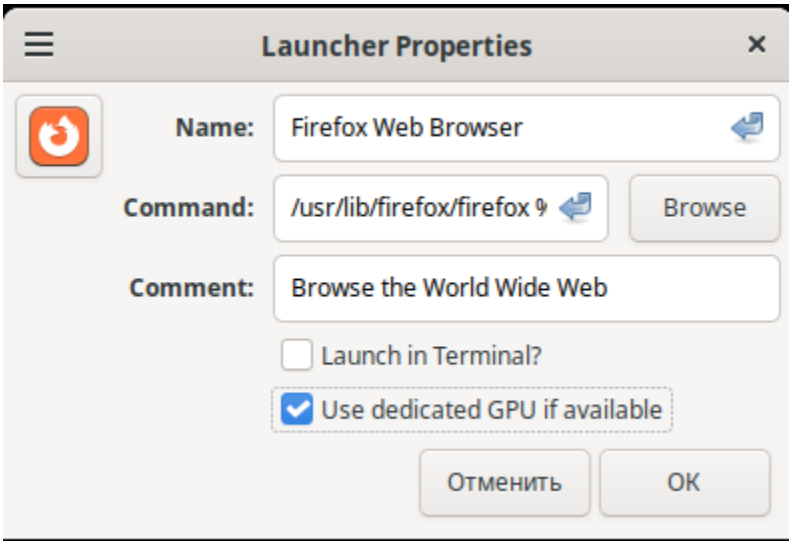
KDE Plasma

После установки `switcheroo-control` щелкните правой кнопкой мыши на нужную вам значок приложения на рабочем столе или в меню приложений, затем перейдите в *"Свойства"* -> *"Приложение"* -> *"Дополнительные параметры"* -> *"Запускать с использованием дискретной графики"*.



Cinnamon

По аналогии с Plasma в Cinnamon, вы также можете зайти в меню приложений, кликнуть правой кнопкой мыши по значку приложения и в появившемся контекстном меню выбрать *"Свойства"*, после чего в появившемся меню поставить флажок *"Использовать выделенный графический процессор, если он доступен"*.



GNOME

В GNOME также следует щелкнуть правой кнопкой мыши по значку приложения и выбрать *"Запустить с дискретной графикой"*. Но учтите, что GNOME не запоминает этот выбор, и в следующий раз, когда вы просто щелкните по значку, приложение все равно будет работать с интегрированной графикой.

1.4.4 Устранение проблем с PRIME

1.4.4.1 Внешний монитор сильно тормозит

Это известная проблема с драйвером NVIDIA, связанная с тем, что порт для подключения монитором управляется NVIDIA dGPU, в то время как композитинг и отображение кадров выполняется с учетом работы iGPU. Вам следует установить последнюю версию драйвера NVIDIA и использовать Wayland с композитором, поддерживающим явную синхронизацию. Для GNOME это было исправлено в версии 46.2. Для Plasma 6 это, вероятно, было исправлено в версии 6.1, хотя некоторые пользователи сообщают о нормальной работе уже в версии 6.0. В других окружениях/оконных менеджерах эта проблема все еще присутствует, поэтому для ее устранения необходимо перейти на последнюю версию GNOME или Plasma.

В случае работы в окружениях на базе протокола X11 вы также можете решить эту проблему переставив iGPU и dGPU местами, сделав тем самым дискретную графику NVIDIA главным GPU, что позволяет избавиться от тормозов. Чтобы это сделать нужно создать следующий конфиг:

Список 14: `sudo nano /etc/X11/xorg.conf.d/10-gpu.conf`

```
Section "ServerLayout"
  Identifier "layout"
  Screen 0 "nvidia"
  Inactive "integrated"
EndSection

Section "Device"
    Identifier  "nvidia"
    Driver      "nvidia"
    BusID       "PCI:1:0:0" # Например: "PCI:1:0:0"
EndSection
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
Section "Screen"
    Identifier "nvidia"
    Device "nvidia"
    Option "AllowEmptyInitialConfiguration"
EndSection

Section "Device"
    Identifier  "integrated"
    Driver      "modesetting"
EndSection

Section "Screen"
    Identifier "integrated"
    Device "integrated"
EndSection
```

В поле "BusID" вы должны указать собственное значение номера шины вашего dGPU в том формате, в котором он указан в примере. Изменить нужно лишь первую цифру из примера с `1` на ваш номер шины, если конечно он уже не равен `1`. Его вы можете узнать при помощи следующей команды: `lspci -Dd "10de:*:030x" | cut -f 2 -d ":" | awk '{x=$0+0;print x}'`. После сохранения файла и перезагрузки вашим GPU по умолчанию должна стать NVIDIA графика.

1.5 Ускорение загрузки системы

1.5.1 Ускорение распаковки initramfs

Как уже было сказано, initramfs - это начальное загрузочное окружение, которое идет в дополнение к образу ядра Linux и должно содержать в себе все необходимые ядру модули и утилиты для его правильной загрузки (прежде всего необходимые для монтирования корневого раздела). Для экономии места на загрузочном разделе данное окружение поставляется в виде самораспаковывающегося архива, который распаковывается на лету во время загрузки системы. В Arch Linux программа для генерации initramfs - mkinitcpio, по умолчанию сжимает их при помощи алгоритма zstd, который имеет оптимальные показатели скорости сжатия и распаковки. При этом понятно, что скорость сжатия initramfs не так важна, как скорость распаковки - ведь она напрямую влияет на скорость загрузки системы. Поэтому для ускорения данного процесса лучше всего использовать алгоритм с самой быстрой скоростью распаковки - `lz4`.

Чтобы использовать `lz4` в качестве основного алгоритма сжатия для initramfs, нам следует отредактировать конфигурационный файл `/etc/mkinitcpio.conf` и добавить в него следующие строчки:

Список 15: /etc/mkinitcpio.conf

```
COMPRESSION="lz4"
COMPRESSION_OPTIONS=(-9)
```

Не забываем обновить все образы initramfs после проделанных изменений:

```
sudo mkinitcpio -P
```

1.5.2 Ускорение загрузки системы с помощью systemd

Есть ещё способ ускорить загрузку системы, используя систему инициализации systemd, указав её использование на самом раннем этапе загрузки ядра внутри initramfs окружения. Для этого нужно убрать `base` и `udev` из массива `HOOKS` в файле `/etc/mkinitcpio.conf`, и заменить их на `systemd` чтобы он выглядел примерно так:

Список 16: sudo nano /etc/mkinitcpio.conf

```
HOOKS=(systemd autodetect microcode modconf kms keyboard sd-vconsole block␣
↪filesystems fsck)
```

Предупреждение

Для систем с зашифрованным корневым разделом к представленному перечню хуков вам также следует добавить `sd-encrypt` через пробел сразу после хука `sd-vconsole`.

Это немного увеличит образ initramfs, но заметно может ускорить запуск системы на пару секунд.

Не забываем обновить все образы initramfs после проделанных изменений:

```
sudo mkinitcpio -P
```

1.6 Настройка служб

1.6.1 Полезные службы

1.6.1.1 zram-generator

zram-generator — демон для создания блочных устройств ZRAM. ZRAM - это альтернативный механизм подкачки в ядре Linux, который позволяет избавиться от обычной подкачки на диске и сжимать неиспользуемые данные прямо внутри памяти ресурсами CPU. Больше подробностей о том, как именно работает подкачка и в частности ZRAM вы можете в разделе *Настройка параметров ядра*. Установка `zram-generator` выполняется всего парой команд:

```
sudo pacman -S zram-generator
```

После установки необходимо создать конфиг с указанием всех желаемых параметров, таких как алгоритм сжатия и размер блочного устройства:

Список 17: `sudo nano /etc/systemd/zram-generator.conf`

```
[zram0]
zram-size = ram
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
compression-algorithm = zstd
swap-priority = 100
fs-type = swap
```

Здесь мы указываем размер ZRAM равным количеству оперативной памяти (`zram-size`), а также алгоритм сжатия zstd. Это позволит экономить больше памяти, так как по заверениям разработчиков, эффективность сжатия в среднем равна 1:3, что позволяет хранить внутри ZRAM объем данных больший, чем вы в принципе можете уместить в ОЗУ. Вы также можете указать размер даже больший чем RAM, к примеру `ram * 2`, чтобы гарантировать, что размера блочного устройства всегда хватит для сжатия большего количества страниц.

После создания файла конфигурации можно запускать саму службу:

```
sudo systemctl daemon-reload
sudo systemctl start systemd-zram-setup@zram0.service
```

Предупреждение

Во избежание конфликтов, после установки zram обязательно отключите zswap через добавление параметра ядра `zswap.enabled=0`.

Примечание

Как уже было сказано ранее, сжатие страниц в памяти осуществляется ресурсами CPU, но если он у вас достаточно слабый, то вы можете оказаться чувствительными к задержкам на распаковку/сжатие большого объема страниц. В этом случае имеет смысл либо вернуться к использованию обычного свопа, либо использовать менее ресурсоемкий алгоритм сжатия (`compression-algorithm`) как `lzo`.

1.6.1.2 systemd-oomd

OOM киллером называют специальный демон, который предотвращает возникновение так называемых OOM (Out-Of-Memory) ситуаций. Если по простому, то он просто убивает самый "жирный" процесс в группе процессов (cgroup) прежде, чем он забьет всю память и ваш компьютер зависнет. В ядре Linux уже есть встроенный OOM киллер, но он отличается медленной скоростью реакции, поэтому лучше использовать OOM киллер в пространстве пользователя. Система инициализации systemd предлагает встроенный OOM киллер - `systemd-oomd`, который отличается малым потреблением ресурсом в фоне и не создает нагрузки на процессор, которая свойственна другим OOM киллерам из-за отслеживания потребления памяти процессов без использования механизма PSI, предлагаемым ядром Linux. Поэтому именно его и рекомендуется использовать. Включить его можно при помощи данной команды:

```
sudo systemctl enable --now systemd-oomd
```

1.6.1.3 Ananicy CPP

Ananicy CPP — это форк одноименного демона, распределяющий приоритет задач. Его установка очень сильно повышает отклик системы. В отличии от оригинального Ananicy, данный форк переписан полностью на C++, из-за чего достигается прирост в скорости работы:

```
sudo pacman -S ananicy-cpp
sudo systemctl enable --now ananicy-cpp
```

Кроме того рекомендуется установить уже готовую, большую базу правил для ananicy-cpp:

```
# Далее описывается установка дополнительных правил по перераспределению приоритетов␣
↪процессов
git clone https://aur.archlinux.org/cachyos-ananicy-rules-git.git
cd cachyos-ananicy-rules-git
makepkg -sric
sudo systemctl restart ananicy-cpp
```

1.6.1.4 TRIM

TRIM - это встроенная команда контроллера для очищения уже неиспользуемых ячеек на твердотельном накопителе. Её очень полезно периодически выполнять с целью профилактики SSD. Чтобы это происходило автоматически раз в неделю рекомендуется включить уже готовую службу:

```
sudo systemctl enable fstrim.timer
```

Если по каким-то причинам вы не используете systemd или вам нужно выполнить TRIM прямо сейчас воспользуйтесь одноименной командой `fstrim`:

```
sudo fstrim -v /
```

Предупреждение

Если вы используйте файловую систему Btrfs и имеете версию ядра 6.2 и выше, то выполнять включение службы для осуществления периодического выполнения команды TRIM - не нужно, т. к. Btrfs сам выполняет её в асинхронном режиме.

1.6.1.5 irqbalance

irqbalance - это демон, что автоматически балансирует обработку прерываний по ядрам процессора.

```
sudo pacman -S irqbalance
sudo systemctl enable --now irqbalance
```

1.6.2 Отключение лишних служб

Мы разобрались с установкой и включением полезных служб, теперь неплохо было бы отключить все лишнее, что есть в системе. Для этого прежде всего нужно проанализировать какие службы тормозят запуск системы при помощи следующей команды: `systemd-analyze blame` - она отсортирует все службы по скорости их загрузки. Не следует торопиться и отключать все подряд, нужно внимательно вчитываться в описание каждой службы. Стоит обратить свое внимание также на пользовательские службы рабочих окружений KDE Plasma, GNOME и Cinnamon (если вы их не используете, то можете просто пропустить разделы связанные с ними).

1.6.2.1 Службы индексирования файлов

Многие пользователи Windows знают о службе индексирования поиска, которая занимается тем, что в фоновом режиме проходит по всей файловой системе в поисках новых файлов или каталогов, чтобы внести их в собственную базу, которая будет использована для ускорения встроенного поиска или поиска в файловом менеджере. На первый взгляд все звучит хорошо, но на деле процессы всех подобных служб являются очень прожорливыми и часто создают чрезмерную нагрузку на диск.

В Linux подобные службы есть только у рабочих окружений GNOME и KDE Plasma. В KDE Plasma встроенным файловым индексатором является Baloo, который отличается своей склонностью часто "подтекать" с точки зрения потребления памяти, а в GNOME есть Tracker 3, который хоть и менее прожорливый по сравнению с аналогом от KDE, но все ещё потребляет не мало ресурсов.

Так как отключение всех подобных служб может положительно влиять на жизненный цикл вашего носителя, то рекомендуется выполнить это сразу после установки в зависимости от вашего рабочего окружения:

GNOME > 47

```
systemctl --user mask localsearch-3.service localsearch-control-3.service \
   localsearch-writeback-3.service
```

GNOME <= 46

```
systemctl --user mask tracker-extract-3 tracker-miner-fs-3 \
   tracker-miner-fs-control-3 tracker-miner-rss-3 tracker-writeback-3 \
   tracker-xdg-portal-3
rm -rf ~/.cache/tracker ~/.local/share/tracker
```

KDE Plasma

```
balooctl6 suspend
balooctl6 disable
balooctl6 purge
```

Предупреждение

Обратите внимание, что после отключения встроенный поиск в GNOME и KDE Plasma может работать немного медленнее.

1.6.2.2 Отключение пользовательских служб GNOME/Cinnamon

GSD (gnome-settings-daemon) - это, как следует из названия, службы настройки GNOME и связанных приложений. Если отойти от строгого определения, то это просто службы-настройки на все случаи жизни, которые просто висят у вас в оперативной памяти в ожидании когда вам, или другому приложению, к примеру, понадобиться настроить или интегрировать поддержку планшета Wacom в рабочее окружение, или для уведомления вас о различных событиях, таких как недостаточное место на диске или начале печати, а также для применения изменений совершенных в настройках GNOME на лету. Так как другое рабочее окружение - Cinnamon, является форком GNOME 3, то оно также имеет собственные службы настройки, называемые CSD службами, и большая часть из них являются "близницами" тех служб, которые есть в GNOME, поэтому их функционал во многом совпадает. Все команды по отключению служб с одинаковым назначением в обоих окружения будут продублированы.

Служба интеграции рабочего окружения с графическим планшетом Wacom. Позволяет настраивать яркость планшета средствами окружения (GNOME или Cinnamon). Если у вас такого нет - смело отключайте:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.Wacom.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-wacom.desktop ~/.config/autostart
echo "Hidden=true" >> ~/.config/autostart/cinnamon-settings-daemon-wacom.desktop
```

Служба уведомления о начале печати. Если нет принтера или вам просто не нужны эти постоянные уведомления - отключаем:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.PrintNotifications.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-print-notifications.desktop ~/.
↪config/autostart
echo "Hidden=true" >> ~/.config/autostart/cinnamon-settings-daemon-print-
↪notifications.desktop
```

Службы управления цветовыми профилями дисплея и принтеров. Если вы отключите данную службу, то не будет работать тёплый режим экрана (Системный аналог Redshift):

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.Color.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-color.desktop ~/.config/autostart
echo "Hidden=true" >> ~/.config/autostart/cinnamon-settings-daemon-color.desktop
```

Отключение службы управления специальными возможностями системы:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.A11ySettings.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-a11y-*.desktop ~/.config/autostart
echo "Hidden=true" >> ~/.config/autostart/cinnamon-settings-daemon-a11y-*.desktop
```

Внимание

Не отключать данную службу людям с ограниченными возможностями (инвалидам)!

Службы управления беспроводными интернет-подключениями и Bluetooth. Не рекомендуется отключать для ноутбуков с активным использованием Wi-Fi и Bluetooth, но если вы используете настольный ПК без использования беспроводных технологий, то смело отключайте:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.Wwan.service
systemctl --user mask org.gnome.SettingsDaemon.Rfkill.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-rfkill.desktop ~/.config/autostart
echo "Hidden=true" >> ~/.config/autostart/cinnamon-settings-daemon-rfkill.desktop
```

Отключение службы защиты от неавторизованных USB устройств при блокировке экрана:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.UsbProtection.service
```

Примечание

Данная служба может быть полезна если у вас ноутбук и вы часто посещаете вместе ним общественные места.

Службу для автоматической блокировки экрана. Можете отключить по собственному желанию:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.ScreensaverProxy.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-screensaver-proxy.desktop ~/.config/
↪autostart
echo "Hidden=true" >> ~/.config/autostart/cinnamon-settings-daemon-screensaver-proxy.
↪desktop
```

Служба для автоматического управления общим доступом к файлам и директориям. Если никогда не пользовались, можете отключить:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.Sharing.service
```

Примечание

Данная служба есть только в окружении GNOME.

Служба интеграции рабочего окружения с карт-ридером. Если у вас карт-ридера нет, то смело отключайте:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.Smartcard.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-smartcard.desktop ~/.config/
↪autostart
echo "Hidden=true" >> cinnamon-settings-daemon-smartcard.desktop
```

Служба автоматического оповещения вас о недостаточном количестве свободного места на диске. Если вы делаете это самостоятельно при помощи специальных средств, как например Baobab, то можно отключить данную службу:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.Housekeeping.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-housekeeping.desktop ~/.config/
↪autostart
echo "Hidden=true" >> cinnamon-settings-daemon-housekeeping.desktop
```

Служба управления питанием и функциями энергосбережения. Рекомендуется оставить эту службу включенной если у вас ноутбук, т. к. без неё не будет работать регулирование яркости средствами рабочего окружения и управление сном, но можете отключить, если у вас настольный ПК:

GNOME

```
systemctl --user mask org.gnome.SettingsDaemon.Power.service
```

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-power.desktop ~/.config/autostart
echo "Hidden=true" >> cinnamon-settings-daemon-power.desktop
```

Служба интеграции работы буфера обмена с Cinnamon. Если вы никогда не пользовались виджетом истории буфера обмена в трее, то можете отключить данную службу:

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-clipboard.desktop ~/.config/
↪autostart
echo "Hidden=true" >> cinnamon-settings-daemon-clipboard.desktop
```

Примечание

Данная служба есть только в окружении Cinnamon.

Служба для автоматического подстраивания интерфейса Cinnamon при повороте дисплея. Если у вас нет сенсорного экрана или поддержки переворота дисплея - смело отключайте:

Cinnamon

```
cp -v /etc/xdg/autostart/cinnamon-settings-daemon-orientation.desktop ~/.config/
↪autostart
echo "Hidden=true" >> cinnamon-settings-daemon-orientation.desktop
```

Примечание

Данная служба есть только в окружении Cinnamon.

Если после отключения какой-либо из вышеперечисленных служб что-то пошло не так, или просто какую-либо из них понадобилось снова включить, то выполните следующую команду в зависимости от используемого рабочего окружения предварительно подставив имя в неё нужной службы:

GNOME

```
systemctl --user unmask --now СЛУЖБА
```

Cinnamon

```
rm ~/.config/autostart/cinnamon-settings-daemon-СЛУЖБА.desktop
```

Служба вернется в строй после перезагрузки рабочего окружения.

1.6.2.3 Отключение ненужных служб Plasma

По аналогии с GNOME, у Plasma тоже есть свои службы настройки, которые хоть и гораздо менее требовательны к ресурсам. Тем не менее, это по прежнему солянка из различных процессов, которые вам далеко не всегда пригодятся, а отключая ненужные из них вы можете чуть снизить потребление оперативной памяти вашей оболочкой, т.к. по умолчанию все службы включены.

Настройка служб происходит в графических настройках Plasma, в разделе "*Запуск и завершение*" -> *"Управление службами"*

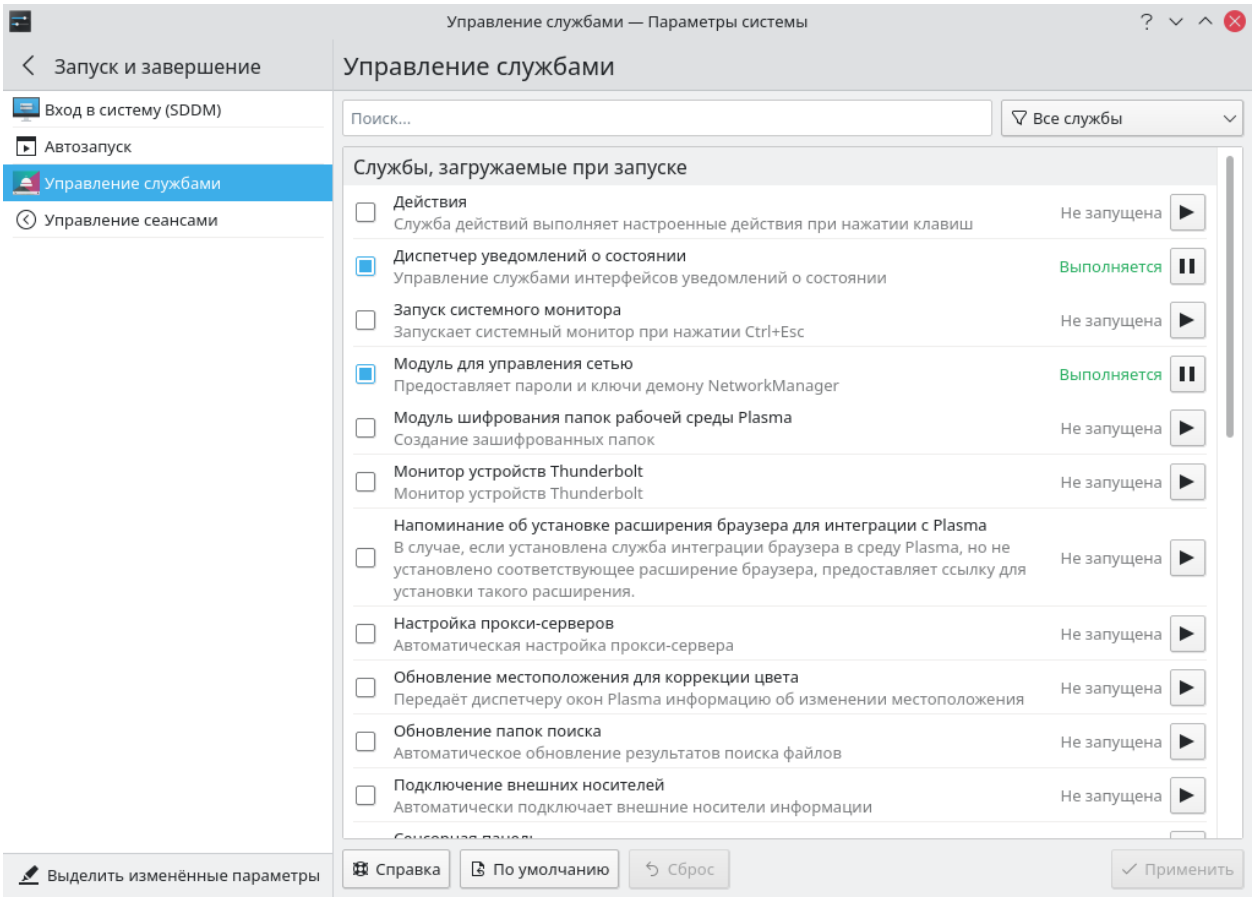

Управление службами — Параметры системы
Запуск и завершение
Управление службами
Вход в систему (SDDM)
Автозапуск
Управление службами
Управление сеансами
Поиск...
Все службы
Службы, загружаемые при запуске
Действия
Служба действий выполняет настроенные действия при нажатии клавиш
Не запущена
Диспетчер уведомлений о состоянии
Управление службами интерфейсов уведомлений о состоянии
Выполняется
Запуск системного монитора
Запускает системный монитор при нажатии Ctrl+Esc
Не запущена
Модуль для управления сетью
Предоставляет пароли и ключи демону NetworkManager
Выполняется
Модуль шифрования папок рабочей среды Plasma
Создание зашифрованных папок
Не запущена
Монитор устройств Thunderbolt
Монитор устройств Thunderbolt
Не запущена
Напоминание об установке расширения браузера для интеграции с Plasma
В случае, если установлена служба интеграции браузера в среду Plasma, но не установлено соответствующее расширение браузера, предоставляет ссылку для установки такого расширения.
Не запущена
Настройка прокси-серверов
Автоматическая настройка прокси-сервера
Не запущена
Обновление местоположения для коррекции цвета
Передаёт диспетчеру окон Plasma информацию об изменении местоположения
Не запущена
Обновление папок поиска
Автоматическое обновление результатов поиска файлов
Не запущена
Подключение внешних носителей
Автоматически подключает внешние носители информации
Не запущена
Выделить изменённые параметры
Справка
По умолчанию
Сброс
Применить

Таблица 2: Список служб рекомендуемых к отключению

Название службы	Описание
Запуск системного монитора	Отслеживает нажатие клавиш Ctrl+Esc для запуска системного монитора. Не думаю, кто-либо активно этим пользовался ранее, поэтому лучше отключить.
Напоминание об установке расширения браузера	Довольно бесполезная служба, отключаем.
Bluetooth	Отключайте, если у вас нет модуля Bluetooth или вы им не пользуйтесь (Данный пункт может отсутствовать если не установлен пакет `bluedevil`).
Настройки прокси-серверов	Отключайте, если не используете системный прокси/VPN.
Учётный записи	Нужна только если у вас больше одной учетной записи на компьютере.
Сенсорная панель	Отключаем, если сенсорная панель отсутствует или вы ей не пользуетесь.
Обновление местоположения для коррекции цвета	Служба, которая автоматически корректирует "температуру" теплого режима экрана в зависимости от вашего местоположения. Отключайте, если не используете теплый режим или не желаете раскрывать собственное местоположение.
Модуль шифрования папок рабочей среды Plasma	Отключайте, если вы не используете шифрование для отдельных файлов или имеете уже шифрование для всей системы.
Слежение за изменениями в URL	Работает только в сетевых папках просматриваемых через Dolphin. Если вы не часто используете сетевые диски или сервисы, то рекомендуется отключить.
Слежение за свободным местом на диске	Автоматически оповещает вас о недостаточном количестве свободного места на диске. Если вы делаете это самостоятельно при помощи специальных виджетов, то можно отключить.
SMART	Автоматически отслеживает состояние вашего SSD носителя. Довольно полезная служба, но если вы предпочитаете делать это самостоятельно, то можете отключить.
Действия	Обеспечивает действий назначенных пользователем в настройках Dolphin/других приложения. Если вы их не используете, то можете отключить.
Модуль для управления сетью	Добавляет системный лоток виджет для управления сетевыми подключениями. Отключайте, если не используете NetworkManager.
Состояние сети	Оповещает пользователя в случае неработоспособности интернет-соединения. Так как понять это можно будет и по косвенным признакам, то службу можно отключать.
Служба синхронизации параметров GNOME/GTK	Осуществляет смену темы применяемой для приложений GTK на лету. Если отключить, то смена GTK темы будет применяться только после перезагрузки оболочки.
Обновление папок поиска	Автоматически обновляет результаты поиска файлов. Отключаем на свое усмотрение. Кроме того, судя по всему работает только в Dolphin.
Служба локальных сообщений	Формирует уведомления для сообщений передаваемых между терминалами через команды wall и write. В настоящий момент это очень редко используется и может быть нужно только на многопользовательских системах, поэтому можно смело отключать.
Подключение внешних носителей	Автоматически примонтирует внешние устройства при их подключении. Например, такие как USB-флешки. Отключайте на свое усмотрение, но в целях безопасности рекомендуется отключить.
Часовой пояс	Информирует другие приложения об изменении системного часового пояса. Довольно редко применимо, можно отключить.
Фоновая служба меню приложений	Немного странная служба. По своему назначению она осуществляет обновление "Меню Приложений" при появлении новых ярлыков, однако даже при её отключении этот функционал работает. Отключайте на свое усмотрение.

1.7 Оптимизация пакетов

1.7.1 Настройка makepkg.conf

Прежде чем приступать к сборке пакетов, мы должны изменить так называемые флаги компиляции, что являются указателями для компилятора, какие инструкции и оптимизации использовать при сборке программ.

Для этого создадим пользовательский конфиг `~/.makepkg.conf` в домашней директории, чтобы переопределить системные настройки:

Список 18: `nano ~/.makepkg.conf`

```
CFLAGS="-march=native -mtune=native -O2 -pipe -fno-plt -fexceptions \
      -Wp,-D_FORTIFY_SOURCE=3 -Wformat -Werror=format-security \
      -fstack-clash-protection -fcf-protection"
CXXFLAGS="$CFLAGS -Wp,-D_GLIBCXX_ASSERTIONS"
RUSTFLAGS="-C opt-level=3 -C target-cpu=native -C link-arg=-z -C link-arg=pack-
↪relative-relocs"
MAKEFLAGS="-j$(nproc) -l$(nproc)"
```

Примечание

Где "-O2" - **Это не нуль/ноль**

Данные флаги компилятора выжимают максимум производительности при компиляции, но могут вызывать ошибки сборки в очень редких приложениях. Если такое случится, то отключите параметр 'lto' в строке `options` добавив перед ним символ восклицательного знака ! (*"!lto"*).

1.7.1.1 Использование tmpfs для сборки в ОЗУ

Во время сборки программ компилируются множество временных промежуточных файлов и записываются на диск (HDD/SSD) для последующей компоновки в исполняемый файл или библиотеку. Для ускорения процесса сборки пакетов можно использовать вместо HDD/SSD - оперативную память, а точнее *tmpfs*. Поскольку ОЗУ значительно быстрее любого HDD или SSD, то сборка происходит быстрее. Помимо этого уменьшается нагрузка на систему ввода-вывода, и как следствие меньше изнашивается диск. Использовать tmpfs для makepkg можно несколькими способами:

- Непосредственно указывать переменную перед сборкой:

```
BUILDDIR=/tmp/makepkg makepkg -sric
```

- Для сборки всего - задать параметр (раскомментировать в файле `/etc/makepkg.conf`) `BUILDDIR` для использования директории `/tmp`:

```
BUILDDIR=/tmp/makepkg
```

- Создать отдельную директорию *tmpfs* заданного размера:
  - Необходимо добавить в `/etc/fstab` директорию для монтирования *tmpfs*, указав путь и максимальный объём директории, которая может расширяться при работе *tmpfs* (учтите что *tmpfs* использует ОЗУ, поэтому внимательно подходите к вопросу выделяемого объема, он не должен превышать общий объем доступной памяти, несмотря на то, что изначально *tmpfs* ничего не занимает), например:

```
tmpfs   /var/tmp/makepkg         tmpfs   rw,nodev,nosuid,size=16G          0 ␣
↪0
```

– Далее, как и в предыдущем случае, указать `BUILDDIR` в `/etc/makepkg.conf`, но уже с путем к директории указанной в *fstab*:

```
BUILDDIR=/var/tmp/makepkg
```

Внимание

На системах с небольшим количеством ОЗУ (например 4 ГБ и менее) *tmpfs* может негативно сказаться на сборке тяжёлых пакетов, что может привести к недостатку ОЗУ для сборки.

Примечание

Можно указать параметр PKGDEST для определения директории собранного пакета.

Количество доступного и используемого места в *tmpfs* можно посмотреть:

```
df -h | grep tmpfs
```

1.7.1.2 Включение ccache

В Linux системах есть не так много программ, сборка которых может занять больше двух часов, но они все таки есть. Потому, было бы неплохо ускорить повторную компиляцию таких программ как Wine/Proton-GE и т.д.

ccache - это кэш для компиляторов C/C++, в частности совместимый с компиляторами GCC/Clang, цель которого состоит в ускорении повторного процесса компиляции одного и того же кода. Это значит, что если при сборке программы новой версии, будут замечены полностью идентичные блоки исходного кода в сравнении с его старой версией, то компиляция этих исходных текстов производиться не будет. Вместо этого, уже готовый, скомпилированный код старой версии будет вынут из кэша ccache. За счёт этого и достигается многократное ускорение процесса компиляции.

Установка

```
sudo pacman -S ccache
```

После установки его ещё нужно активировать в ваших настройках makepkg. Для этого отредактируем конфигурационный файл

Список 19: `nano ~/.makepkg.conf`

```
BUILDENV=(!distcc color ccache check !sign)
```

После этого повторная пересборка желаемых программ и их обновление должны значительно ускориться.

Внимание

ccache может ломать сборку некоторых программ, поэтому будьте осторожны с его применением.

1.7.2 Форсирование использования Clang при сборке пакетов

В системах на базе ядра Linux различают две основных группы компиляторов, это LLVM и GCC. И те, и другие хорошо справляются с возложенными на них задачами, но LLVM имеет чуть большее преимущество с точки зрения производительности при меньших потерях в качестве конечного кода. Поэтому в целом применение компиляторов LLVM для сборки различных пакетов при задании флага -O3 (максимальная производительность) является совершенно оправданным, и может дать реальный прирост при работе программ.

Компилятором для языков C/C++ в составе LLVM является Clang и Clang++ соответственно. Его использование при сборке пакетов мы и будем форсировать через makepkg.conf

Для начала выполним их установку:

```
sudo pacman -Syu llvm clang lld mold openmp
```

Теперь клонируем уже готовый конфигурационный файл `/etc/makepkg.conf` под новыми именем в домашнюю директорию `~/.makepkg-clang.conf`:

```
cp /etc/makepkg.conf ~/.makepkg-clang.conf
```

Это поможет нам в случае чего откатиться к использованию компиляторов GCC если возникнут проблемы со сборкой пакетов через LLVM/Clang.

Теперь откроем выше скопированный файл и добавим туда после строки `CHOST="x86_64-pc-linux-gnu"` следующее:

```
export CC=clang
export CXX=clang++
export AR=llvm-ar
export NM=llvm-nm
export STRIP=llvm-strip
export OBJCOPY=llvm-objcopy
export OBJDUMP=llvm-objdump
export READELF=llvm-readelf
export RANLIB=llvm-ranlib
export HOSTCC=clang
export HOSTCXX=clang++
export HOSTAR=llvm-ar
```

При использовании Clang из пакета *llvm-git* (установка описана ниже) стоит установить компоновщик mold, а также другие флаги при сборке пакетов:

```
CFLAGS="-march=native -mtune=native -O3 -falign-functions=32 -fno-math-errno \
        -fno-trapping-math -Wp,-D_FORTIFY_SOURCE=2 -Wformat -Werror=format-security \
        -fstack-clash-protection"
CXXFLAGS="$CFLAGS -Wp,-D_GLIBCXX_ASSERTIONS"
LDFLAGS="-fuse-ld=mold -Wl,-O1 -Wl,--sort-common -Wl,--as-needed -Wl,-z,relro -Wl,-z,
↪now \
         -Wl,-z,pack-relative-relocs"
LTOFLAGS="-flto=auto"
RUSTFLAGS="-C opt-level=3 -C target-cpu=native -C link-arg=-z -C link-arg=pack-
↪relative-relocs \
           -C link-arg=-fuse-ld=mold"
#-- Make Flags: change this for DistCC/SMP systems
MAKEFLAGS="-j$(nproc)"
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
NINJAFLAGS="-j$(nproc)"
OPTIONS=(strip docs !libtool !staticlibs emptydirs zipman purge !debug lto)
```

Отлично, теперь вы можете собрать нужные вам пакеты (программы) через LLVM/Clang просто добавив к уже известной команде makepkg следующие параметры:

```
makepkg --config ~/.makepkg-clang.conf -sric --skippgpcheck --skipchecksums
```

Внимание

Далеко не все пакеты так уж гладко собираются через Clang, в частности не пытайтесь собирать им Wine/DXVK, т.к. это официально не поддерживается и с 98% вероятностью приведет к ошибке сборки. Но в случае неудачи вы всегда можете использовать компиляторы GCC, которые у вас заданы в настройках makepkg.conf по умолчанию, т.е. просто уберите опцию `--config ~/.makepkg-clang.conf` из команды `makepkg`.

Мы рекомендуем вам пересобрать наиболее важные пакеты. Например такие как драйвера (то есть mesa, lib32-mesa, если у вас Intel/AMD), Xorg сервер, а также связанные с ним компоненты, или Wayland, критически важные пакеты вашего DE/WM, например: gnome-shell, plasma-desktop. А также композиторы kwin, mutter, picom и т.д. в зависимости от того, чем именно вы пользуетесь.

Больше подробностей по теме вы можете найти в данной статье:

https://habr.com/ru/company/ruvds/blog/561286/

1.7.3 Установка оптимизированных пакетов

Нативная компиляция - это конечно хорошо, но не у каждого человека есть время заниматься подобными вещами, да и всю систему пересобирать с нативными флагами тоже никто не будет (иначе вам сюда: https://gentoo.org). Возникает вопрос: как сделать все с наименьшим количеством напряга?

Для начала сделаем небольшое отступление. У архитектуры *x86_64* различают несколько поколений или "уровней". Это *x86_64*, *x86_64_v2*, *x86_64_v3* и *x86_64_v4* (новейшие процессоры). Различия между этими "поколениями" состоят в применяемом наборе инструкций и возможностей процессора. Например, если вы собираете программу для x86_64_v2, то вы автоматически задействуете инструкции SSE3, SSE4_1 и т.д. При этом такая программа не будет работать на предыдущих поколениях, то есть на процессорах которые не поддерживают набор инструкций *x86_64_v2*. При этом к *x86_64_v2* и другим уровням относятся различные процессоры, как AMD, так и Intel. При этом логично, что чем выше поколение x86_64 поддерживает ваш процессор, тем больше будет производительность за счет использования многих оптимизаций и доп. инструкций. Подробнее об этих уровнях или же поколениях можете прочитать здесь (англ.).

Смысл в том, что существует сторонний репозиторий Arch Linux - ALHP, который содержит **все пакеты** из официальных репозиториев, но собранных для процессоров x86_64_v2 или x86_64_v3. То есть это те же самые, уже готовые пакеты из официальных репозиториев, но собранные с различными оптимизациями для определенной группы процессоров (поколений x86_64).

Опасно

Прежде чем мы подключим данный репозиторий, нужно **обязательно** понять к какому поколению относиться ваш процессор, иначе, если вы установите пакеты собранные для x86_64_v3, но *ваш процессор при этом не будет относиться к поколению x86_64_v3*, то *ваша система станет полностью не работоспособной*, хотя её и все ещё можно будет восстановить через LiveCD окружение при помощи pacstrap.

Опасно

Оптимизированные пакеты для процессоров Intel поддерживают только полные процессоры серий Core 2 и i3/i5/i7. Многие процессоры Pentium/Celeron не имеют полного набора инструкций, необходимого для использования оптимизированных пакетов. Пользователям этих процессоров следует установить универсальные пакеты или пакеты оптимизированные ниже на один уровень (то есть если у вас поддерживается v3, то подключайте репозиторий с v2 и т.д.), даже если GCC возвращает значение, соответствующее полному набору флагов Core i3/i5/i7, например, Haswell.

Проверить поколение вашего процессора можно следующей командой:

```
/lib/ld-linux-x86-64.so.2 --help | grep -B 3 -E "x86-64-v2"
```

После каждого поколения будет написано, поддерживается оно вашим процессором или нет. Например:

```
Subdirectories of glibc-hwcaps directories, in priority order:
x86-64-v4
x86-64-v3
x86-64-v2 (supported, searched)
```

Если у вас поддерживается хотя бы x86_64_v2, то вы так же сможете использовать данный репозиторий, ибо он предоставляет пакеты как для x86_64_v2, так и для x86_64_v3. **Главное не перепутаете, какое именно у вас поколение**.

Чтобы подключить репозиторий установим ключи для проверки подписей пакетов:

```
# Ключи для пакетов
git clone https://aur.archlinux.org/alhp-keyring.git
cd alhp-keyring
makepkg -sric --skippgpcheck
```

А также список зеркал:

```
git clone https://aur.archlinux.org/alhp-mirrorlist.git
cd alhp-mirrorlist
makepkg -sric
```

После этого нужно отредактировать конфиг pacman добавив репозиторий для нужной архитектуры (`sudo nano /etc/pacman.conf`).

Итак, **если ваш процессор поддерживает только x86_64_v2** (как допустим у автора), то пишем следующее:

```
[core-x86-64-v2]
Include = /etc/pacman.d/alhp-mirrorlist

[extra-x86-64-v2]
Include = /etc/pacman.d/alhp-mirrorlist

[multilib-x86-64-v2]
Include = /etc/pacman.d/alhp-mirrorlist

[core]
Include = /etc/pacman.d/mirrorlist
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
[extra]
Include = /etc/pacman.d/mirrorlist

[multilib]
Include = /etc/pacman.d/mirrorlist
```

Если же у вас процессор поддерживает x86_64_v3, то пишем следующее:

```
[core-x86-64-v3]
Include = /etc/pacman.d/alhp-mirrorlist

[extra-x86-64-v3]
Include = /etc/pacman.d/alhp-mirrorlist

[multilib-x86-64-v3]
Include = /etc/pacman.d/alhp-mirrorlist

[core]
Include = /etc/pacman.d/mirrorlist

[extra]
Include = /etc/pacman.d/mirrorlist

[multilib]
Include = /etc/pacman.d/mirrorlist
```

После этого выполняем полное обновление системы:

```
sudo pacman -Syyuu
```

Перезагружаемся и наслаждаемся результатом (если вы все сделали правильно).

1.8 Низкие задержки звука

1.8.1 PipeWire

PipeWire - это новая альтернатива PulseAudio, которая призвана избавить от проблем своего предшественника, уменьшить задержки звука, снизить потребление памяти и улучшить безопасность. PipeWire поставляется по умолчанию в Arch Linux в качестве звукового сервера, требуемого для клиентов libpulse[1], так что никаких команд для его установки прописывать не нужно. Однако несмотря на это рекомендуется установить дополнительные компоненты и явно включить пользовательские службы для уменьшения их задержки запуска через активацию путем сокетов, которая используется по умолчанию для запуска PipeWire:

```
sudo pacman -S pipewire-jack lib32-pipewire gst-plugin-pipewire
systemctl --user enable --now pipewire pipewire-pulse wireplumber
```

Примечание

Пакет `lib32-pipewire` нужен для правильной работы звука в 32-битных играх запускаемых через Wine или Proton, а также в нативных портах игр под Linux.

[1] https://gitlab.archlinux.org/archlinux/packaging/packages/pipewire/-/commit/14614b08f6f8cf8e50b4cbb78a141e82066e7f80

Для непосредственно уменьшения самих задержек установим дополнительный пакет `realtime-privileges` и добавим пользователя в группу `realtime`:

```
sudo pacman -S realtime-privileges rtkit
sudo usermod -aG realtime "$USER"
```

Дополнительно советуем установить реализацию Jack API. См. раздел ниже.

1.8.1.1 Настройка PipeWire

Несмотря на то, что настройки по умолчанию могут работать достаточно хорошо для большинства оборудования, имеет смысл выполнить дополнительную настройку для улучшения качества звука (особенно если вы являетесь обладателем ЦАП или полноценной звуковой карты).

Сначала создадим пути для хранения конфигурационных файлов в домашней директории:

```
mkdir -p ~/.config/pipewire/pipewire.conf.d
```

В появившейся директории создадим файл со следующим содержанием:

Список 20: `nano ~/.config/pipewire/pipewire.conf.d/20-no-resampling.conf`

```
context.properties = {
  default.clock.rate = 48000
  default.clock.allowed-rates = [ 44100 48000 96000 192000 ]
}
```

Фактически здесь мы настраиваем две вещи: первое, это частота дискретизации, используемая по умолчанию (`defalut.clock.rate`), в зависимости от которой PipeWire так же считает оптимальные задержки для вывода звука, а именно так называемые значения `quantum`, о которых будет рассказано в следующем разделе. Во-вторых, мы явно перечисляем все доступные частоты дискретизации (`defalut.clock.allowed-rates`), поддерживаемые нашим устройством вывода звука (ЦАП или встроенная аудиокарта). Это нужно для того, чтобы PipeWire не делал ресемплирования, то есть изменения частоты дискретизации исходного аудиопотока к частоте используемой по умолчанию, что может повлечь за собой ухудшение качества итогового звучания, а также дополнительные накладные расходы в виде нагрузки на CPU.

Важно отметить, что PipeWire выполняет переключение между указанными частотами дискретизации только в состоянии покоя непосредственно перед началом вывода нового аудиопотока, однако если вы начинаете проигрывать ещё один аудиопоток, то PipeWire не поменяет частоту дискретизации с учетом второго аудиопотока, а продолжит использовать ту же частоту дискретизации, что и у первого, и это может опять же привести к ситуации ресемплинга второго аудиопотока до уровня первого. В случае же если первоначальный аудиопоток не попадал в указанный диапозон частот, то он также бы ресемплировался от целовой к самой "ближайшей" частоте из перечисленных.

Чтобы узнать весь диапозон частот доступных для вашего устройства, следует использовать данную команду:

Звуковой чип

```
cat /proc/asound/card0/codec\#0 | grep -A 8 "Audio Output" -m 1 | grep rates
```

ЦАП

```
cat /proc/asound/card0/stream0 | grep Rates | uniq
```

Частоты, которые были получены таким образом, нужно прописать через пробел взамен тех, что даны в примере выше. Если доступно несколько устройств, то при помощи команды `cat /proc/asound/cards` узнайте номер звуковой карты, которая используется непосредственно для вывода звука, и подставьте его в команду выше.

1.8.1.1.1 Микширование стерео в 5.1

PipeWire так же как и PulseAuido позволяет микшировать звук в 5.1. Эта возможность отключена по умолчанию, но для неё существует заранее подготовленный конфигурационный файл, который нам нужно просто перенести в домашнюю директорию:

```
mkdir -p ~/.config/pipewire/pipewire-pulse.conf.d ~/.config/pipewire/client-rt.conf.d
cp /usr/share/pipewire/client-rt.conf.avail/20-upmix.conf ~/.config/pipewire/pipewire-
↪pulse.conf.d
cp /usr/share/pipewire/client-rt.conf.avail/20-upmix.conf ~/.config/pipewire/client-
↪rt.conf.d
```

1.8.1.1.2 Исправление хрипов под нагрузкой

Некоторые пользователи после перехода на PipeWire могут столкнуться с появлением "хрипов" во время произведения звука, если система находится под высокой нагрузкой (например, фоновой компиляцией или во время игры). Это происходит потому, что PipeWire старается осуществлять вывод с звука с наименьшими задержками, что сложно гарантировать когда система нагружена даже с установленными `realtime-privileges`.

Для их исправления создадим новый конфигурационный файл и изменим следующие значения для размера буфера по умолчанию:

Список 21: `nano ~/.config/pipewire/pipewire.conf.d/10-sound.conf`

```
context.properties = {
  default.clock.min-quantum = 512
  default.clock.quantum = 4096
  default.clock.max-quantum = 8192
}
```

Если после этого проблема осталась, то рекомендуется умножить значение `min-quantum` на два и проверить снова до тех пор пока проблема не перестанет воспроизводится.

1.8.1.2 JACK

JACK - это ещё один альтернативный звуковой сервер с одноименной библиотекой API, цель которого состоит выведение звука с минимальной задержкой. Несмотря на неплохую поддержку JACK со стороны многих приложений, автор рекомендует по прежнему использовать звуковой сервер PipeWire, так как актуальная реализация JACK - jack2, практически не получает серьёзных улучшений и находится скорее в состоянии сопровождения.

Для правильной работы приложений использующих JACK в качестве библиотеки API лучше всего установить пакет `pipewire-jack`, который предоставляет совместимость между такими приложениями со звуковым сервером PipeWire, что позволяет всем изменениям проделенным выше для его настройки распространяться и на приложения, использующие JACK.

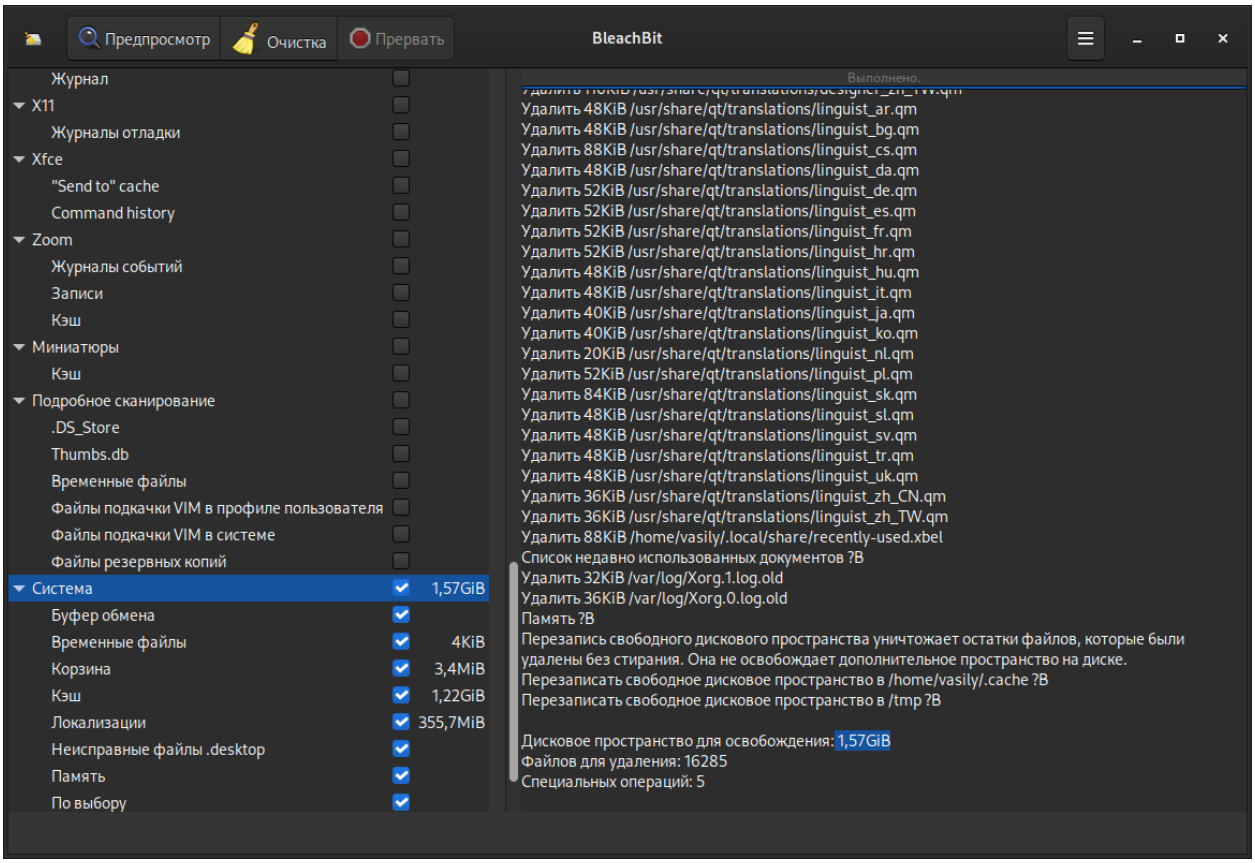
1.9 Профилактика диска

Рациональное использование пространства на диске также входит в перечень задач по оптимизации системы. Хотя это и не влияет напрямую на производительность самой системы, это позволяет всё время иметь пространство на носителе, чтобы использовать его для хранения новой информации, поэтому в данном разделе затрагивается тема профилактики носителя или же его чистки от "мусора": различных временных данных, кэшей, баз данных.

1.9.1 Чистка при помощи Bleachbit

Аналог CCleaner для Linux, помогает выполнить очистку системы от накопившегося в ней мусора.

Советуем выполнять чистку системы уже после проведения всех оптимизаций.



Установка + дополнительные фильтры:

```
sudo pacman -S bleachbit

# Дополнительные фильтры

git clone https://aur.archlinux.org/cleanerml-git.git # Загрузка исходников.
cd cleanerml-git                                      # Переход в cleanerm.
makepkg -sric                                         # Сборка и установка.
```

1.9.2 Автоматическая очистка кэша pacman

Кэш пакетов pacman имеет плохое свойство забиваться и со временем занимает много места на диске. Чтобы этого не происходило, создадим небольшой демон, который будет автоматически его очищать, например, каждую неделю. В этом нам могут встроенные средства systemd для создания таймеров - специальных служб, которые устанавливают периодичность выполнения того или иного события, например, запуска другой службы (в нашем случае службы очистки кэша). Напишем таймер, выполняющий команду `pacman -Scc` регулярно раз в неделю с периодом проверки времени один раз в час. Для этого сначала создадим службу, которая будет регулярно выполняться, назовем её `pacman-cleaner.service`:

Список 22: `/etc/systemd/system/pacman-cleaner.service`

```
[Unit]
Description=Cleans pacman cache

[Service]
Type=oneshot
ExecStart=/usr/bin/pacman -Scc --noconfirm

[Install]
WantedBy=multi-user.target
```

И для этой службы создадим соответствующий таймер, который будет активировать её выполенение каждую неделю:

Список 23: `/etc/systemd/system/pacman-cleaner.timer`

```
[Unit]
Description=Run clean of pacman cache every week

[Timer]
OnCalendar=weekly
AccuracySec=1h
Persistent=true

[Install]
WantedBy=timers.target
```

Не забываем включить этот самый таймер:

```
sudo systemctl enable --now pacman-cleaner.timer
```

1.9.3 Оптимизация баз данных SQLite

Базы данных типа SQLite часто используется для локального хранения с целью кэширования тех или иных данных. Например, Firefox использует SQLite базу внутри текущего профиля для хранения всех пиктограм ранее посещаемых сайтов. Базы такого типа поддаются оптимизации занимаемого места на диске через специальную операцию `VACUUM`.

Для профаликтики диска и экономии места вы можете захотеть периодически выполнять данную операцию над всеми базами данных в вашей домашней директории при помощи следующей команды:

```
find ~/ -type f -regextype posix-egrep -regex '.*\.(db|sqlite)' \
  -exec bash -c '[ "$(file -b --mime-type {})" = "application/vnd.sqlite3" ] &&␣
↪sqlite3 {} "VACUUM; REINDEX;"' \; 2>/dev/null
```

Предупреждение

Перед запуском данной команды рекомендуется закрыть все приложения, так как операция `VACUUM` не может быть выполнена для открытых и используемых в данный момент баз данных.

Данную команду рекомендуется периодически выполнять вручную или при помощи systemd-таймера по аналогии с очисткой кэша pacman как было показано выше.

1.10 Разгон монитора

Опасно

Инструкции в данном разделе можно выполнять по желанию, цель автора лишь показать саму возможность разгона в Linux, а не призвать всех это делать.

В этом разделе мы опишем новый способ разгона монитора, который является более универсальным чем предыдущий через различные манипуляции с конфигом Xorg.

1.10.1 Подготовка

Суть способа состоит в редактировании EDID файла вашего монитора, что позволяет этому способу работать на любой конфигурации.

Для начала найдем нужный нам EDID файл через команду:

```
find /sys/devices/pci*/ -name edid
```

Команда выведет список EDID файлов для различных типов подключения, вы должны выбрать нужный вам и скопировать его в домашнюю директорию. Например мне нужен EDID файл для моего монитора который подключен по HDMI, значит:

```
cp -r /sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0/drm/card0/card0-HDMI-A-1/edid␣
↪~/
```

Отлично, теперь выполним установку редактора EDID. В нашем случае это будет нативный wxedid из AUR, но вы можете воспользоваться любым другим.

```
git clone https://aur.archlinux.org/wxedid.git # Стянуть исходники ПО
cd wxedid                                      # Переходим в директорию
makepkg -sric                                  # Сборка и установка
```

Откроем редактор через меню или команду:

```
wxedid
```

1.10.2 Использование редактора wxedid

После запуска редактора в контекстном меню выбираем **File -> Open EDID binary** для редактирования нашего EDID файла в домашней директории.

Теперь, для активации полного цветового диапазона меняем данные в строках:

1) SPF: Supported features -> изменить значение vsig_format на 0b00

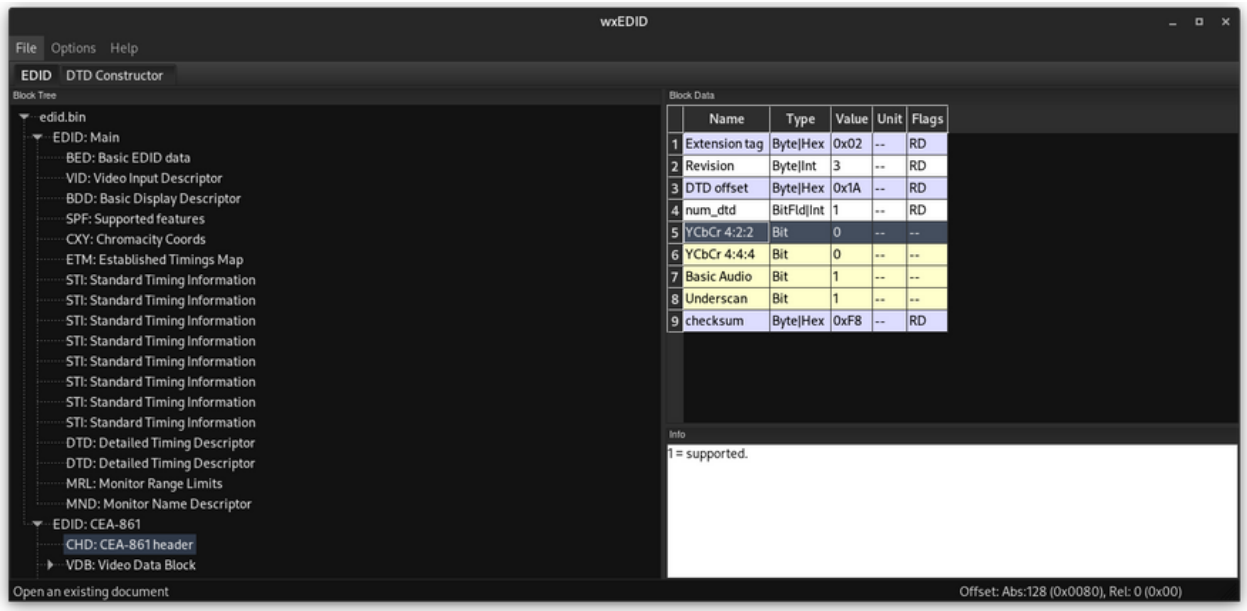
2) CHD: CEA-861 header -> изменить значение YCbCr4:2:2 и YCbCr4:4:4 на 0

3) VSD: Vendor Specific Data Block -> изменить значение DC_Y444 на 0

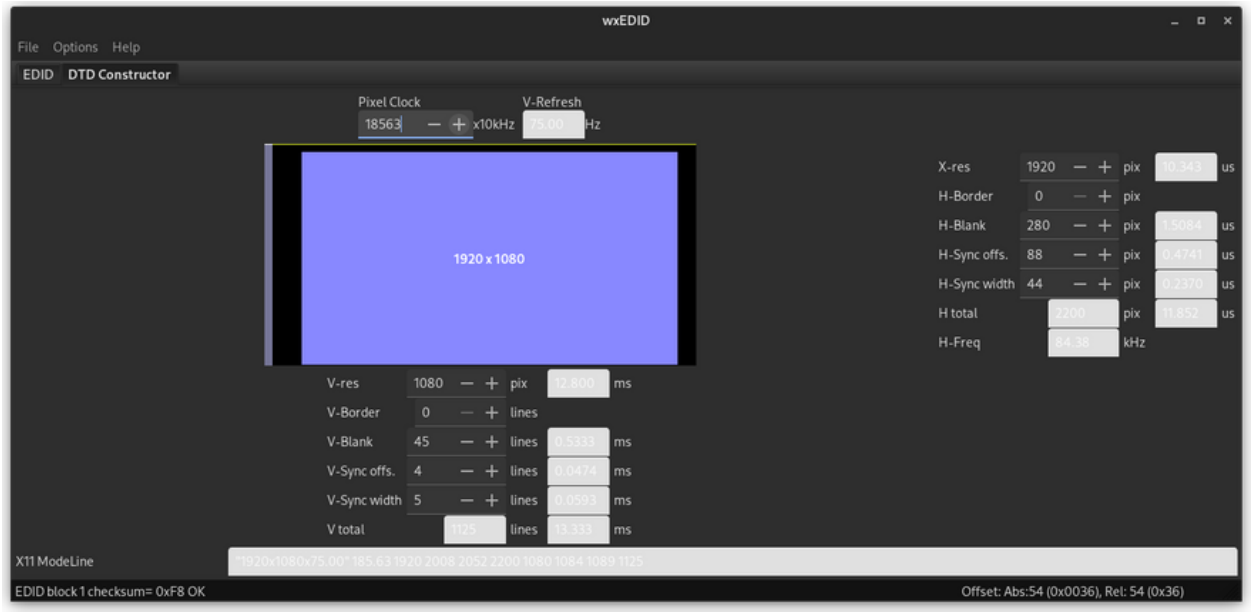
Это необходимо чтобы исправить давнюю проблему с неверно выставляемым в Linux цветовым диапозоном вашего монитора.

Предупреждение

У Mutter начиная с версии GNOME 45 появилась поддержка форматов YUV (https://gitlab.gnome.org/GNOME/mutter/-/merge_requests/2191), поэтому пользователи этого окружения могут пропустить данные шаги.



Для разгона же вам нужно выбрать *DTD: Detailed Timing Descriptor*. У вас их может быть несколько, т.к. каждый из них работает для отдельного разрешения монитора. Вам нужно выбрать тот, у которого самое большое разрешение. Вы это поймете по строчкам *H-Active pix* и *V-Active lines*. После этого перейдите во вкладку *DTD Constructor* и постепенно увеличиваете значение *Pixel Clock* до нужной вам частоты монитора.



В контекстном меню сохраняем изменения (*File-> Save EDID Binary*) и выходим из редактора.

Дело осталось за малым, нужно подменить используемый ядром EDID файл.

Скопируем модифицированный файл из нашей домашней директории в `/usr/lib/firmware/edid`:

```
sudo mkdir -p /usr/lib/firmware/edid
sudo cp -r ~/*.bin /usr/lib/firmware/edid/edid2.bin
```

Чтобы ядро предпочитало использовать отредактированный файл EDID вместо стандартного, нам нужно указать специальный параметр ядра для модуля `drm`:

Список 24: `sudo nano /etc/modprobe.d/drm.conf`

```
options drm edid_firmware=edid/edid2.bin
```

Также необходимо добавить файл `/usr/lib/firmware/edid/edid2.bin` в образы initramfs. Для этого редактируем файл `/etc/mkinitcpio.conf` и в строке `FILES=()` пишем следующее:

```
FILES=(/usr/lib/firmware/edid/edid2.bin)
```

После чего обновляем образы initramfs через команду `sudo mkinitcpio -P`.

Затем перезагружаемся и наслаждаемся новой плавностью картинки.

Предупреждение

Если вы используете закрытый драйвера NVIDIA, то параметр ядра `drm.edid_firmware` для вас просто не будет работать. Вместо него вы должны прописать параметр `Option "CustomEDID" "HDMI-0:/home/ваше_имя_пользователя/edid.bin"` (тип подключения меняете на требуемый) в ваш конфиг `/etc/X11/xorg.conf`. Как можно понять, при этом разгон не будет работать в Wayland.

1.11 Настройка параметров ядра

Предупреждение

Данный раздел находится в разработке и предназначен для опытных пользователей Linux, которые хотят выполнить более тонкую настройку системы. Тем не менее, в нем автор немного упростил внутреннюю картину работы ядра, чтобы не делать текст чрезмерно объемным и сложным для понимания простому пользователю. Так, например, здесь вы увидите несколько упрощенную модель устройства виртуальной памяти ядра Linux, поэтому если вы эксперт в вопросе, то не судите строго.

1.11.1 Введение

Как и любая другая программа, ядро имеет свои собственные настройки и параметры, которые контролируют поведение определенных его частей. И хотя настройки ядра Linux являются менее очевидными для понимания и часто "скрыты" с глаз обычных пользователей, в данном разделе мы рассмотрим различные настройки ядра с целью улучшения производительности и отзывчивости системы для домашнего ПК или ноутбука, подобрав наиболее оптимальные значения в зависимости от вашей конфигурации.

Как мы поймем далее, несмотря на то, что ядро Linux принято считать монолитным, все настройки ядра относятся к определенным его подсистемам, поэтому раздел будет разбит на систематические блоки, каждый из которых будет выполнять настройку конкретной подсистемы, будь то подсистема ввод/вывода или сети.

Стоит также отметить, что так как ядро является общей составляющей всех дистрибутивов Linux, то почти вся информация которая будет представлена в этом разделе применима не только к Arch Linux, но и к другим дистрибутивам. Учтите, что некоторые параметры зависят от вашей версии ядра, которая, по понятным причинам, от одного дистрибутива к другому может отличаться, поэтому обращайте внимание на примечания, где указано с какой версии появился тот или иной параметр. Узнать версию используемого вами ядра можно через команду `uname -r`.

1.11.1.1 Зачем?

Часто люди задаются вопросом, зачем пытаться лезть под капот, когда "очевидно", что все уже настроено и отполировано за тебя. Отчасти это правда, и ядро Linux с каждой версией улучшается и "вылизывается" тысячами разработчиками по всему миру, и, наверное, параметры о которых пойдет речь далее уже имеют наиболее оптимальные значения. Но к сожалению, это не совсем так. Или скорее совсем не так. Во-первых, разработчики ядра часто не могут иметь представления о том, на каком конкретном железе будет работать ядро и для каких целей оно будет использоваться, в следствии чего главным приоритетом при разработке является совместимость и адаптивность ядра к как можно большему числу возможных задач и конфигураций. Такой компромиссный подход к разработке не всегда дает наилучшие результаты в чем-то конкретном, но зато позволяет ядру Linux одинаково подходить как для работы на серверах, роутерах, телефонах, микроконтроллерах, так и простых ПК. Грубо говоря, ядро Linux представляет собой швейцарский нож от мира IT, которым хоть и можно порезать хлеб, но удобнее это сделать обычным ножом. Наша задача в данном разделе это как раз заточить ядро под конкретную задачу, в нашем случае это интерактивное использование на домашнем компьютере.

Если вы переживаете за стабильность вашей системы, то предварительно сделайте резервную копию, хотя на самом деле все параметры, о которых пойдет речь далее, могут быть отключены в любой момент простым удалением файла настройки, поэтому даже при возникновении проблем со стабильностью или регрессиями у вас не должно возникнуть проблем с откатом к стоковым значениям.

1.11.1.2 Виды настроек

В ядре Linux все параметры можно поделить на типы в зависимости от способа установки их значения[1]. Часть из них может быть установлена только на этапе загрузки ядра, то есть в качестве опций командой строки[2]. Это то, что мы обычно пониманием под просто "параметрами ядра". Они указываются в настройках вашего загрузчика, будь то GRUB, refind или systemd-boot. К этой же категории можно отнести параметры модулей ядра, значения к которым передаются во время их загрузки. При этом значения параметров могут быть переданы как часть общих параметров загрузки ядра в конфиге вашего загрузчика при их указании в следующем формате: `module.parameter=value` (например `nvidia-drm.modeset=1`). То есть сначала указывается имя модуля (драйвера), затем имя параметра и через знак равно передается значение. Другой способ передачи параметров для модулей - это использование конфигурационных файлов `modprobe`. В этом случае не имеет значения какой у вас загрузчик, достаточно создать файл в директории `/etc/modprobe.d` с любым названием, и прописать параметры в следующем формате:

```
options module parameter1=value1 parameter2=value2
```

Именно вторым способом передачи параметров по возможности мы будем пользоваться далее в разделе, когда речь будет заходить о настройке различных модулей.

1.11.1.2.1 sysctl

Другой тип, это параметры, значение которых можно изменить прямо во время работы системы, что называется "на лету". Такие настройки представлены в виде файлов на псевдофайловой системе procfs в директории `/proc/sys`[3]. procfs называется псевдофайловой потому, что физически она не расположена на диске, а все файлы и директории создаются самим ядром при запуске системы в оперативной памяти. По этой причине у них отсутствует размер, и все они имеют чисто служебный характер. В директории `/proc/sys/` каждая настройка это отдельный файл, куда мы должны просто передать (записать) значение в виде числа (часто 1 и 0 означают включить/выключить, но некоторые параметры также могут принимать значение, которое имеет строго определенный смысл, и только из определенного диапазона). Все настройки объедены в директории, которые характеризуют их отношение к чему-то общему. Например, все файлы в подкаталоге `vm` - это настройки для механизма виртуальной памяти ядра[4] , включая настройки для подкачки, кэширования, и т. д. В `kernel` - общие настройки ядра[5], а в `net`[6] - настройки сетевой подсистемы, протоколов TCP и IP. Именно эти три категории мы и будем рассматривать далее.

Конечно, процесс поиска всех файлов-настроек и записи значений средствами командой строки каждый раз весьма утомителен. Поэтому разработчики создали специальную утилиту под названием `sysctl`, которая значительно упрощает данный процесс. Теперь нам не нужно лазить каждый раз в `/proc/sys/`, чтобы изменить значение параметров. Вместо этого достаточно прописать в терминале:

```
sysctl -w kernel.sysrq=1
```

Это то же самое, что и данная команда, которая прописывает значение напрямую в файл из директории `/proc/sys/`:

```
echo "1" > /proc/sys/kernel/sysrq
```

Обратите внимание, что для изменения настроек всегда нужны права root, поэтому перед каждой такой командой мы должны добавить `sudo`:

[1] https://medium.rip/@justaboutcloud/a-dive-deep-into-kernel-parameters-part-1-kernel-boot-parameters-139905e3432
[2] https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html
[3] https://www.kernel.org/doc/html/latest/admin-guide/sysctl/index.html
[4] https://www.kernel.org/doc/html/latest/admin-guide/sysctl/vm.html
[5] https://www.kernel.org/doc/html/latest/admin-guide/sysctl/kernel.html
[6] https://www.kernel.org/doc/html/latest/admin-guide/sysctl/net.html

```
sudo sysctl -w kernel.sysrq=1
```

Другим преимуществом sysctl является то, что мы можем делать такие изменения постоянными, просто прописав соответствующую строку в файл, который находится в директории `/etc/sysctl.d/`, например в `/etc/sysctl.d/99-sysctl.conf`:

```
kernel.sysrq = 1
```

Собственно именно добавлением таких строк мы и будем применять соответствующие настройки.

Предупреждение

Настройки прописываемые в файле `/etc/sysctl.conf` не применяются начиная с версии 21x в systemd, поэтому прописывайте их только в файлах, которые расположены в подкаталоге `/etc/sysctl.d`. Имя файла не имеет значения.

1.11.1.2.2 tmpfiles.d

К сожалению, далеко не все настройки ядра можно изменить при помощи sysctl или псевдофайловой ФС `/proc/sys`. Часть из них является отладочными, поэтому они расположены в виде файлов на другой псевдофайловой системе - sysfs, которая в основном отвечает за представление информации об устройствах, которыми управляет ядро. В директории в `/sys/kernel` представлены ряд других полезных параметров, которые мы рассмотрим в рамках общей темы. Чтобы выполнить установку значения в файлах, которые находятся в `/sys/kernel/`, мы будем использовать такой инструмент как systemd-tmpfiles.d[7]. Он есть только в дистрибутивах, использующих systemd в качестве системы инициализации, то есть в большей части дистрибутивов Linux включая Arch. Суть этой службы состоит в управлении, создании и удалении временных файлов или редактировании уже существующих. В нашем случае мы будем его использовать для записи значений в файлы настроек расположенных в `/sys/kernel/`. Для этого, по аналогии с sysctl, нужно создать файл в директории `/etc/tmpfiles.d`, например `/etc/tmpfiles.d/99-settings.conf`. Формат записи каждой строки в файле будет следующим:

```
w! /sys/kernel/mm/lru_gen/min_ttl_ms - - - - 2000
```

Первый символ - это тип действия, который `systemd-tmpfiles` будет выполнять с указанным по пути файлом. В нашем случае мы будем использовать только запись *w* некоторого значения в уже существующие файлы, а не создавать новые. Восклицательный знак `!` указывает, что значение будет прописываться только один раз при загрузке системы. После пути до файла идут четыре прочерка, в них должны быть указаны права доступа, которые мы хотим изменить, но так как мы имеем дело со служебными файлами, то пишем везде прочерки, чтобы ничего не менять. В конце указываем значение, которое будет прописано в файл, то есть значение параметра.

Другими словами, везде, куда не дотянется sysctl, мы будем использовать `tmpfiles`.

1.11.1.2.3 udev

По сути первых двух инструментов уже достаточно, чтобы выполнить полную настройку ядра, но мы используем ещё одну вещь - правила udev. Udev[8] - менеджер для управления вашими устройствами, который отслеживает их подключение/выключение, и предоставляет возможность создавать так называемые "правила", которые вызываются каждый раз, когда происходит определенное действие с тем или иным устройством. Внутри этого правила можно указать, при каких событиях и для какого конкретно устройства (условие для срабатывания) мы будем выполнять определенную команду или устанавливать некоторое значение. Это очень полезный инструмент, который позволит нам применять целый ряд настроек в зависимости от некоторых условий и

[7] https://www.freedesktop.org/software/systemd/man/systemd-tmpfiles.html
[8] https://www.freedesktop.org/software/systemd/man/udev.html

подстраиваясь под железо, которое у вас есть в системе. Приведу пример, чтобы стало понятнее. Для разных типов носителей подходит разный планировщик ввода/вывода. Для обычных SSD - `mq-deadline`, для HDD - `bfq`. Правила udev позволят нам при подключении определенного типа устройства сразу выбирать нужный планировщик и дополнительные параметры для него, даже если у вас в системе есть и SSD, и HDD одновременно. Подробнее планировщики ввода/вывода будут рассмотрены далее вместе с синтаксисом самих правил.

1.11.2 Оптимизация ввода/вывода

Фуууух, что же, надеюсь вы не устали от всего этого скучного вступления выше и мы можем наконец-то переходить к сути. Начнем с оптимизации ввода/вывода, то бишь к настройке подкачки (она же *свап*, от англ. *swap*), различных кэшей и планировщиков.

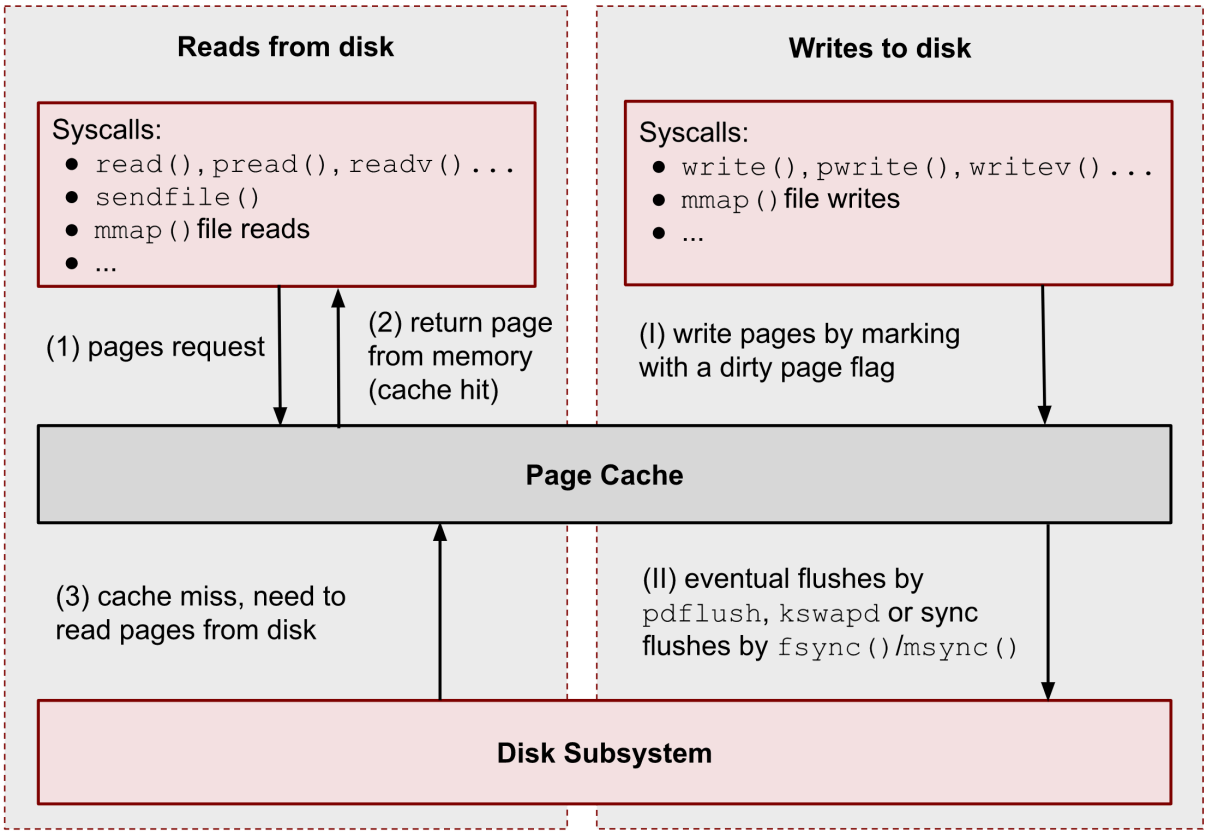
1.11.2.1 Общие сведения

Прежде чем перейти непосредственно к настройке необходимо понять принцип работы механизма виртуальной памяти и подкачки в Linux. Это важно, так как в этой теме ходит целая куча различных мифов, которые мы сейчас разберем.

Итак, для начала чрезвычайно важно понять, что ядро Linux разбивает всю вашу память на маленькие "гранулы" - страницы памяти, как правило по 4 КБ (для x86 архитектуры), не больше и не меньше. Это может показаться странным, но если не вдаваться в технические подробности, то такой подход позволяет ядру Linux проявлять достаточно большую гибкость, так как данные страницы могут быть одинаково обработаны ядром вне зависимости от того, что в них записано, предотвращая обильную фрагментацию. Тем не менее, все страницы памяти можно разбить на несколько типов. Сейчас мы не будем рассматривать их все, но остановимся на самых главных:

- Файловая "подложка" или *файловые страницы* - это страницы в которых ядро "отображает", то есть представляет данные файла, считываемые с диска в виде страниц в памяти. С этими страницами тесно связано понятие страничного кэша (page cache)[9]. Если некоторый процесс открывает какой-то новый файл и читает из него информацию, то в первый раз ядро считывает эти данные с диска и сохраняет их в страничном кэше, а все последующие операции ввода и вывода к этим же данным будут осуществляться уже при использовании кэша, что значительно ускоряет все базовые операции чтения и записи, предотвращая повторные обращения к диску. При этом память для таких страниц выделяется по требованию, поэтому если процесс открыл файл, но ничего из него не читает, то никакой реальной памяти для таких страниц выделено не будет. Собственно, то, что вы видите в графе "Кэш" в любой программе аналоге системного монитора в Linux - и есть страничный кэш. Обратите внимание, что исполняемые файлы (программы) тоже загружаются в память как файловые страницы.

[9] https://docs.kernel.org/admin-guide/mm/concepts.html#anonymous-memory



(Licensed under the CC BY-NC 4.0. © Vladislav Biriukov, All rights reserved)

- Очевидно, что далеко не все данные, которыми оперирует программа, могут быть представлены в виде реальных файлов на диске, поэтому были созданы *анонимные страницы*, которые, как следует из названия, не ассоциированы с файлами[10]. Программы запрашивают их у ядра во время своей работы для динамических данных. Если вы разработчик, то вы наверняка сталкивались с такими понятиями как "Куча" (Heap) и "Стек" (Stack). Так вот, ядро хранит данные из кучи и стека именно в анонимных страницах памяти.
- Грязные страницы (dirty pages) - по сути это подвид файловых страниц, ключевое отличие которых состоит в том, что программы в них пишут какие-то изменения, а так как ядро кэширует все данные считываемые из файлов во избежание излишней нагрузки на диск, то изменения, которые программа делает с файлом, на самом деле происходят сначала в кэше, и только потом синхронизируются с реальным файлом на диске. Более подробно об этом виде страниц и процессе их синхронизации с диском мы поговорим в следующем разделе.

Вернемся к подкачке. Один из самых больших мифов, связанных с подкачкой, состоит в том, что пользователи рассматривают её как некую "дополнительную память", которую свободно можно использовать в случае нехватки реальной, то есть физической памяти. Это конечно же не так, хотя бы потому, что процессор имеет доступ к оперированию только данными, которые находятся внутри ОЗУ. В случае нехватки памяти у ядра есть по сути всего один вариант - это освобождать уже имеющуюся память от тех страниц, которые не используются в данный момент, выгружая их в область на диске которую мы и называем подкачкой. Да, память не берется из воздуха, и подкачка - это просто "чердак", куда ядро скидывает все неиспользуемые вещи, чтобы освободить место для новых или более часто используемых страниц. При этом для процессов не меняется ровным счетом ничего, ибо они как и раньше могут обратиться к данным в памяти, которые были расположены на странице, которая была вытеснена ядром в подкачку, но когда процесс это сделает, ядро найдет эту страницу, считает её

[10] https://biriukov.dev/docs/page-cache/2-essential-page-cache-theory/

из подкачки и обратно загрузит в оперативную память. Это ещё одно преимущество механизма виртуальной памяти, повсеместно используемого ядром Linux.

Вопрос лишь в том, какие именно страницы нужно "вытеснить" из памяти. На самом деле, это достаточно сложный вопрос. Прежде всего, конечно же это будут именно анонимные страницы, так как файловые страницы и так по сути ассоциированы с данными на диске, следовательно в случае чего их точно так же можно повторно считать, и выгружать их в подкачку просто не имеет никакого смысла, что и происходит на практике. Но что если анонимных страниц много, а часть из них реально используется программами в данный момент? Какие из них тогда должны первым делом попасть в подкачку? На данный и многие другие вопросы отвечает специальный алгоритм в ядре Linux, называемый LRU (Least recently used) (а поныне и MGLRU). Если очень упрощенно, то данный алгоритм ведет учет использования каждой страницы, то есть количество обращений к ней, и на основе данной статистики предполагает, какие из них реже всего используются процессами, и следовательно какие из них можно без проблем выгрузить в подкачку.

Рядовые пользователи часто не до конца понимают, какие именно данные расположены у них в подкачке. Теперь мы можем дать чёткий ответ: в подкачке хранятся только неиспользуемые анонимные страницы памяти.

1.11.2.2 Настройка подкачки

Мы разобрались с основополагающими понятиями, и наконец-то можем переходить к настройке. Для настройки поведения подкачки используется параметр sysctl `vm.swappiness` (значение по умолчанию 60)[с. 50, 4]. Вокруг него так же ходит целый ряд заблуждений, что приводит к неправильным умозаключениям. Итак, во-первых, `vm.swappiness` напрямую никак не влияет на то, когда у вас начнет использоваться подкачка, то есть его значение - это вовсе не процент занятой памяти, при достижении которого начинает использоваться подкачка. Ядро всегда начинает использовать подкачку только в ситуациях нехватки памяти (это, как правило, когда занято 85-90% ОЗУ).

Во-вторых, параметр `vm.swappiness` влияет только на предпочтение ядра к вытеснению определенного типа страниц в случае этой самой нехватки. Он принимает значения от `0` до `200` (начиная с версии ядра 5.8 и выше, до этого максимальным значением было 100). Для более наглядного понимания, параметр `vm.swappiness` можно представить в виде весов, где более низкие (ниже 100) значения приводят к склонности ядра сначала вытеснять все страницы из файлового кэша, а более высокие (больше 100) - освобождение анонимных страниц из памяти в подкачку[11]. Значение `100` - это своего рода баланс, при котором ядро будет в одинаковой степени стараться вытеснять как файловые, так и анонимные страницы.

Другим крайне распространенным заблуждением является то, что более низкие значения `vm.swappiness` уменьшают использование подкачки - следовательно уменьшается нагрузка на диск, и что это якобы увеличивает отзывчивость системы. На деле это лишь на половину правда, так как, да, ядро при низких значениях старается откладывать использование подкачки, хотя это и не значит, что она вообще не будет использоваться, но важно понять, что это происходит за счёт более агрессивного вытеснения файловых страниц из страничного кэша - что точно так же приводит к нагрузке на ввод/вывод. Почему? Потому что каждый раз, когда ядро вытесняет страницу из страничного кэша, это приводит к тому, что все ранее хранящиеся в ней данные снова придется считывать с диска по новой.

Во-вторых, нагрузка на ввод/вывод, которую создаёт подкачка оказывается слишком переоценена. Для современных SSD накопителей переварить такую нагрузку без замедления работы системы не составит труда. Тем не менее, если страница была вытеснена в подкачку, то любая операция обращения к ней будет в разы медленнее, чем если бы она находилась в ОЗУ, даже если ваш носитель это NVMe накопитель, то операция записи страницы в файл/раздел подкачки и последующая операция чтения из него будет в любом случае затратна. Но даже если у вас HDD, то вам на помощь спешит Zswap - ещё один встроенный механизм ядра Linux, позволяющий значительно снизить нагрузку на диск и ускорить процесс вытеснения. Он представляет собой буфер в памяти, в который попадают анонимные страницы, которые на самом деле должны были попасть в подкачку на диске, и сжимаются внутри него, экономя тем самым драгоценную память насколько это возможно. Если пул страниц Zswap заполнится (по умолчанию он равен 20%), то ядро выполнит выгрузку страниц из Zswap в подкачку[12].

[11] https://www.howtogeek.com/449691/what-is-swapiness-on-linux-and-how-to-change-it/
[12] https://docs.kernel.org/admin-guide/mm/zswap.html

На сегодняшний день механизм Zswap используется во многих дистрибутивах Linux *по умолчанию*, в том числе в Arch, просто вы об этом могли не знать, и потому могли думать, что ядро "насилует" ваш диск при малейшем использовании подкачки. Никакой дополнительной настройки для его работы как правило не требуется.

Учитывая всё вышеперечисленное, автор рекомендует устанавливать значение `vm.swappiness` в `100`. Это позволит ядру равномерно вытеснять в подкачку оба типа страниц. В современных реалиях выкручивание параметра в низкие значения не приводит к желаемому эффекту. Конечно, всё индивидуально, и имеет смысл поиграться на своем железе, чтобы понять что лучше подходит лично вам имея прописанный багаж знаний по теме. Зафиксировать это значение можно через конфиг sysctl:

Список 25: `sudo nano /etc/sysctl.d/90-sysctl.conf`

```
vm.swappiness = 100
```

Предупреждение

Автор настоятельно не рекомендует устанавливать значение параметра в 0 или отключать подкачку вовсе. Подробнее о том, почему это вредно читайте в данной статье - https://habr.com/ru/company/flant/blog/348324/. Если вы хотите минимизировать использование подкачки чтобы минимизировать нагрузку на ввод/вывод, то используйте ZRAM, о котором пойдет речь далее.

1.11.2.2.1 ZRAM

Но что делать, если у вас и правда очень медленный носитель или вы хотите минимизировать нагрузку на ввод/вывод и износ диска? В этом случае лучшим решением является использование ZRAM - вида подкачки, при котором все неиспользуемые анонимные страницы не выгружаются на диск, а сжимаются прямо внутри памяти при помощи алгоритмов сжатия без потерь[13]. Точно так же как вы сжимаете простые файлы через архиватор, то же самое делает ядро со страницами памяти. Понятно, что уже сжатые страницы использовать нельзя, поэтому если они снова понадобятся процессу, то ядру придется их расжать перед использованием. Конечно, стоит учитывать, что сжатие и расжатие страниц происходит ресурсами процессора, и это имеет определенные накладные расходы, но они довольно несущественны для современных многоядерных процессоров, чтобы ими можно было пренебречь. Тем не менее, всегда можно выбрать более "легковесный" алгоритм сжатия.

Примечание

Некоторые пользователи задаются вопросом: В чем разница между zswap и ZRAM? На самом деле хотя они и занимаются по сути одной и той же работой, разница здесь в том, что Zswap является сжатым *буфером* в памяти, то есть промежуточным звеном между памятью и подкачкой, которое призвано помочь минимизировать нагрузку на ввод/вывод, а не заменить обычную подкачку на диске целиком как это делает ZRAM. Вытеснная страница при включенном Zswap имеет следующий цикл жизни: RAM -> Zswap -> Подкачка. Если процесс обратиться к странице, которая была вытеснена в Zswap, но которая так и не попала в подкачку на диске, то тогда ядро просто распакует её внутри памяти готовой для использования. В случае если она всё таки была вытеснена на диск, ядро считает её с диска и загрузит в память, как это обычно и происходит без zswap.

Об установке ZRAM было уже коротко рассказано в разделе *Настройка служб*. Однако не во всех дистрибутивах Linux есть служба `zram-generator`, поэтому покажем универсальный способ его настройки, основанный на обычных правилах udev.

Прежде чем мы перейдем к настройке ZRAM надо уточнить, что одновременное использование ZRAM и zswap имеет неопределенный эффект. С одной стороны, это вполне возможно, и в этом случае Zswap становится

[13] https://docs.kernel.org/admin-guide/blockdev/zram.html

промежуточным буфером уже для ZRAM, но это не имеет особого смысла, так как они оба занимаются одним и тем же - сжатием данных внутри ОЗУ. ZRAM также ведет свою статистику о том, какие страницы и в каком количестве были сжаты, и которая может быть искажена, в силу того что помимо него в системе может работать Zswap, поэтому настоятельно рекомендуется его отключить перед использованием ZRAM. Для этого достаточно указать параметр ядра `zswap.enabled=0` в конфиге вашего загрузчика, либо деактивировать прямо во время работы системы:

```
echo 0 > /sys/module/zswap/parameters/enabled
```

Если у вас затруднения с настройкой вашего загрузчика (а такое вполне может быть на атомарных системах), то вы можете настроить его перманентное отключение через создание файла в директории `/etc/tmpfiles.d` со следующим содержимым:

Список 26: `sudo nano /etc/tmpfiles.d/90-disable-zswap.conf`

```
w! /sys/module/zswap/parameters/enabled - - - - 0
```

Примечание

Важно отметить, что для использования ZRAM вам вовсе не обязательно отключать обычную подкачку, если она у вас до этого была настроена. В этом случае ядро по умолчанию будет использовать в качестве основной подкачки тот раздел или файл, примонтированный в служебную точку монтирования `[swap]`, который имеет приоритет выше, чем другой. Поэтому если вы установите для ZRAM приоритет `100`, как мы это сделаем ниже в файле `/etc/fstab`, то обычная подкачка на диске станет использоваться ядром только как запасная в случае если ZRAM переполнится, либо при использовании функции гибернации, которая может работать только с подкачкой на диске.

Перейдем к настройке ZRAM. Обратите внимание, что среди "мейнстримных" дистрибутивов Linux (как например Fedora) ZRAM начинают поставлять по умолчанию вместо обычной подкачки на диске. Поэтому сначала проверьте, не задействован ли уже ZRAM в вашей системе. Сделать это можно очень просто через команду `zramctl`, либо проверив по наличию файла `/dev/zram0`, который представляет собой блочное устройство куда будут попадать все вытесняемые ядром страницы (этакий виртуальный раздел подкачки).

Если же нет, то продолжаем. Для начала нам нужно форсировать загрузку модуля ZRAM, для этого нужно создать файл в директории `/etc/modules-load.d/30-zram.conf` и прописать в него всего одну строчку:

Список 27: `sudo nano /etc/modules-load.d/zram.conf`

```
zram
```

Теперь используя правила udev, мы будем создавать наше блочное устройство `/dev/zram0` и делать из него раздел подкачки. Для этого создадим файл в директории `/etc/udev/rules.d/30-zram.rules`:

Список 28: `sudo nano /etc/udev/rules.d/30-zram.rules`

```
ACTION=="add", KERNEL=="zram0", ATTR{comp_algorithm}="zstd", \
    ATTR{disksize}="8G", \
    RUN="/usr/bin/mkswap -U clear /dev/%k", TAG+="systemd"
```

Теперь подробно о том, что из себя представляет само udev правило. В начале мы указываем при каком действии мы хотим, чтобы оно срабатывало. В нашем случае это `ACTION=="add"`, то есть появление нового блочного устройства под названием `KERNEL=="zram0"`. Это блочное устройство создается ядром автоматически при загрузке модуля ZRAM, форсированную загрузку которого мы уже прописали выше. Здесь можно заметить, что

все проверки в правилах udev осуществляются через ==.

А дальше мы говорим, что в этом случае нужно делать. Во-первых, мы меняем значение атрибута (в udev правилах все они пишутся как `ATTR{name}`, где *name* - имя атрибута) `comp_algorithm` нашего блочного устройства, который указывает на используемый алгоритм сжатия. Для ZRAM в ядре предложены три алгоритма сжатия: `lzo`, `lz4`, `zstd`. В подавляющем большинстве случаев вы должны использовать только `zstd`, так как это наиболее оптимальный алгоритм по соотношению скорости/эффективности сжатия. LZ4 может быть быстрее при расжатии, но в остальном он не имеет больших преимуществ. LZO следует использовать только на очень слабых процессорах, которые просто не тянут сжатие большого объема страниц через Zstd.

Следующим атрибутом мы меняем `disksize` - это размер блочного устройства. Теперь очень важно: размер блочного устройства - это тот объем **несжатых страниц**, который может попасть внутрь ZRAM, и он может быть равен объему ОЗУ или даже быть в два раза больше него. Как это возможно? Представим, что у вас 4 Гб ОЗУ. Вы устанавливаете объем ZRAM тоже в 4 Гб. Вы полностью забиваете всю свою память, открывая 300 вкладок в Chromium, и любой системный монитор или аналог `htop` покажет вам, что подкачка тоже полностью забита, но проблема в том, что это тот размер страниц, которые попали в ZRAM до сжатия. То есть на деле у вас в ОЗУ вытесненные страницы занимают в разы меньший объем *из-за сжатия*. Увидеть это можно через команду `zramctl`, вывод которой может быть следующим:

```
NAME       ALGORITHM DISKSIZE DATA COMPR  TOTAL STREAMS MOUNTPOINT
/dev/zram0 zstd           15G   1G  232M 243.3M      16 [SWAP]
```

Здесь колонка `DATA` показывает какой объем страниц попал в `/dev/zram0`. Если вы опять откроете `htop` или другой аналог системного монитора, то вы увидите точно такой же объем того сколько у вас "занято" подкачки, но вот колонка `COMPR` показывает уже реальный размер вытесненных внутрь ZRAM страниц *после сжатия*, который очевидно будет меньше в 2-3 раза. Именно поэтому я рекомендую вам установить объем блочного устройства ZRAM, который в два раза больше, чем объем всей вашей памяти (Значение `8Gb` - **это лишь пример**, замените его на то, какой объем вашей памяти и умножьте это на **два**). Конечно, здесь нужно оговориться, что не все страницы бывают так уж хорошо сжимаемыми, но в большинстве случаев они будут помещаться без каких-либо проблем.

Надеюсь это добавило понимание того, почему не всегда нужно верить цифрам, которые вам говорит, например, команда `free`. Завершает наше udev правило действие, которое мы хотим сделать с нашим блочным устройством - запустить команду `mkswap`, чтобы сделать из нашего `/dev/zram0` раздел подкачки.

Всё, что нам осталось теперь - это добавить запись в `/etc/fstab`, что `/dev/zram0` это вообще-то наша подкачка и установить ей приоритет `100`.

Список 29: `sudo nano /etc/fstab`

```
/dev/zram0 none swap defaults,pri=100 0 0
```

На этом все, теперь можно перезагружаться и проверять работу через `zramctl`. Если такой способ для вас показался слишком сложным, то обратитесь к использованию `zram-generator` как уже было показано ранее.

Значение же `vm.swappiness` при использовании ZRAM рекомендуется установить в `150`, так как более низкие значения приведут к излишнему вытеснению из файлового кэша, а анонимные страницы, которые потенциально могут быть легко сжаты, будут вытесняться в последний момент, что нежелательно. А вот при значении `150`, файловый кэш будет дольше оставаться нетронутым, благодаря чему обращения к ранее открытым файлам останутся быстрыми, но при этом анонимные страницы просто сожмутся внутри памяти. Такой подход минимизирует нагрузку на ввод/вывод.

1.11.2.2.2 Отключение упреждающего чтения

Из-за того, что процесс чтения вытесненной в подкачку страницы с диска и её записи обратно в оперативную память является довольно дорогостоящей операцией, ядро использует некоторые трюки, для того чтобы делать их как можно реже. Один из таких трюков это "упреждающее чтение" (*readahead*), когда при обращении процесса к вытесненной странице, ядро считывает не только запрошенную страницу, но и ещё некоторое количество страниц последовательно следующих за ней внутри подкачки.

Смысл здесь в том, что страница на практике это очень маленький фрагмент данных, которыми оперирует процесс, поэтому с большой долей вероятности обратившись к одной 4 Кб странице, процесс сделает ещё два и более запросов к тем страницам, которые тоже могли быть вытеснены в подкачку и быть записанными в него после той, которую процесс запрашивает в данный момент, и чтобы их потом тоже не искать и не читать ядро делает это сразу вместе с той вытесненной страницей, которую запросил процесс сейчас, так скажем, двух зайцев одним выстрелом.

Количество таких последовательно считываемых страниц за раз контролируется значением параметра `vm.page-cluster`. Это значение является степенью двойки, возведя в которую и можно получить количество страниц. Например, если установлено значение `1`, то количество страниц, которые ядро считает заранее, будет равно 2^1, то есть просто два. Если значение параметра равно `2`, то количество страниц уже будет равно в 2^2, то есть `4` и так далее. При значении `0` количество страниц будет 2^0, то есть 1 - это значение отключает упреждающее чтение страниц из подкачки.

На первый взгляд всё звучит здорово, и надо бы выкрутить значение побольше, чтобы ядро читало больше страниц за раз, но есть одна маленькая проблема, из-за которой я настоятельно рекомендую отключать этот параметр. Дело в том, что ядро считывает из подкачки страницы, которые были записаны по порядку за той страницей, которая в данный момент запрошена для загрузки обратно в память. Мы подразумевали, что это будут страницы того же процесса, который запросил данную страницу, но на деле это может вообще не так. Ядро записывает страницы из памяти в подкачку в том порядке, в котором они были вытеснены, и они вообще не обязательно могут относится к одному и тому же процессу, а даже если к одному, то могут быть совсем не теми, которые процесс запросит в будущем. Короче говоря, с упреждающим чтением мы играем в своего рода рулетку, повезет или нет. Но в подавляющем большинстве случаев ядро просто вернет в память обратно ещё 8 страниц (согласно значению по умолчанию), которые могут никогда не пригодиться в будущем, а если они не пригодятся, то их придется опять вытеснять в подкачку.

Таким образом, упреждающее чтение не только не решает заявленную проблему, но и наоборот её усугубляет. Для ZRAM это, конечно, может и не так критично, так как это вызовет лишь дополнительные циклы сжатия/расжатия страниц, но это в любом случае холостая работа. По этой причине разработчики ChromeOS и Android отключают данный параметр в своих системах по умолчанию[14][15], что советую сделать и вам. Для этого как обычно достаточно просто прописать значение в конфиге sysctl:

[14] https://issues.chromium.org/issues/41028506

[15] https://chromium.googlesource.com/chromiumos/overlays/chromiumos-overlay/+/HEAD/chromeos-base/chromeos-base/files/00-sysctl.conf#116

Список 30: `sudo nano /etc/sysctl.d/99-sysctl.conf`

```
vm.page-cluster = 0
```

1.11.2.3 Алгоритм MGLRU

Мы уже говорили, что LRU - это алгоритм используемый ядром Linux для ведения учёта количества обращений ко всем страницам внутри памяти, позволяющий составлять выборку тех страниц, которые реже всего используются процессами и соответственно могут быть спокойно вытеснены в подкачку. Но начиная с версии 6.1 в ядре появилась альтернативная реализация этого алгоритма, называемая *MGLRU* (Multi-Generational LRU)[16]. Принципиальное отличие MGLRU от простого LRU алгоритма состоит в том, что выборка страниц, которые должны быть вытеснены, формируется не на основе только лишь одного признака (количества обращений к странице), а на основе целых двух признаков - количества обращений и времени последнего обращения. По этой причине новый алгоритм объединяет все страницы в так называемые "поколения" на основе времени обращения к ним, собственно именно поэтому его название и можно дословно перевести как "Многопоколенный LRU". Такой подход позволяет добиться большей точности в выборе из имеющихся страниц тех, которые по настоящему используются реже других, что в свою очередь позволяет уменьшать количество операций возврата страниц из подкачки, ибо чем точнее работает алгоритм выборки, тем больше вероятность, что вытесненная страница действительно никогда больше не понадобится и её не надо будет считывать с диска и загружать обратно в память.

Для того чтобы проверить собрана ли ваша версия ядра с поддержкой MGLRU достаточно прописать одну команду:

```
zgrep "CONFIG_LRU_GEN_ENABLED" /proc/config.gz
```

Если вывод команды не пустой, значит ваша текущая версия ядра собрана с поддержкой данного алгоритма, но это вовсе не значит, что он используется по умолчанию. Алгоритм MGLRU можно беспрепятственно включить или выключить прямо во время работы системы. Проверить статус работы алгоритма можно через файл `/sys/kernel/mm/lru_gen/enabled`:

```
cat /sys/kernel/mm/lru_gen/enabled
```

Если вывод команды равен `0x0000`, значит MGLRU выключен, и его нужно самостоятельно включить следующей командой:

```
echo "y" | sudo tee /sys/kernel/mm/lru_gen/enabled
```

Обратите внимание, что в большинстве дистрибутивов Linux версии ядра с поддержкой MGLRU поставляются по умолчанию, поэтому никаких дополнительных действий для его включения делать как правило не нужно.

1.11.2.3.1 Защита от Page Trashing

Одним из преимуществ алгоритма MGLRU над своим предшественником является предоставление дополнительной защиты от ситуаций Page Trashing.

Page Trashing - это ситуация острой нехватки памяти, при которой памяти становится настолько мало, что ядро начинает вытеснять в подкачку даже те страницы, которые активно используются процессами во время своей работы, так как все остальные малоиспользуемые страницы уже были вытеснены. Это приводит к тому, что количество операций возврата страниц из подкачки многократно увеличивается, так как к данным часто используемым страницам все время обращаются процессы, из-за чего ядру приходится читать их из подкачки с диска или распаковывать их из памяти, если речь идёт про ZRAM, и заново загружать память, после чего снова их вытеснять, так как других кандидатов для этого больше не осталось. Такой цикл становится очень заметным для

[16] https://docs.kernel.org/admin-guide/mm/multigen_lru.html

пользователя, так как он порождает кратковременные зависания системы, ибо процессу каждый раз приходится ожидать, пока ядро достанет страницы из подкачки и загрузит их обратно в память.

Конечно, если потребление памяти в этом случае продолжит расти, то мы столкнемся с ситуацией Out Of Memory (OOM), после чего либо специальный демон по наводке ядра убьёт самый прожорливый процесс, чтобы освободить память, либо система полностью зависнет. Если потребление останется тем же, то мы продолжим испытывать постоянные микрозависания, что не очень приятно.

Здесь на сцену выходит алгоритм MGLRU, который хоть и не позволяет на 100% защититься от таких ситуаций, но позволяет убрать те самые кратковременные зависания, сделав систему более стрессоустойчивой и отзывчивой в условиях нехватки ОЗУ. Суть защиты состоит в том, что MGLRU предотвращает вытеснение "рабочего набора" страниц процесса (то есть таких страниц, которые действительно активно используются) в течении `N` миллисекунд, оставляя их не тронутыми в памяти на протяжении по крайне мере этого гарантированного времени. В этом случае процессам не придется каждый раз ожидать долгого восстановления страниц из подкачки и они сохранят свою скорость работы, но с другой стороны это увеличивает шанс возникновения ситуаций OOM, так как чем больше разрастается такой "рабочий набор" страниц, тем больше потребление памяти. По этой причине данный механизм защиты выключен по умолчанию, так как возникновение OOM ситуаций часто нежелательно на серверах и системах с большой нагрузкой, не предназначенных для интерактивного использования, где такие небольшие зависания были бы заметны глазу.

Для того чтобы включить данный механизм при использовании MGLRU нам нужно изменить значение параметра `min_ttl_ms` (по умолчанию 0), который как раз таки и устанавливает то время в миллисекундах, в течении которого рабочий набор страниц не будет вытесняться. Автор рекомендует брать значение от `1000` (это одна секунда), но не большее `5000`, ибо это приведет к более частому возникновению OOM. Оптимальное значение для большинства - `2000` (2 секунды). В этом случае система достаточно сохранит свою интерактивность под нагрузкой. Указать значение можно как всегда через псевдофайловую систему sysfs, для автоматизации процесса воспользуемся файлом конфигурации `systemd-tmpfiles`:

Список 31: `sudo nano /etc/tmpfiles.d/90-page-trashing.conf`

```
w! /sys/kernel/mm/lru_gen/min_ttl_ms - - - - 2000
```

1.11.2.4 Настройка грязных страниц

В теоретическом разделе про работу памяти в Linux мы уже говорили, что ядро отображает всю информацию об обычных файлах в виде кусочков - файловых страниц, при этом реальную память данная страница получает только непосредственно когда какая-то программа, то есть процесс начинает что-то читать или писать в файл, и если точнее, в определенное место внутри файла ассоциированное с данной страницей. Со чтением все понятно, мы просто сохраняем считанный набор байт с диска в память и многократно переиспользуем результат. Но что происходит в случае с записью?

Когда какой-то процесс начинает писать изменения в файл, то эти изменения сначала попадают в файловые страницы, но так как подразумевается, что проделанные изменения происходят с реальными файлами на диске, то перед ядром возникает задача синхронизации изменений между страничным кэшом и диском. С этой целью все измененные файловые страницы помечаются как "грязные" (*dirty pages*). Ядро ведет учёт таких страниц и в фоновом режиме, при определенных условиях, о которых пойдет речь далее, начинает "сбрасывать" такие страницы на диск, то есть записывать изменения над файлами уже по настоящему.

Смысл от такого буферизированного подхода состоит в том, чтобы минимизировать количество реальных операций записи, ибо приложения как правило большую часть времени не добавляют новые данные внутрь файла, а изменяют уже существующие и могут делать это много раз подряд в течение времени своей работы. Если приложение X изменяет 10 раз один и тот же файл в одном месте с малыми интервалами между такими операциями записи, то нет никакого смысла делать запись сразу же, ведь чем дольше ядро удерживает изменения внутри страничного кэша, тем больше уменьшает количество конечных, настоящих записей на диск, и вместо 10 операцией записи на диск мы можем получить одну запись уже итогового варианта изменений. Однако

такой подход порождает и определенные риски, так как избыточное кэширование изменений внутри ОЗУ может привести к потери данных в случае отключения питания или непредвиденного зависания системы.

Стоит также отметить, что у приложений остается возможность выполнять прямую запись в файл минуя страничный кэш. Первый способ это использование *Direct I/O* (буквально: прямой ввод/вывод). Для его применения приложению нужно установить специальный флаг при открытии файла - `O_DIRECT`, после чего все операции над этим файлом будут производиться в обход страничного кэша. Второй способ заключается в том, чтобы использовать страничный кэш большую часть времени работы программы, но форсировать его "промывку" (термин "промывка" (flush) является антонимом к слову "грязный") в определенные моменты времени, например при окончании работы с файлом или его сохранении в текстовом редакторе. В этом случае приложение выполняет системные вызовы `sync()` или `fsync()`, которые сигнализируют ядру о том, что нужно в принудительном порядке записать все проделанные им изменения из страничного кэша на диск.

Но вернемся к тому, как именно ядро сбрасывает грязные страницы на диск. За это отвечают так называемые специальные ядерные потоки `pdflush`, которые производят "промывку" грязных страниц в фоновом режиме при соблюдении некоторых условий. Во-первых, данные потоки начинают работать только тогда, когда набирается необходимый общий объем грязных страниц, который устанавливается параметрами `vm.dirty_background_ratio` или `vm.dirty_background_bytes`. До тех пор пока указанная нижняя граница не будет достигнута, изменения внутри грязных страниц так и будут оставаться в ОЗУ, за тем исключением, если, как и было указано выше, процесс явно не попросит записать на диск изменения через вызовы `sync()` или `fsync()`. При этом важно отметить, что если страница была изменена процессом, то при штатной работе потоков `pdflush` без принудительной промывки со стороны самого приложения, страница становится готовой к записи не сразу же, а только по истечению времени указанного в качестве значения параметра `vm.dirty_expire_centisecs`, которое представлено в виде сантисекунд (одна сотая от секунды) и по умолчанию равно `3000`[c. 50, 4] (30 секунд).

После запуска потоков `pdflush` их работа происходит не непрерывно как можно было бы подумать, а с интервалами между которыми они просыпаются и выполняют часть работы. Время этих промежутков определяется значением параметра `vm.dirty_writeback_centisecs`, так же принимающего значение в виде сантисекунд и равного по умолчанию `500`[c. 50, 4], то есть 5 секунд, что весьма много, но это гарантирует, что потоки `pdflush` не будут создавать чрезмерной нагрузки. Наконец, существует также верхняя граница, которая определяет максимально возможный объем грязных страниц. Она определяется значением параметра `vm.dirty_ratio`, либо `vm.dirty_bytes`. Если к тому времени, когда потоки `pdflush` начали свою работу, объем грязных страниц продолжал увеличиваться с такой скоростью, что ядро просто не успевало записать все поступающие изменения на диск, то возникает так называемый "троттлинг" ввода/вывода.

В старых версиях ядра "троттлинг" ввода/вывода проявлялся только непосредственно по достижению верхней границы количества грязных страниц, и приводил к полной блокировке всех операций ввода/вывода до тех пор пока потоки `pdflush` полностью не запишут уже накопленные ранее изменения на диск. Это приводило к очень печальным последствиям, в том числе известный баг в ядре 12309 был связан именно с тем, что интенсивная запись каким-либо процессом на носитель с очень низкой скоростью (вроде простой USB флешки) приводила к ярко выраженным зависаниям всей системы, так как операции I/O блокировались, а фоновые потоки `pdflush` не могли быстро записать изменения в силу аппаратных ограничений самого носителя.

В новых версиях ядра были предприняты большие усилия к исправлению данной проблемы[17], и в конце концов было принято решение, которое можно охарактеризовать как "размывание" процесса троттлинга во времени. То есть, когда текущий объем грязных страниц начинает быть равным примерно 1/2 между значениями `vm.dirty_background_bytes` (или `vm.dirty_background_ratio`) и `vm.dirty_bytes` (или `vm.dirty_ratio`), то есть между нижней и верхней границей соответственно, то тогда ядро начинает постепенно создать кратковременные паузы (блокировки) в работе ввода/вывода для процесса, в результате работы которого появляется большое количество грязных страниц, так чтобы потоки `pdflush` успевали обработать уже накопленные грязные страницы. Такие палки в колеса активно пишущему процессу закономерно приводят к падению пропускной способности записи, но позволяют избавиться от эффекта "голодания", когда один процесс полностью оккупирует всю квоту на грязные страницы, не позволяя ничего писать другим процессам, а также

[17] https://unix.stackexchange.com/questions/480399/why-were-usb-stick-stall-problems-reported-in-2013-why-wasnt-this-problem-so/480400#480400

от полных блокировок ввода/вывода, так как в случае достижения верхней границы ядро просто тормозит работу ввода/вывода для процесса таким образом, чтобы потоки `pdflush` гарантированно могли записать все полученные грязные страницы до снятия блокировки, как правило тем самым уравнивая скорость записи грязных страниц приложением со скоростью записи потоков `pdflush`[18],[19].

Учитывая количество параметров, контролирующих поведение грязных страниц и факторов, оказывающих влияние на их работу, возникает вполне закономерный вопрос о том, как это настроить оптимальным образом для своей конфигурации и задач? Для начала, как вы уже могли заметить, существует некоторая двойственность в вопросе указания нижней и верхней границы работы потоков `pdflush`, так для их настройки существует две пары настроек `vm.dirty_background_bytes` и `vm.dirty_bytes` или `vm.dirty_background_ratio` и `vm.dirty_ratio`. Несмотря на то, что обе пары контроллируют по сути одно и то же, они конфликтуют друг с другом, то есть указать можно только один из пары, так как указание одного отменяет значение другого. Кроме того существует некоторая разница в их семантике. Все параметры с окончанием `ratio` указывают процент от *свободной в данный момент памяти*, который могут занимать грязные страницы вообще (в случае с `vm.dirty_ratio`) или же пороговое значение для начала работы потоков `pdflush` (`vm.dirty_background_ratio`). Частое заблуждение относительно этой пары параметров состоит в том, что процент берется от общего количества памяти в целом, а не от свободной, что приводит к неправильным умозаключения о выборе значения в зависимости от объема памяти.

В целом, по мнению автора, использование параметров `vm.dirty_ratio` и `vm.dirty_background_ratio` нежелательно, так как их поведение не является строго фиксированным и объем грязных страниц таким образом находится в обратной пропорциональной зависимости по отношению к текущему уровню потребления памяти, который склонен к тенденции увеличения в процессе работы системы больше, чем к уменьшению. Скажем, мы можем взять 2% от 32 Гб в качестве значения к параметру `vm.dirty_ratio`. Если в моменте вся память свободна (что, конечно, в действительности невозможно), мы получаем максимальный объем грязных страниц равный примерно 678 Мб, что на первый взгляд много, но среднестатический пользователь гораздо чаще открывает новые вкладки в браузере или открывает новые приложения, чем их закрывает, поэтому легко представить ситуацию, когда даже с 32 Гб ОЗУ вы достигаете уровня потребления 28 Гб ОЗУ, к примеру, компилируя что-то внутри tmpfs, и в этом случае объем грязных уже будет составлять всего 85 Мб и дальше ещё больше уменьшаться. То есть, по существу использование параметров с окончанием `ratio` приводит к тому, что большую часть времени работы системы объем грязных страниц представляет собой убывающую геометрическую прогрессию. В то же время другая пара параметров, `vm.dirty_bytes` и `vm.dirty_background_bytes`, не имеет такой зависимости[20] и позволяет однозначно определить порог грязных страниц для начала работы потоков `pdflush` и установить максимально возможный объем грязных страниц вне зависимости от текущего уровня потребления памяти.

Сами же значения к `vm.dirty_bytes` и `vm.dirty_background_bytes` следует выбирать в зависимости от ваших целей и задач, но для домашнего использования в качестве `vm.dirty_bytes` разумно брать тот объем данных, который ваш основной носитель может обработать за единицу времени, то есть его пропускную способность, так как тогда даже в худшем случае указанный объем грязных страниц может быть записан достаточно быстро. Значение же `vm.dirty_background_bytes` как правило лучше делать равным 1/2 или даже 1/4 от значения `vm.dirty_bytes`, так как чем больше "расстояние" между порогом к запуску потоков `pdflush` и максимальным объемом грязных страниц, тем меньше вероятность столкнутся с эффектом троттлинга и падением пропускной способности записи. Так же слишком завышенное значение `vm.dirty_background_bytes` черевато "застоем" данных внутри ОЗУ, что сулит риски их потери при отключении питания или зависаниях системы. Нужно понимать, что сверхвысокие значения просто не имеют смысла при простом домашнем использовании, так как рядовой пользователь не имеет приложений, которые могли бы иметь большую интенсивность записи данных на диск, как например СУБД. Как правило самыми интенсивными приложениями с точки зрения записи остаются торрент клиенты, Steam, и другие программы для загрузки контента, однако объем данных, который они записывают на диск ограничен пропускной способностью вашего сетевого канала, который у большинства людей хоть и чисто номинально составляет 100 Мб/с, однако в ряде случаев оказывается куда ниже, так что сверх большие объемы грязных страниц указывать просто нет смысла.

[18] https://github.com/torvalds/linux/blob/fb527fc1f36e252cd1f62a26be4906949e7708ff/mm/page-writeback.c#L410-L411
[19] https://stackoverflow.com/a/73808616
[20] https://lwn.net/Articles/456904/

В качестве начальных значений, на которые можно было бы опереться, автор рекомендует взять 32 или 64 Мб в качестве `dirty_background_bytes` и 256 Мб в качестве `dirty_bytes`:

Список 32: `sudo nano /etc/sysctl.d/30-dirty-pages.conf`

```
vm.dirty_background_bytes=67108864
vm.dirty_bytes=268435456
```

Вы вправе кратно уменьшить значение параметра `vm.dirty_bytes`, если у вас медленный HDD, или же наоборот увеличить вплоть до 1-2 Гб, если имеете сверхбыстрый носитель и высокую скорость передачи данных по сети.

Что касается значений параметров `vm.dirty_expire_centisecs` и `vm.dirty_writeback_centisecs`, которые управляют частотой работы `pdflush` потоков, то вы могли заметить, что значения по умолчанию сильно завышены. Ожидать 30 секунд, как предписывает значение по умолчанию параметра `vm.dirty_expire_centisecs`, перед тем чтобы позволить записывать `pdflush` новую грязную страницу кажется чрезмерным, поэтому разумно уменьшить значение данного параметра вдвое, то есть сократить период ожидания до 15 секунд, либо же ещё меньше, но устанавливать сверх низкие значения вроде 1-3 секунд также не рекомендуется, так как это может свести на нет все преимущества кэширования при записи. Оптимальным, по мнению автора, является значение в 15 секунд, то есть значение `1500` при переводе в сантисекунды:

Список 33: `sudo nano /etc/sysctl.d/30-dirty-pages-expire.conf`

```
vm.dirty_expire_centisecs=1500
```

Интервал времени между периодами работы потоков `pdflush` определяемый параметром `vm.dirty_writeback_centisecs` так же можно уменьшить, так как современные SSD носители достаточно хорошо справляются с интенсивной нагрузкой, поэтому можно увеличить частоту работы `pdflush` потоков и таким образом ещё больше уменьшить шансы на столкновение с эффектом троттлинга при записи:

Список 34: `sudo nano /etc/sysctl.d/30-dirty-pages-writeback.conf`

```
vm.dirty_writeback_centisecs=100
```

1.11.2.5 Настройка кэша VFS

В страничный кэш попадают не только файловые страницы, в которых хранятся непосредственно данные считываемые с диска, но и метаданные к файлам и директориям. Доступ к ним осуществляется через так называемые индексные дескрипторы (*inode*) - специальные структуры, которые используются вашей файловой системой для хранения атрибутов, прав доступа и прочей служебной информации, а также они содержат номера секторов диска, которые указывают, где хранятся данные самого файла на носителе.

Перед открытием любого файла или дириктории сначала нужно выполнить его поиск на файловой системе, и это не самая быстрая операция как может показаться, даже несмотря на различные оптимизации, предоставляемые современными файловыми системами такими как использование B-деревьев для быстрого прохода по ним. В результате этой операции ядро как раз таки находит нужный индексный дескриптор, имея который можно обратиться к данным файла. Поэтому ядро кэширует все используемые во время работы системы дескрипторы и информацию о директориях внутри VFS[21] кэша, для того чтобы сделать все последующие обращения к файлам быстрыми, потому что ядро уже будет знать про них всё, что нужно.

Но все эти метаданные так или иначе занимают место внутри памяти, поэтому когда ядро начинает "промывку"

[21] VFS (Virtual File System) - виртуальная файловая система, на деле является программным интерфейсом, который абстрагирует всё взаимодействие между конкретной файловой системой (Btrfs/ext4/xfs и т.д.) и программами, позволяя тем осуществлять запись и чтение ничего не зная о том, какая именно файловая система используется в данный момент.

(flush) страничного кэша, то оно вытесняет из него как данные самих файлов, так и метаданные для них. В ядре также есть специальный параметр sysctl `vm.vfs_cache_pressure`, который как раз таки регулирует, что будет вытесняться в первую очередь - сами данные или метаданные из кэша VFS. Здесь всё по аналогии с параметром `vm.swappiness`. При значении равном `100` (значение по умолчанию) ядро будет пытаться равномерно выгружать из памяти как кусочки содержимого самих файлов, так и индексные дескрипторы из кэша VFS. При значениях меньше `100` ядро будет больше отдавать предпочтение хранению метаданных в памяти, при значениях больше `100` - наоборот, больше избавляться от них в пользу обычных данных считываемых с диска.

Для наилучшего быстродействия системы рекомендуется устанавливать значение равным `50`[22], при котором вытеснение страниц, относящихся к VFS кэшу, происходит реже, чем для обычных файловых страниц, так как метаданные имеют большую ценность по сравнению с данными самих файлов, которые можно достаточно быстро повторно считать в страничный кэш на большинстве SSD накопителей при наличии индексного дескриптора файла, который как раз таки хранится внутри VFS кэша. Для сохранения значения как и всегда пропишем его в конфигурационный файл sysctl:

Список 35: `sudo nano /etc/sysctl.d/90-vfs-cache.conf`

```
vm.vfs_cache_pressure = 50
```

Конечно, лучший способ увеличения быстродействия ввод/вывода это кэшировать как можно больше данных в памяти, так как это самое быстрое устройство хранения в вашем компьютере (без учета кэша процессора), поэтому лучше всего как можно больше минимизировать вытеснение страниц из страничного кэша, но мы это уже сделали в разделе про настройку подкачки, установив большое значение параметра `vm.swappiness` и используя ZRAM для сжатия анонимных страниц прямо внутри памяти.

1.11.2.6 Настройка планировщиков ввода/вывода

Планировщики ввода/вывода - это специальные модули ядра, которые регулируют порядок выполнения операций ввода/вывода во времени на уровне обращения к блочным устройстам (HDD дискам или SSD/NVMe/microSD/SD накопителям). Если вам казалось, что все запросы на чтение или запись происходят сразу же, то это не так.

Все запросы к носителю сначала попадают в очередь, которой и управляет планировщик ввода/вывода. В зависимости от используемого алгоритма он "ранжирует" все поступающие запросы таким образом, чтобы запросы которые осуществляются к соседним блокам на диске шли как бы друг за другом, а не в том порядке в котором они поступили в очередь. К примеру, если к планировщику поступили запросы на чтение `9`, `3` и `5` блоков (условная запись), то он попытается разместить их в очереди как `3`, `5` и `9`. Зачем это делается? В силу исторических причин, все планировщики изначально разрабатывались с целью нивелировать недостатки механических дисков (и HDD в том числе), которые в силу своей специфики работы были чувствительны к порядку осуществления любых операций чтения или записи, так как чтобы выполнить любую операцию головке жесткого диска нужно было сначала найти нужный блок, а когда головка сначала выполняет чтение блока `9`, а потом чтение "назад" блока `3`, чтобы потом опять переместить головку вперед на блок `5`, то очевидно что это несколько уменьшает пропускную способность диска.

Поэтому все планировщики и работают по принципу "лифта" (*elevator*): когда планировщик добавляет все запросы в очередь, но при этом планирует их выполнение уже в порядке возрастания по номерам блоков, к которым они обращаются. Кроме того, планировщик всегда будет отдавать предпочтение запросам на чтение запросам на запись, в силу того, что выполнение запросов на запись может быть неявно отложено ядром, либо происходит куда быстрее в силу того, что запись сначала осуществляется в страничный кэш (то есть в ОЗУ), а только потом на диск. В случае с операциями чтения их выполнение не может быть отложено, банально в силу того, что все программы, которые читают файлы, явно ожидают получения какого-то результата.

Конечно, на деле алгоритм планирования запросов ввода/вывода куда сложнее, но общий принцип остается тем же. На текущий момент в ядре существует три "реальных" планировщика ввода/вывода: `BFQ`, `mq-deadline`, `kyber`. Существует также четвертый вариант `none`, который устанавливает простую FIFO очередь для всех

[22] https://github.com/xanmod/linux/commit/530ab0753af93a405ce429088fe1c04602e5c646

запросов. Это значит, что они будут обрабатываться ровно в том порядке, в котором поступили без какого-либо планирования.

Хотя выбор не велик, выбор планировщика может сильно зависеть от типа используемого носителя. Общие рекомендации к выбору планировщика под определенный тип носителя состоят в следующем:

- Для NVMe и SATA SSD накопителей используйте `none`. Дело в том, что вся вышеописанная логика нахождения нужных блоков с использованием головки совершенно не актуальна для твердотельных накопителей с быстрым произвольным доступом[23], где любое обращение к блокам осуществляется за фиксированное время, поэтому порядок выполнения запросов для них не имеет такого же значения как для жёстких дисков. В то же время, накладные расходы при планировании прямо пропорциональны количеству запросов в очереди, которые планировщику нужно обработать ресурсами CPU, но в NVMe и простых SSD носителях планированием поступающих запросов на аппаратном уровне уже занимается встроенный контроллер, поэтому планировщик в ядре Linux по сути работает в холостую[24], нагружая при этом процессор, что в свою очередь может вызывать кратковременные зависания системы при большой нагрузке на ввод/вывод.
- Однако для SATA SSD с плохим контроллером или устаревшим интерфейсом подключения (SATA 2) имеет смысл использовать планировщик `mq-deadline`. Для SD/microSD карт так же имеет смысл использовать только mq-deadline.
- Для HDD следует использовать BFQ, но в целом любой планировщик должен быть лучше, чем его отсутствие как уже объяснено выше.

Как вы видите, здесь мы проигнорировали планировщик Kyber по той причине, что он практически не развивается за последние 3 года (то есть не получает новых значимых улучшений/оптимизаций) и рассчитан на работу со сверх быстрыми накопителями, которые чувствительны к задержкам, что не совсем актуально для домашней системы.

Итак, теория это хорошо, но как их все таки включить? Самый универсальный способ это написать собственные правила Udev, которые могли бы автоматически выбирать нужный планировщик в зависимости от типа носителя. Чтобы создать новые правила просто создадим новый файл в `/etc/udev/rules.d/90-io-schedulers.rules`:

Список 36: `sudo nano /etc/udev/rules.d/90-io-schedulers.rules`

```
# HDD
ACTION=="add|change", KERNEL=="sd[a-z]*", ATTR{queue/rotational}=="1", ATTR{queue/
↪scheduler}="bfq"

# eMMC/SD/microSD cards
ACTION=="add|change", KERNEL=="mmcblk[0-9]*", ATTR{queue/rotational}=="0", ATTR{queue/
↪scheduler}="mq-deadline"

# SSD
ACTION=="add|change", KERNEL=="sd[a-z]*", ATTR{queue/rotational}=="0", ATTR{queue/
↪scheduler}="none"

# NVMe SSD
ACTION=="add|change", KERNEL=="nvme[0-9]*", ATTR{queue/rotational}=="0", ATTR{queue/
↪scheduler}="none"
```

(Чтобы использовать планировщик `mq-deadline` для SATA SSD просто поменяйте значение внутри кавычек в третьей строке с `none` на `mq-deadline`).

[23] https://www.hotstorage.org/2023/papers/hotstorage23-final1.pdf
[24] https://www.phoronix.com/review/linux-56-nvme

Помните, что универсального рецепта не существует, и всегда следует выполнить собственные тесты и бенчмарки (например при помощи программы KDiskMark), чтобы понять какой из планировщиков вам подходит лучше.

1.11.2.7 Увеличение размера карты памяти процесса

Так как виртуальные страницы процесса представляют собой кучу маленьких фрагментов его данных, то для удобства вся его виртуальная память разграничивается ядром на *зоны*. Например, в одной зоне памяти процесса может быть загружена библиотека `libc.so.6`, а в других зонах - бинарный код другой библиотеки или данные самой программы, память под которые она запросила у ядра через функцию `mmap`. Зон может быть несколько, так как они различаются по своим правам доступа (Да, права есть не только у файлов, но и у виртуальных страниц). Информация об этих зонах памяти процесса ядро хранит в так называемой *виртуальной карте памяти* (*memory map*). Здесь речь идёт вовсе не о носителе данных, а о той карте, которая используются ядром для того чтобы понимать, начиная с какого адреса в памяти процесса расположена та или иная зона. Вы можете просмотреть эту карту прочитав файл `maps` на псевдофайловой системе procfs в директории, которая предоставляет информацию о процессе с соответствующим ID.

Размер таких карт у каждого процесса ограничен значением параметра `vm.max_map_count` , которое указывает на максимальное количество записей, которое может хранится в карте у процесса. По умолчанию это значение равно `65530`[c. 50, 4]. К сожалению, современные программы, в особенности игры запускаемые через Wine/Proton, с их потреблением более 8 Гб на процесс, могут запрашивать чрезмерно много виртуальных страниц у ядра, из-за чего количество зон для их процессов начинает превышать установленный лимит по умолчанию. Это приводит к тому, что ядро просто не даёт выделить ещё одну зону в памяти у процесса, что в свою очередь приводит к проблемам со стабильностью (приложение может просто аварийно завершится) и производительностью в таких прожорливых программах. Поэтому если вы столкнулись с неполадками во время игры, такими как частые вылеты или микрофризы, не спишите сразу писать гневные письма разработчикам Wine, а попробуйте увеличить значение данного параметра.

Чтобы избежать всех этих потенциальных проблем, рекомендуется увеличить допустимый размер карты памяти процесса, то есть значение параметра sysctl `vm_max_map_count` до `1048576`. Тогда памяти хватит точно всем :)

Список 37: `sudo nano /etc/sysctl.d/99-sysctl.conf`

```
vm.max_map_count = 1048576
```

Примечание

Во многих дистрибутивах., например таких как Fedora и Arch Linux, данное значение уже установлено по умолчанию[2526] поэтому пользователям данных дистрибутивов не нужно делать никаких дополнительных действий.

1.11.2.8 Отключение многоступенчатого включения дисков

Для классических жестких дисков и иных носителей, подключаемых через интерфейс SATA, существует механизм по многоступенчатому включению питания, называемый *Staggered spin-up* (сокращенно - *SSU*), суть которого состоит в том, что питание всех носителей у вас в системе включается последовательно. То есть сначала начинает свою работу диск №1, потом диск №2 и так далее. SSU был разработан и включен в спецификацию SATA 2.5 достаточно давно - в 2005 году, когда компьютеры были слишком слабыми и не выдерживали одновременного включения всех дисков при загрузке системы.

В современных реалиях никакой необходимости в использовании SSU больше нет, так как современные блоки питания спокойно выдерживают пиковое энергопотребление, вызванное включением 3-4х и более носителей одновременно, каждый из которых как правило имеет пиковую нагрузку всего в 5W-12W[27], в то же время

[25] https://pagure.io/fesco/issue/2993
[26] https://archlinux.org/news/increasing-the-default-vmmax_map_count-value/
[27] https://superuser.com/questions/565653/how-much-power-does-a-hard-drive-use

использование SSU по умолчанию приводит к снижению скорости загрузки системы, поэтому он однозначно рекомендуется к отключению на всех современных системах, если вы имеете блок питания, который покрывает потребление вашего железа не "впритык", а хотя бы с небольшим запасом.

В Linux это можно сделать через использование параметра `ignore_sss=1` для модуля ядра `libahci`, отвечающего за реализацию общих функций по управлению всеми ATA устройствами. Для этого следует создать новый файл в директории `/etc/modprobe.d`:

Список 38: `sudo nano /etc/modprobe.d/30-ahci-disable-sss.conf`

```
options libahci ignore_sss=1
```

После этого на системах под управлением дистрибутива Arch Linux или другого, основанного на нем, требуется также выполнить обновление образов initramfs, чтобы данная конфигурация для `modprobe` стала их частью и была применена на этапе ранней загрузки системы:

```
sudo mkinitcpio -P
```

Примечание

Вы можете пропустить шаги из данного раздел, если имеете всего один жесткий носитель или не имеете их вообще, так как данный параметр не оказывает влияния на устройства, подключенные не через SATA.

1.11.3 Оптимизация работы CPU

Нельзя прямо или косвенно улучшить характеристики вашего процессора, однако можно использовать более эффективное и отвечающее вашим задачам планирование работы процессов, а также отключить некоторые лишние возможности ядра, которые могут приносить больше вреда, чем пользы для простого пользователя. В этом разделе пойдет речь о том, как добиться более эффективного использования ресурсов вашего "камня" с целью получить более хорошую производительность.

1.11.3.1 Масштабирование частоты CPU

В Linux существует специальные модули ядра, так называемые драйверы масштабирования CPU. Они корректируют частоту вашего процессора в режиме реального времени в зависимости от используемой политики (`governor`) масштабирования частот процессора. Для x86 архитектуры стандартным таким драйвером в ядре Linux принято считать `acpi-cpufreq`, который, как понятно из названия, осуществляет регулировку путем специальных вызовов APCI, которые устанавливают так называемый P-state (Peformance state) уровень для вашего процессора. Уровни P-State вашего процессора можно сравнить с коробкой передач у автомобиля: он может лететь на всей скорости, ехать с обычной автомобиля: он может лететь на всей скорости, ехать с обычной скоростью, и стоять на месте в ожидании того, когда надо будет сдвинуться с места. Собственно, по этой аналогии `acpi-cpufreq` и переключается между 3-х стандартных P-State уровней в зависимости от используемой политики масштабирования.

1.11.3.1.1 Политики масштабирования частоты

Вернемся к политикам масштабирования частоты. Их можно сравнить с планами электропитания в настройках Windows, только в отличии от них политик масштабирования в Linux довольно много: `performance`, `powersave`, `userspace`, `ondemand`, `conservative`, `schedutil`.

Рассмотрим каждую из них подробнее:

`performance` - как понятно из названия, данная политика используется для достижения максимальной производительности, так как она сильно снижает порог нагрузки, переходя через который процессор

начинает работать на полную мощность. Хорошо подходит для настольных ПК, но не слишком желательно для использования на ноутбуках, где важна автономность. Обратите внимание, что поведение данной политики зависит от используемого драйвера масштабирования, об этом подробнее читайте далее в разделе *Альтернативные драйверы масштабирования*.

`powersave` - полная противоположность `performance`, минимизирует потребление энергии через занижение частот процессора до минимума. Может быть очень полезно для ноутбуков при работе от батареи. Обратите внимание, что поведение данной политики зависит от используемого драйвера масштабирования, об этом подробнее читайте далее в разделе *Альтернативные драйверы масштабирования*.

`ondemand` - политика, которая регулирует частоту процессора на основе общей нагрузки на процессор, то есть частота прямо пропорциональна нагрузке: Чем выше нагрузка, тем больше частота, и наоборот, чем ниже, тем ниже и частота. Конечно, скорость возрастания частоты и нагрузки не всегда коррелируют между собой, ибо для принятия решения о том, когда нужно повышать частоту до максимальных значений, драйвер руководствуется значением параметра `up_threshold` (по умолчанию 80%), которое устанавливает порог максимальной нагрузки в процентах, которое должно достичь хотя бы одно из ядер вашего процессора. К примеру, если у вас есть два ядра, и в текущий момент времени драйвер масштабирования зафиксировал на одном ядре процессора нагрузку в 70%, а на другом в 81%, то он станет повышать их частоты в соответствии со значением `up_threshold` по умолчанию. Обратное решение - понижение частоты CPU, драйвер принимает в соответствии со значением другого параметра - `down_threshold`, которое также устанавливает порог ниже которого должно пройти хотя бы одно ядро процессора, тогда частота будет понижена. Во всех остальных случаях частота будет регулироваться как обычно, то есть пропорционально общей нагрузке.

Довольно хороший выбор для большинства конфигураций и задач. Рекомендуется к использованию.

`conservative` - должно быть вы замечали, как запуская игру или "тяжёлое" приложение ваш компьютер или ноутбук начинает гудеть как самолет, так как происходит резкое повышение частот процессора и следовательно растет его температура. `conservative` очень похож на политику `ondemand`, но он делает процесс увеличения частоты CPU более "гладким" и поступательным даже при значительном повышении нагрузки. Это может быть очень полезным, когда вы не хотите чтобы ваш ноутбук резко повышал свою температуру или уходил в так называемый "турбобуст".

`schedutil` - регулирует частоту процессора на основе метрик планировщика CPU, например таких как количество активных процессов на ядро процессора. Не слишком рекомендуется в силу того, что использование данной политики часто приводит к резким скачкам частоты без необходимости. Несмотря на это, данная политика используется по умолчанию в стандартном ядре Arch Linux.

`userspace` - данная политика является заглушкой, которая позволяет передать полномочия управления частотой с драйвера масштабирования на программу, запущенную с правами root, в пространстве пользователя, которая и будет осуществлять процесс регулировки частоты в соответствии с собственной логикой. В большинстве случаев вам никогда не понадобиться эта политика.

1.11.3.1.2 Альтернативные драйверы масштабирования

Современные процессоры Intel и AMD могут самостоятельно осуществлять масштабирование своей частоты на основе информации получаемой через механизм *CPPC* (Collaborative Processor Performance Control), с которым процессор получает от драйвера масштабирования "подсказку", о том какой уровень производительности следует выдавать в данный момент. Специально для работы с этим механизмом были созданы драйверы amd-pstate (поддерживается процессорами Zen 2 и выше) и intel-pstate (поддерживается Sandy Bridge и выше) для процессоров AMD и Intel соответственно.

Примечание

Для процессоров Intel термин CPPC обычно не используется, так как их технология для автономного управления частотой процессора называется HWP (Hardware P-states), но суть остается той же, что и в случае с CPPC.

У драйверов P-state существует несколько режимов работы. Как для intel-pstate, так и для amd-pstate есть `active` и `passive` режим, но для amd-pstate есть также `guided` режим.

В режиме `active`, который используется по умолчанию во всех P-state драйверах, управление частотой выполняется полностью автономно самим процессором, но он получает от драйвера масштабирования "подсказку" - так называемый уровень `energy_performance_preference` (далее EPP), на основе которого процессор понимает с каким уклоном ему регулировать собственную частоту. Таких уровней всего пять: `power`, `balance_power`, `default`, `balance_performance`, `performance`. Как понятно из названия, эти уровни указывают процессору предпочтение к тому, чтобы он работал на максимальную мощность (при использовании уровней `balance_performance` и `performance`) или наоборот экономил энергию и чаще принимал решение о понижении своей частоты или уходе в состояние сна. По умолчанию используется `default`, что представляет собой баланс между максимальной производительностью и энергосбережением.

Важно отметить, что классические политики для управления частоты, которые мы описывали ранее, отходят на второй план, и более того, в режиме `active` вы сможете выбрать всего две "фиктивные" политики масштабирования, это `powersave` и `performance`. Обе из них не оказывают того влияния на частоту процессора, которое мы приписывали им ранее, так как в режиме `active` драйвер не может самостоятельно устанавливать частоту процессора и теперь это зависит только от используемого значения EPP. Поэтому при выборе `performance` политики вы на самом деле просто измените текущий уровень EPP на `performance`, значение которого P-state драйвер передаст процессору через специальный регистр. Но при переключении политики на `powersave` уровень EPP не измениться и вы должны будете установить его самостоятельно (об этом читайте далее).

При использовании режима `passive` P-State драйвер может напрямую устанавливать желаемый уровень производительности, в связи с чем в нем доступен полный набор политик масштабирования, о которых мы говорили ранее. При этом установить уровень EPP становится невозможно, так как процессор уже не управляет частотой полностью самостоятельно, а ожидает переключения уровня P-State со стороны драйвера масштабирования. Данный режим отличается от использования классического драйвера `acpi-cpufreq` тем, что драйвер переключается не между 3-мя уровнями P-State, которые определены стандартом ACPI, а между сразу всеми доступными диапазонами частоты для вашего процессора, что гораздо эффективнее.

Для драйвера `amd-pstate` существует также третий режим работы - `guided`. Он работает аналогично режиму `active`, позволяя процессору самому управлять частотой, но при этом драйвер может устанавливать процессору пороги минимальной и максимальной частоты, что позволяет использовать классические политики масштабирования как в случае с `passive` режимом.

Переключение между всеми тремя режимами может быть осуществлено как при помощи соответствующих параметров ядра `amd_pstate` (например, `amd_pstate=guided`) или `intel_pstate` в зависимости от используемого драйвера масштабирования, так и прямо во время работы системы при помощи файла `status` на псевдофайловой системе sysfs:

AMD

```
echo "passive" | sudo tee /sys/devices/system/cpu/amd_pstate/status
```

AMD (cpupower 6.14+)

```
sudo cpupower set --amd-pstate-mode passive
```

Intel

```
echo "passive" | sudo tee /sys/devices/system/cpu/intel_pstate/status
```

1.11.3.1.3 Настройка параметров масштабирования

Перейдем от теории к практике. Чтобы изменить текущую политику масштабирования частоты можно воспользоваться множеством различных способов, начиная от способа "руками" при помощи sysfs, заканчивая специализированными утилитами как `cpupower` и `power-profiles-daemon`, которые мы и будем использовать для удобства. Для начала установим программу `cpupower`:

```
sudo pacman -S cpupower
```

С её помощью мы можем быстро увидеть информацию о текущей политике масштабирования, используемом драйвере, а также текущую частоту:

```
cpupower frequency-info
```

Установить желаемую политику масштабирования можно через команду `frequency-set`. К примеру, установим политику `performance`:

```
sudo cpupower frequency-set -g performance
```

Примечание

Если команда `cpupower frequency-info` указывает на то, что используется P-State драйвер в автономном режиме, то не следует пытаться применять классические политики масштабирования, вместо этого нужно указывать значение параметра `energy_performance_preference` (EPP) при помощи команды `cpupower set --epp`, например:

```
sudo cpupower set --epp balance_performance
```

Узнать доступные значения параметра EPP можно через:

```
cat /sys/devices/system/cpu/cpu0/cpufreq/energy_performance_available_preferences
```

Это значение будет той самой подсказкой для процессора, о которой мы говорили выше, в соответствии с которой он будет осуществлять самостоятельный контроль своей частоты.

Если вы хотите ограничить максимальную частоту процессора, то вы можете использовать ключ `-u`:

```
# Ограничит максимальную частоту в 3 ГГц
sudo cpupower frequency-set -u 3.0Ghz
```

Все проделанные изменения выше работают только до перезагрузки системы, чтобы их сохранить нам понадобиться одноименный демон `cpupower.service`:

```
sudo systemctl enable --now cpupower
```

А также изменить конфигурацию `/etc/default/cpupower`, которая содержит все применяемые при запуске системы настройки. К примеру, если вам нужно изменить политику масштабирования на постоянной основе, то нужно указать значение параметра `governor` внутри `/etc/default/cpupower`:

Список 39: `sudo nano /etc/default/cpupower`

```
governor='conservative'
```

Примечание

По умолчанию все строки в файле закоментированы. Чтобы раскоментировать нужные параметры уберите знак # в начале строки.

1.11.3.2 Отключение сторожевых таймеров

Сторожевые таймеры (*watchdog timer*) - это аппаратные или программные средства, которые отслеживают зависания системы во время её работы. Все такие таймеры работают по принципу "пуллинга", при котором система должна постоянно выполнять прерывания таймера. Если прерывание не было сделано вовремя, то подразумевается, что система зависла и таймер по истечению времени ожидания должен сигнализировать другое устройство или программу (если речь о программном сторожевом таймере) о зависании, так чтобы были предприняты действия по автоматическому восстановлению работы системы, что может включать в себя её перезапуск или другие процедуры в зависимости от конкретной системы.

На первый взгляд это звучит очень здорово, однако стоит понимать, что сторожевые таймеры задумывались в основном для систем, которые должны работать полностью автономно без прямого участия со стороны пользователя. К ним относятся например сервера, SoC платы для встраивания в различные малые или крупные аппаратные комплексы (вплоть до таких систем как космические зонды) и т. д. На простых домашних ПК или ноутбуках у пользователя всегда есть возможность вручную выполнить перезапуск системы, не говоря о том, что как правило зависания достаточно мощных систем происходит редко и в основном из-за ошибок внутри самого ядра, а на маломощных в основном из-за ситуаций OOM (нехватки памяти).

Так как на домашних системах сторожевые таймеры большую часть времени выполняют только холостую работу, их отключение позволяет снизить общее энергопотребление[28] и увеличить производительность системы[29]. Сделать это можно как и всегда через соответствующий параметр sysctl:

Список 40: `sudo nano /etc/sysctl.d/30-no-watchdog-timers.conf`

```
kernel.watchdog = 0
```

К сожалению, на современных процессорах от AMD и Intel использование данного параметра ядра недостаточно для полного отключения аппаратных сторожевых таймеров[30], которые являются частью чипсета процессора. Поэтому дополнительно требуется также запретить загрузку модулей ядра, которые занимаются управлением этих таймеров. Для процессоров от Intel это `iTCO_wdt`, а для AMD `sp5100-tco`. Чтобы запретить их загрузку нужно создать файл в директории `/etc/modprobe.d` и внести модули в чёрный список:

Список 41: `sudo nano /etc/modprobe.d/30-blacklist-watchdog-timers.conf`

```
blacklist sp5100-tco
blacklist iTCO_wdt
```

На системах под управлением дистрибутива Arch Linux или другого, основанного на нем требуется, также выполнить обновление образов initramfs, чтобы данная конфигурация для `modprobe` стала их частью и была применена на этапе ранней загрузки системы:

[28] https://wiki.archlinux.org/title/Power_management#Disabling_NMI_watchdog
[29] https://bbs.archlinux.org/viewtopic.php?id=163768
[30] https://bbs.archlinux.org/viewtopic.php?id=165834

```
sudo mkinitcpio -P
```

После перезагрузки системы можно удостовериться, что сторожевые таймеры были успешно отключены через команду `wdctl`, она должна вывести ошибку об отсутствии соответствующих устройств.

1.11.3.3 Отключение защиты от уязвимостей (по желанию)

Осторожно

Данный раздел носит сугубо информационный характер и ни к чему не призывает. Безопасность системы является таким же важным её аспектом как и производительность, однако для тех, кто все же хочет "выжать максимум" несмотря на связанные с этим риски, и представлен материал ниже.

К сожалению современные процессоры (если говорить об архитектуре x86) обзавелись достаточно большим багажом аппаратных уязвимостей, которые приходится различными способами исправлять разработчикам ядра. И как правило все такие исправления влекут за собой большие потери производительности. Иногда такие исправления приводят к потерям порядка ~27-28% производительности[31][32], а иногда и все 50%[33], как в случае с нашумевшей уязвимостью Spectre, что часто приводит к недоумению относительно целесообразности таких исправлений не только со стороны простых пользователей, но и со стороны других разработчиков ядра[34].

Здесь стоит отметить, что те же уязвимости Spectre и Meltdown сами по себе не являются "прямым ключом" к вашей системе, так как даже при их использовании скорость утечки в продемонстрированных ранее примерах эксплуатации составляет всего 1 Кб/с. Кроме того, большинство современных браузеров имеет свою встроенную систему защиты для запуска всех критически важных процессов по обмену данными в изолированном окружении, которое предотвращает эксплуатацию подобных уязвимостей, хотя некоторые браузеры, например такие как Firefox, в последних версиях отказываются от включения по умолчанию частей такой дополнительной защиты в пользу лучшей производительности[35], в частности на практике использование уязвимостей класса Spectre v2 имеет слабый практический характер, а также такая защита может быть не нужна из-за включенной по умолчанию в ядре общей защиты против уязвимостей класса Spectre. Если вы хотите форсировать включение этой дополнительной защиты в Firefox вдобавок к уже существующим мерам защиты в ядре, то сделать это можно через переменную окружения `MOZ_SANDBOX_NO_SPEC_ALLOW=1`.

Если же вы все же решили выполнить полное отключение всей защиты в ядре в пользу большей производительности, то сделать это можно только при указании параметра загрузки ядра `mitigations=off`, который следует прописать в конфигурации вашего загрузчика:

Предупреждение

Отключение защиты с целью повышения производительности имеет смысл только на старых поколениях процессоров Intel и AMD. На самых последних линейках это может иметь даже негативные последствия[36].

GRUB 2

Чтобы изменить параметры по умолчанию к загрузке любого ядра при использовании загрузчика GRUB 2 нужно открыть файл `/etc/default/grub`, найти строку `GRUB_CMDLINE_LINUX_DEFAULT` и внутрь двойных кавычек дописать к уже имеющимся параметрам `mitigations=off` через пробел. Пример такой строки конфигурации:

[31] https://linuxreviews.org/HOWTO_make_Linux_run_blazing_fast_(again)_on_Intel_CPUs
[32] https://talawah.io/blog/extreme-http-performance-tuning-one-point-two-million/#_2-speculative-execution-mitigations
[33] https://lkml.org/lkml/2018/11/19/37
[34] https://lkml.org/lkml/2018/11/19/69
[35] https://hg.mozilla.org/mozilla-central/rev/ebdd80f675b6
[36] https://www.tomshardware.com/news/ryzen-7000-security-mitigations-faster-on-linux

Список 42: `sudo nano /etc/default/grub`

```
...
GRUB_CMDLINE_LINUX_DEFAULT="quiet loglevel=3 mitigations=off"
...
```

После чего необходимо обновить конфигурацию загрузчика:

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

rEFInd

Если вы используете загрузчик rEFInd вместе со всеми его возможностями по автоматическому определению всех установленных версией ядра у вас в системе, то чтобы добавить параметр `mitigations=off` нужно отредактировать файл `/boot/refind_linux.conf` и в первую строку содержащую дописать данный параметр внутрь вторых двойных кавычек. Пример такого файла конфигурации (не копировать полностью!):

Список 43: `sudo nano /boot/refind_linux.conf`

```
"Boot to default params"    "root=PARTUUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX rw␣
↪mitigations=off"
"Boot to single-user mode"    "root=PARTUUID=XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX rw␣
↪single"
```

systemd-boot

К сожалению при использовании загрузчика systemd-boot нельзя указать общие для всех установленных ядер в системе параметры загрузки, поэтому приходится указывать их в отдельности для каждой версии ядра. Например, чтобы добавить параметр `mitigations=off` для обычного ядра используемого в Arch Linux, вам нужно найти файл внутри директории `/boot/efi/entries`, который содержит строку `linux  /vmlinuz-linux` и ниже внутри этого файла дописать через пробел параметр в строке с `options`. Пример конфигурации:

Список 44: `sudo nano /boot/efi/loader/entries/arch.conf`

```
title   Arch Linux
linux   /vmlinuz-linux
initrd  /initramfs-linux.img
options root=UUID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx rw mitigations=off
```

Предупреждение

Путь до файла конфигурации может отличаться в зависимости от того в какую директорию вы смонтировали загрузочный раздел, некоторые пользователи вместо `/boot/efi` используют точку монтирования `/efi`. Узнать используемую точку монтирования можно через команду `bootctl --print-esp-path`.

1.12 Файловые системы

1.12.1 Нюансы выбора файловой системы и флагов монтирования

В отличие от Windows, в Linux-подобных системах выбор файловой системы не ограничивается в обязательном порядке со стороны дистрибутива, и может применяться исходя из личных предпочтений пользователя с оглядкой на поддержку со стороны ядра.

В Linux файловые системы можно разделить на два типа:

- Обычные: ext4, F2FS, XFS.
- Системы на базе CoW (Copy-on-Write): Btrfs, bcachefs и другие.

1.12.2 Обычные системы

Эти системы как правило согласно тестам показывают более высокие скорости операций ввода-вывода (IOPS) и скорости записи/чтения, но в тоже время могут не обладать некоторым дополнительным функционалом, например сжатие "налету", быстрые бэкапы и прочее.

1.12.2.1 Файловая система ext4

Данная система уже давно зарекомендовала себя как простое универсальное решение. Это своего рода как *NTFS* в Windows и если вы не уверены какую файловую систему вам выбрать под свои задачи, то можно использовать **EXT4** - особо ничего не потеряете, особенно если вы начинающий пользователь.

EXT4 по умолчанию используется во многих дистрибутивах Linux. Подходит как для жестких дисков (HDD), так и для SSD, хоть для последних есть и другие варианты.

EXT4 при должных оптимизациях показывает себя лучше BTRFS на жёстких дисках.

1.12.2.1.1 Настройка внешнего журнала ext4

Вынесение журнала файловой системы на более быстрый накопитель (желательно SSD) является безопасным и очень эффективным способом приблизиться к развитию потенциала жёсткого диска. Результаты простых бенчмарков ты можешь найти тут.

Размер внешнего журнала должен определяться в зависимости от предполагаемой нагрузки на диски, но, в целом, рекоммендуемым значением для накопителей от 1 ТБ является 1-2 ГБ.

Для внешнего журнала тебе понадобится создать отдельный раздел нужного размера, а после отформатировать его командой:

```
mke2fs -O journal_dev /dev/nvme0n1p2
```

А после подключить этот журнал к существующей файловой системе:

```
tune2fs -J device=/dev/nvme0n1p2 /dev/sda
```

1.12.2.1.2 Оптимальные параметры монтирования для ext4

Если вы используете внешний журнал, то оптимальными параметрами будут:

```
relatime,commit=300,journal_async_commit,nodelalloc,data=journal
```

Если нет, то относительно безопасным вариантом будет:

```
relatime,commit=300,data=ordered
```

В последнем параметр `data=ordered` можно заменить на `data=writeback`, что отключит использование журнала вовсе, но может вызывать потери данных или их повреждение.

Определение всех параметров монтирования ты можешь найти тут.

Внимание

Параметр `commit=300` может вызывать повышенное потребление оперативной памяти и портить данные на диске. Можете установить значение на более консервативное -- например, `commit=100`.

Прописывать их нужно в файл `/etc/fstab` для нужных разделов.

1.12.2.2 Файловая система XFS

Высокопроизводительная файловая система, которая отлично подходит для HDD. Благодаря особенностям архитектуры данная система отлично справляется с задачами хранения различных данных. Данная система разрабатывалась в первую очередь под HDD, однако она также не плохо справляется с SSD, особенно когда дело касается базы данных.

1.12.2.3 Файловая система F2FS

Файловая система разработанная компанией Samsung как уверяется специально для NAND накопителей (SSD, флешки и т.п.). Иногда может выделяться немного большими показателями IOPS и скоростью линейного чтения, но большими задержками по сравнению с той же **EXT4** (от 0,5 мс и доходящих до пятикратной разницы). Поддерживает функцию сжатия "налету".

1.12.3 Системы CoW

Файловые системы CoW (или копирование при записи) примечательны тем что они не создают дубликат файла при его копировании, а обновляют метаданные, тем самым экономя место на носителе, поэтому данные системы при создании backup-ов экономят время, поскольку backup происходит практически мгновенно и свободное место.

Также данные системы поддерживают разного рода дополнительные функции, например возможность сжатия файловой системы "налету". Ценой дополнительной нагрузки на CPU можно сжимать записываемые данные на носитель, тем самым экономя место и его ресурс, что может быть особенно полезно при использовании SSD. Однако стоит отметить, что далеко не все файлы хорошо поддаются сжатию, также зачастую страдает скорость записи данных на диск, особенно при высоких уровнях сжатия.

1.12.3.1 Файловая система btrfs

Данная файловая система выделяется большим функционалом и возможностями: хотите две системы на одном ssd без разделения SSD или изменения таблицы разметки - для вас есть подтома (subvol), хотите экономить место при записи на свой SSD - можете использовать сжатие на лету (compress), нужно быстро сделать бэкап раздела или всей системы - не вопрос вам также помогут (subvol и snapshot).

Данная система может проигрывать другим файловым системам по скорости операций ввода-вывода (IOPS) и скорости чтения/записи в связи с системой CoW, а также наличию фрагментации, что может быть критично для HDD. CoW можно отключить, однако тогда **btrfs** потеряет практически весь свой функционал и уже проще использовать туже **EXT4**.

По умолчанию используется например в Fedora, SteamOS (Steam Deck).

1.12.3.1.1 Оптимальные флаги монтирования

Вот оптимальные параметры для SSD носителей. Описание каждого из них вы можете найти - здесь.

Предупреждение

ВАЖНО! Следующие параметры работают только с Btrfs, кроме `relatime` и `commit`.

```
rw,relatime,ssd_spread,max_inline=256,commit=300
```

Оптимальные параметры монтирования для HDD:

```
rw,relatime,max_inline=256,commit=300,autodefrag
```

Прежде всего, отметим, что вы можете изменить *relatime* на *noatime* или *lazytime* - все три параметра отвечают за запоминание времени доступа к файлами и прочим связанным с ним атрибутами, что только портит отклик.

Параметр noatime полностью выключает данную функцию, что может привести к некоторым багам в приложениях зависимых от времени (например git), но автор никогда не встречал данной проблемы. Параметр *lazytime* успешно будет выполнять все функции времени, но выполнять их промежуточную запись в оперативной памяти, что позволит избежать замедления без потери функционала, но говорят *lazytime* чудит, поэтому автор советует *noatime*.

Но если вы хотите минимум возможных проблем, то оставьте флаг *relatime*.

Рекомендуется минимизировать использование CoW в Btrfs для HDD во избежание излишней фрагментации. Но автор рекомендует делать это выборочно, то есть не использовать опцию монтирования `nodatacow`. Полное отключение CoW по сути лишает Btrfs всех присущих преимуществ, а также ведет к отключение проверок хэш-сумм файлов, что может привести к поломкам и повреждению файловой системы. Поэтому мы должны использовать специальный флаг файловой системы, чтобы отключить его для отдельных директорий/файлов. Это могут быть:

1) Директория для торрентов. Автор крайне рекомендует выполнять загрузку всех торрентов в отдельную директорию и отключать для неё CoW во избежание фрагментации:

```
sudo chattr -R +C ~/Torrents # Укажите свою директорию с торрентами
```

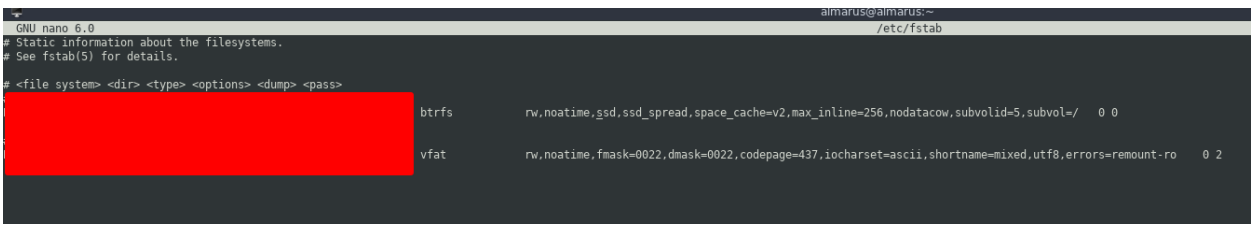
2) Файлам баз данных очень чувствительны к фрагментации при CoW. Как правило все они расположены в директории `/var/db`, но часть пользовательских приложений любит их создавать в `~/.local/share`:

```
sudo chattr -R +C /var/db
sudo chattr -R +C ~/.local/share
```

3) Для образов виртуальных машин также не должен использоваться CoW, иначе это приведет к фрагментации.

Только учтите, что изменения не применятся к уже существующим файлам, а только к новосозданным.

Прописывать их нужно в файл `/etc/fstab` для корневого и домашнего разделов. Некоторые из данных флагов будут применяться только для новых файлов.



Внимание

Параметр `commit=600` может вызывать повышенное потребление оперативной памяти и портить данные на диске. Можете установить значение на более консервативное -- например, `commit=100`.

Внимание

При использовании Btrfs для корневого раздела, обязательно установите пакет btrfs-progs.

1.12.3.1.2 Сжатие в файловой системе Btrfs

В файловой системе Btrfs есть возможность включения сжатия. Все записываемые файлы по возможности будут сжиматься и экономить пространство на носителе HDD или SSD.

Для SSD это может быть важно в связи с их ограниченным ресурсом на запись.

Согласно wiki Btrfs, официально имеется 3 поддерживаемых алгоритма:

- zlib - высокая степень сжатия, но низкая скорость сжатия и распаковки
- lzo - высокая скорость сжатия и распаковки, но наименьший уровень сжатия из представленных алгоритмов
- zstd - степень сжатия сравнимая с zlib и более быстрые сжатие и распаковка, однако уступающие по скорости lzo

Для включения алгоритма сжатия в файловой системе необходимо:

1. Убедиться в наличии необходимого алгоритма в системе или установить выбранный (zlib, lzo или zstd соответственно).
2. Отредактировать файл `etc/fstab`, добавив для необходимого раздела или носителя следующий флаг монтирования:

```
compress='алгоритм':N
```

(Где `N` - степень сжатия: для zlib - `N` = 1,2,...9; для lzo - выбор уровня сжатия не предусмотрен, поэтому `:N` - не указываются; для zstd - `N` =1,2,...15. Чем выше данный параметр, тем сильнее будут сжиматься данные, конечно при условии что это возможно, но также будет повышена нагрузка на процессор, поскольку сжатие выполняется за счет его ресурсов. Согласно wiki Btrfs, оптимальным значением `N` по отношению `степень сжатия / скорость` считается `3`)

Например для zstd со степенью сжатия 3 запись будет выглядеть примерно следующим образом, если учесть приведенные выше флаги монтирования:

```
rw,relatime,compress=zstd:3,ssd_spread,max_inline=256,commit=600
```

После выставления данного флага монтирования новые файлы начнут сжиматься при записи на диск. Для сжатия уже имеющихся данных необходимо выполнить команду:

```
sudo btrfs filesystem defragment -calg /path
```

(Где `-calg` - алгоритм (указывается как czlib, clzo или czstd в зависимости от выбранного алгоритма), `path` - путь к разделу или папке)

Для сжатия уже существующих данных в папке или целого раздела необходимо указать ключ `-r` перед `-calg`:

```
sudo btrfs filesystem defragment -r -calg /path
```

(Где `path` - путь к разделу или папке)

> **Внимание**
>
> Степень сжатия в данном случае не указывается!

1.12.3.1.3 Определение эффективности сжатия

Если вы хотите определить эффективность сжатия на вашем разделе/диске, то вам необходимо воспользоваться программой compsize. Установить ее можно с помощью команды:

```
sudo pacman -S compsize
```

Для выполнения проверки на эффективность необходимо использовать команду:

```
sudo compsize /path
```

(Где `path` - путь к разделу, папке или файлу)

Пример вывода команды:

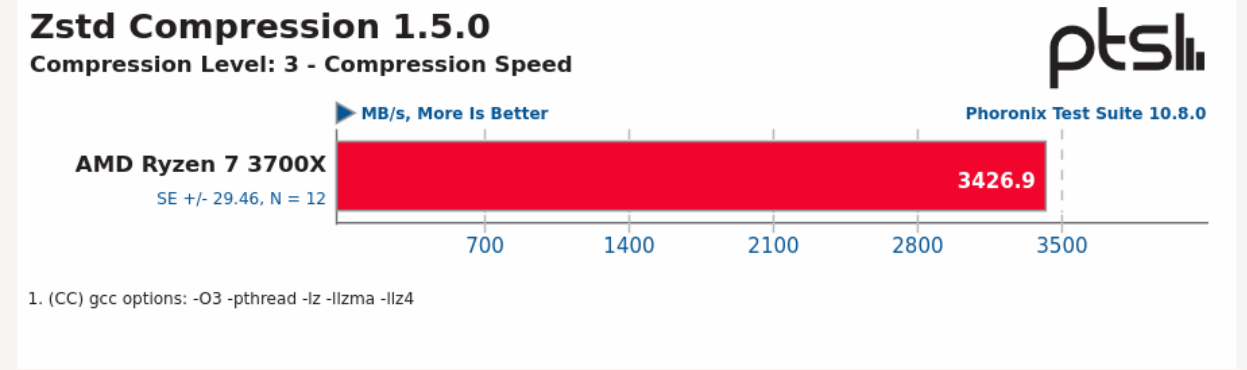
```
Processed 309409 files, 1592042 regular extents (1599469 refs), 56369 inline.
Type       Perc     Disk Usage   Uncompressed Referenced
TOTAL       86%      645G         742G         742G
none       100%      619G         619G         619G
zstd        21%       26G         123G         123G
```

Пояснения:

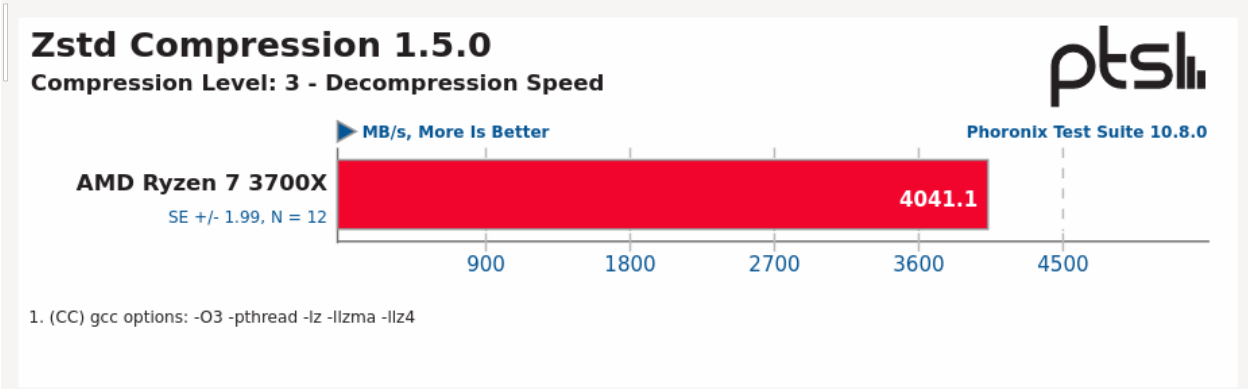
- **Первый столбец:**
  - Строка *TOTAL* - итоговые данные, которые учитывают все сжатые и не сжатые файлы и разные алгоритмы (если такие имеются).
  - Строка *none* - данные, которые не были сжаты.
  - Далее отображаются все использованные алгоритмы (в данном случае - zstd).
- Второй столбец показывает данные в процентах.
- Третий столбец отображает фактически использованное место на диске/разделе.
- Четвертый столбец показывает данные без сжатия.
- Пятый - видимый размер файла, тот, который зачастую отображается в системе.

1.12.3.1.4 Скорость обработки алгоритма zstd на примере AMD Ryzen 7 3700X

Для сравнения степеней сжатия алгоритма zstd использовалась бенчмарк платформа phoronix-test-suite. В данной программе, для проверки скорости сжатия и распаковки данных, доступно три степени - `3`, `8`, `19`. Для получения информации о падении скорости выполнения сжатия нам будет достаточно и первых двух, поскольку степень сжатия 19 на данный момент не поддерживается (однако данные также приведены для ознакомления), и если обратить внимание на полученные данные, то это и не имеет особого смысла. Далее представлены результаты замеров:

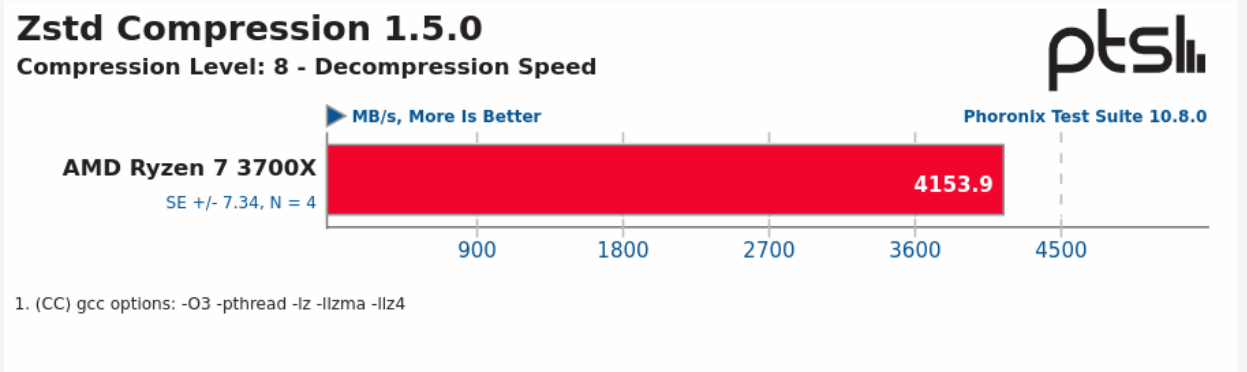


1. (CC) gcc options: -O3 -pthread -lz -llzma -llz4



1. (CC) gcc options: -O3 -pthread -lz -llzma -llz4

1. (CC) gcc options: -O3 -pthread -lz -llzma -llz4



1. (CC) gcc options: -O3 -pthread -lz -llzma -llz4

Как можно видеть из графиков, падение скорости при перехода от степени `3` к степени `8` сопровождается падением скорости сжатия более чем в **4,7** раз (не говоря о более высоких степенях сжатия) и практически не изменяется при выполнении распаковки, что может негативно сказаться на скорости установки программ и возможно в некоторых других ситуациях которые требует выполнения записи на диск.

Стоит отметить, что в случае выполнения установки игр с использованием степени сжатия `15`, было замечено повышение нагрузки на процессор вплоть до 72-75% в тех случаях, когда файлы поддавались сжатию.

1.12.3.1.5 Список протестированных игр на эффективность сжатия (Спасибо @dewdpol!)

Далее представлен список протестированных игр на сжатие в файловой системе Btrfs. Данные были получены с помощью программы compsize и являются округленными, поэтому информация может нести частично ознакомительный характер.

№	Игра	Алгоритм	Уровень сжатия	Необходимое место (N)	Используемое место(U)	U/N	Экономия
1	A Plague Tale: Innocence	zstd	3	41 GB	41 GB	99%	182 MB
			15				306 MB
2	A Story About My Uncle	zstd	3	1,1 GB	1,1 GB	94%	63 MB
			15			93%	74 MB
3	Aegis Defenders	zstd	3	1,3 GB	240 MB	17%	1,10 GB
			15		230 MB	16%	1,11 GB
4	Among Us	zstd	3	429 MB	284 MB	66%	145 MB
			15		279 MB	65%	150 MB
5	Aragami	zstd	3	7,6 GB	5,4 GB	71%	2,20 GB
			15		5,3 GB	69%	2,27 GB

продолжается на следующей странице

Таблица 3 – продолжение с предыдущей страницы

№	Игра	Алгоритм	Уровень сжатия	Необходимое место (N)	Используемое место(U)	U/N	Экономия
6	Armello	zstd	3	1,6 GB	1,5 GB	95%	73 MB
			15			94%	83 MB
7	Bastion	zstd	3	1,1 GB	1,1 GB	94%	67 MB
			15		1,0 GB	93%	81 MB
8	BattleBlock Theater	zstd	3	1,8 GB	1,7 GB	93%	117,8 MB
			15				118,7 MB
9	Beholder	zstd	3	1,9 GB	1,0 GB	55%	0,77 GB
			15		1,1 GB	58%	0,82 GB
10	Beholder 2	zstd	3	2,5 GB	2,2 GB	85%	385 MB
			15		2,1 GB	81%	483 MB
11	Blasphemous	zstd	3	854 MB	805 MB	94%	48 MB
			15		802 MB	93%	51 MB
12	Blue Fire	zstd	3	6,0 GB	4,9 GB	81%	1,10 GB
			15		4,7 GB	77%	1,30 GB
13	Brothers - A Tale of Two Sons	zstd	3	1,2 GB	1,1 GB	95%	53 MB
			15				52 MB
14	Castle Crashers	zstd	3	199 MB	183 MB	92%	15,4 MB
			15			91%	15,8 MB
15	Celeste	zstd	3	1,1 GB	897 MB	78%	251 MB
			15		871 MB	75%	277 MB
16	Child of light	zstd	3	2,3 GB	2,3 GB	99%	15 MB
			15				9,5 MB
17	Children of Morta	zstd	3	1,6 GB	1,5 GB	94%	87 MB
			15				92 MB
18	CODE VEIN	zstd	3	35 GB	35 GB	99%	75 MB
			15				124 MB
19	Cortex Command	zstd	3	97 MB	65 MB	67%	32 MB
			15		64 MB	66%	33 MB
20	Cuphead	zstd	3	3,6 GB	3,3 GB	93%	223 MB
			15				233 MB
21	Curse of Dead Gods	zsrd	3	2,7 GB	1,4 GB	53%	1,25 GB
			15			51%	1,29 GB
22	D-Corp	zstd	3	1,2 GB	720 MB	57%	525 MB
			15		697 MB	55%	549 MB
23	Dark Souls: Prepare To Die Edition	zstd	3	3,7 GB	3,7 GB	99%	1,57 MB
			15				1,61 MB
24	Dark Souls III	zstd	3	24 GB	24 GB	99%	0,53 MB
			15				0,60 MB
25	Darkest Dungeon	zstd	3	3,2 GB	2,8 GB	88%	394 MB
			15			87%	410 MB
26	Darkestville Catle	zstd	3	1,7 GB	798 MB	40%	0,99 GB
			15		682 MB	38%	1,02 GB
27	Darksiders III	zstd	3	24 GB	24 GB	99%	22 MB
			15				30 MB
28	Dead Cells	zstd	3	1,1 GB	1,1 GB	97%	24 MB
			15		1,0 GB		31 MB
29	Death's Door	zstd	3	3,6 GB	2,1 GB	58%	1,48 GB
			15			57%	1,54 GB

продолжается на следующей странице

Таблица 3 – продолжение с предыдущей страницы

№	Игра	Алгоритм	Уровень сжатия	Необходимое место (N)	Используемое место(U)	U/N	Экономия
30	Death's Gambit: Afterlife	zstd	3	1 GB	729 MB	66%	367 MB
			15		720 MB	65%	376 MB
31	Deponia: The Complete Journey	zstd	3	9,5 GB	9,5 GB	99%	24,2 MB
			15				25,6 MB
32	Devil May Cry 5	zstd	3	33 GB	33 GB	99%	82 MB
			15				86 MB
33	Disco Elysium	zstd	3	9,5 GB	9,1 GB	96%	305 MB
			15			95%	391 MB
34	Don't Starve Together	zstd	3	2,5 GB	1,8 GB	74%	651 MB
			15			73%	679 MB
35	Eldest Souls	zstd	3	1,0 GB	720 MB	69%	314 MB
			15		708 MB	68%	326 MB
36	Evergate	zstd	3	2,9 GB	1,9 GB	64%	1,01 GB
			15			63%	1,03 GB
37	Frostpunk	zstd	3	8,9 GB	8,9 GB	99%	24 MB
			15				25,2 MB
38	Furi	zstd	3	4,3 GB	2,7 GB	62%	1,53 GB
			15			63%	1,52 GB
39	Gato Roboto	zstd	3	440 MB	415 MB	94%	25,5 MB
			15		414 MB		26,1 MB
40	Gears Tactics	zstd	3	29 GB	29 GB	99%	66 MB
			15				97 MB
41	Ghost of a Tale	zstd	3	4,7 GB	3,7 GB	79%	0,90 GB
			15				0,94 GB
42	Ghostrunner	zstd	3	24 GB	20 GB	84%	3,7 GB
			15				
43	Gibbous - a Cthulhu Adventure	zstd	3	9,0 GB	4,2 GB	47%	4,76%
			15		4,1 GB	46%	4,87 GB
44	Gris	zstd	3	3,2 GB	1,5 GB	47%	1,70 GB
			15			46%	1,73 GB
45	Hades	zstd	3	11 GB	10 GB	95%	480 MB
			15				498 MB
46	Hand of Fate	zstd	3	2,5 GB	2,2 GB	90%	255 MB
			15			89%	287 MB
47	Hand of Fate 2	zstd	3	4,1 GB	4,1 GB	99%	35 MB
			15				38 MB
48	Hellblade: Sanua's Sacrifice	zstd	3	18 GB	16 GB	87%	2,3 GB
			15		18 GB	96%	693 MB
49	Helldivers	zstd	3	6,4 GB	6,4 GB	99%	25 MB
			15				27 MB
50	Hob	zstd	3	2,4 GB	2,2 GB	90%	230 MB
			15		2,1 GB	89%	250 MB
51	Hollow Knight	zstd	3	7,5 GB	1,5 GB	20%	5,87 GB
			15		1,4 GB	19%	5,98 GB

продолжается на следующей странице

Таблица 3 – продолжение с предыдущей страницы

№	Игра	Алгоритм	Уровень сжатия	Необходимое место (N)	Используемое место(U)	U/N	Экономия
52	Inmost	zstd	3	1,3 GB	649 MB	47%	709 MB
			15		638 MB		720 MB
53	Jotun	zstd	3	3,8 GB	1,8 GB	48%	1,91 GB
			15			49%	1,84 GB
54	Journey	zstd	3	3,3 GB	1,8 GB	55%	1,49 GB
			15		1,9 GB	56%	1,44 GB
55	Katana ZERO	zstd	3	216 MB	178 MB	82%	38 MB
			15		177 MB	81%	39 MB
56	Kate	zstd	3	254 MB	104 MB	40%	151 MB
			15		100 MB	39%	155 MB
57	Limbo	zstd	3	98 MB	97 MB	98%	1,7 MB
			15				1,8 MB
58	Little Nightmare	zstd	3	8,9 GB	5,8 GB	65%	3,1 GB
			15		4,8 GB	54%	4,1 GB
59	Loop Hero	zstd	3	140 MB	116 MB	83%	22,8 MB
			15		115 MB	82%	23,9 MB
60	Magicka	zstd	3	1,6 GB	1,6 GB	96%	68 MB
			15			95%	71 MB
61	Magicka 2	zstd	3	2,9 GB		99%	8,1 MB
			15		2,9 GB		8,7 MB
62	Mark of the Ninja: Remastered	zstd	3	7,5 GB	6,9 GB	92%	564 MB
			15				591 MB
63	Master of Anima	zstd	3	1,5 GB	1,2 GB	81%	292 MB
			15			80%	308 MB
64	METAL GEAR RISING: REVENGEANCE	zstd	3	24 GB	24 GB	99%	17,8 MB
			15				19,4 MB
65	Moonlighter	zstd	3	1,1 GB	577 MB	48%	608 MB
			15		572 MB		613 MB
66	Move or Die	zstd	3	666 MB	572 MB	85%	94 MB
			15		567 MB		99 MB
67	My Friend Pedro	zstd	3	3,5 GB	2,9 GB	82%	637 MB
			15			81%	666 MB
68	Nier:Automata	zstd	3	40 GB	37 GB	91%	3,5 GB
			15				3,3 GB
69	Nine Parchments	zstd	3	5,7 GB	5,7 GB	98%	68 MB
			15				78 MB
70	Ori and the Blind Forest: Definitive Edition	zstd	3	10 GB	4,9 GB	48%	5,3 GB
			15		4,7 GB	46%	5,5 GB
71	Ori and the Will of the Wisps	zstd	3	11 GB	5,5 GB	48%	5,8 GB
			15		5,3 GB	46%	6,1 GB
72	Othercide	zstd	3	6,0 GB	5,9 GB	98%	94 MB
			15				113 MB
73	Out of Line	zstd	3	1,3 GB	497 MB	37%	836 MB

продолжается на следующей странице

Таблица 3 – продолжение с предыдущей страницы

№	Игра	Алгоритм	Уровень сжатия	Необходимое место (N)	Используемое место(U)	U/N	Экономия
			15		476 MB	35%	857 MB
74	Outland	zstd	3	675 MB	593 MB	87%	82 MB
			15		589 MB		86 MB
75	Overcooked! 2	zstd	3	7,9 GB	7,7 GB	98%	161 MB
			15			97%	169 MB
76	Papers, Please	zstd	3	58 MB	45 MB	77%	13 MB
			15		44 MB	76%	13,6 MB
77	Path of Exile	zstd	3	27 GB	27 GB	99%	27 MB
			15				29 MB
78	Peace, Death!	zstd	3	83 MB	76 MB	91%	7,2 MB
			15				7,5 MB
79	Peace, Death! 2	zstd	3	34 MB	26 MB	78%	7,04 MB
			15				7,51 MB
80	Pummel Party	zstd	3	2,1 GB	1,4 GB	67%	712 MB
			15			66%	723 MB
81	Remember Me	zstd	3	6,7 GB	6,6 GB	99%	57 MB
			15				58 MB
82	Rocket League	zstd	3	18 GB	18 GB	99%	20 MB
			15				46 MB
83	RUINER	zstd	3	10 GB	10 GB	99%	50 MB
			15				77 MB
84	Salt and Sanctuary	zstd	3	563 MB	540 MB	95%	23 MB
			15				24 MB
85	Samorost 1	zstd	3	68 MB	68 MB	99%	19 KB
			15				23 KB
86	Samorost 2	zstd	3	141 MB	141 MB	99%	1,22 MB
			15		140 MB	98%	1,33 MB
87	Samorost 3	zstd	3	1,1 GB	1,0 GB	99%	9,5 MB
			15			96%	43 MB
88	Sekiro: Shadow Die Twice	zstd	3	13 GB	13 GB	99%	1,5 MB
			15				1,6 MB
89	Severed Steel	zstd	3	4,0 GB	2,7 GB	68%	1,22 GB
			15			67%	1,26 GB
90	Shadow Tactics: Blades of the Shogun	zstd	3	7,3 GB	5,0 GB	69%	2,2 GB
			15		4,8 GB	66%	2,5 GB
91	Shadowrun Returns	zstd	3	2,8 GB	1,1 GB	39%	1,68 GB
			15		1,0 GB	37%	1,74 GB
92	Shattered - Tale of the Forgotten King	zstd	3	6,3 GB	6,3 GB	99%	14,7 MB
			15				15,7 MB
93	Shiro	zstd	3	80 MB	74 MB	91%	6,5 MB
			15		73 MB		6,7 MB
94	Skul: The Hero Slayer	zstd	3	1016 MB	1001 MB	98%	14,5 MB
			15		987 MB	97%	29 MB
95	SpeedRunners	zstd	3	662 MB	651 MB	98%	11 MB
			15		650 MB		12 MB

продолжается на следующей странице

Таблица 3 – продолжение с предыдущей страницы

№	Игра	Алгоритм	Уровень сжатия	Необходимое место (N)	Используемое место(U)	U/N	Экономия
96	Spiritfarer: Farewell	zstd	3	6,0 GB	2,3 GB	38%	3,60 GB
			15			39%	3,58 GB
97	Stoneshard: Prologue	zstd	3	289 MB	261 MB	90%	27,2 MB
			15		260 MB	89%	28,4 MB
98	Stories: The Path of Destinies	zstd	3	1,6 GB	1,6 GB	99%	13,8 MB
			15				14,8 MB
99	Styx: Master of Shadow	zstd	3	6,7 GB	6,6 GB	98%	108 MB
			15				114 MB
100	Styx: Shards of Darkness	zstd	3	10 GB	10 GB	99%	17,1 MB
			15				22,9 MB
101	Sundered: Eldritch Edition	zstd	3	2,2 GB	1,7 GB	75%	584 MB
			15		1,5 GB	69%	719 MB
102	SYNTHETIK	zstd	3	599 MB	518 MB	86%	81 MB
			15		516 MB		83 MB
103	Tabletop Simulator	zstd	3	2,7 GB	1,7GB	65%	0,91 GB
			15			63%	0,95 GB
104	The Escapists 2	zstd	3	2,4 GB	1,7 GB	71%	710 MB
			15				717 MB
105	The Life and Suffering of Sir Brante	zstd	3	2,7 GB	1,1 GB	42%	1,52 GB
			15			43%	1,48 GB
106	The Cave	zstd	3	1,1 GB	1,1 GB	98%	22 MB
			15				24 MB
107	The Red Solstice	zstd	3	2,7 GB	1,4 GB	52%	1,31 GB
			15			51%	1,34 GB
108	They Always Run	zstd	3	10 GB	4,2 GB	39%	6,6 GB
			15		3,8 GB	34%	7,1 GB
109	This War of Mine	zstd	3	2,6 GB	2,5 GB	98%	34 MB
			15				36 MB
110	Titan Souls	zstd	3	206 MB	183 MB	88%	21,9 MB
			15		182 MB		22,5 MB
111	Transistor	zstd	3	3,0 GB	2,7 GB	88%	364 MB
			15			87%	384 MB
112	Trine	zstd	3	1,3 GB	1,3 GB	97%	41 MB
			15			96%	44 MB
113	Undertale	zstd	3	155 MB	141 MB	90%	14,2 MB
			15		140 MB		14,9 MB
114	Valiant Hearts: The Great War	zstd	3	1,2 GB	1,1 GB	99%	9,8 MB
			15				10,2 MB
115	Vanquish	zstd	3	18 GB	18 GB	99%	7,7 MB
			15				12,3 MB
116	Vesper	zstd	3	2,8 GB	998 NB	34%	1,88 GB

продолжается на следующей странице

Таблица 3 – продолжение с предыдущей страницы

№	Игра	Алгоритм	Уровень сжатия	Необходимое место (N)	Используемое место(U)	U/N	Экономия
			15		964 MB	32%	1,92 GB
117	Void Bastards	zstd	3	5,7 GB	2,3 GB	40%	3,30 GB
			15			41%	3,28 GB
118	Wasteland 2: Director's Cut	zstd	3	14 GB	13 GB	91%	1,24 GB
			15				1.10 GB
119	Wasteland 3	zstd	3	26 GB	24 GB	91%	2,11 GB
			15		23 GB	89%	2,71 GB
120	Witch It	zsta	3	4,2 GB	4,1 GB	98%	85 MB
			15			97%	95 MB
121	Wizard of Legend	zstd	3	786 MB	475 MB	60%	312 MB
			15		468 MB	59%	318 MB
	Итого	zstd	3	761 GB	666 GB	87%	94 GB
			15		664 GB		97 GB
	Кэш шейдеров представленных здесь игр в Steam	zstd	3	26 GB	25 GB	99%	217 MB
			15				218 MB

Примечания:

- По возможности данный список будет расширяться новыми играми и другими алгоритмами сжатия.
- U/N - выраженное в процентах соотношение количества фактически занятого места к необходимому, т.е. если от 100% отнять U/N можно получить процент сэкономленного места на диске. Из чего следует, что чем меньше данный показатель, тем лучше.
- Экономия рассчитывалась вручную с округлением в меньшую сторону. Другими словами, если получалось 1,3087... GB, то записывалось как 1,30 GB.

Промежуточные результаты

- **64** игр из представленных **121** - практически не сжимаются, экономия места достигает всего 0-10%.
- **33** игр из представленных **121** - сжимаются с низкой эффективностью, экономия места составляет 11-40%.
- **22** игры из представленных **121** - сжимаются со средней эффективностью, экономия места составляет 41-70%.
- **2** игры из представленных **121** - сжимаются хорошо, экономия места составляет 71-90%.
- Кэш шейдеров, который собирается и хранится на диске в Steam (при включении данной функции) сжимается незначительно - менее 1% экономии.
- С учетом разницы в экономии места порядка **3 GB** между максимальной доступной степенью сжатия `15` и рекомендуемой для Btrfs - `3`, и значительного падения скорости выполнения сжатия, а следовательно и скорости записи на диск, можно отметить, что использование для игр степени сжатия выше `3` выглядит крайне сомнительно.

  Во многом полученные результаты можно объяснить тем что в играх с ростом их объема все чаще могут использоваться алгоритмы сжатия, из-за чего сжатие на уровне файловой системы уже менее эффективно.

1.12.3.2 Файловая система bcachefs

Относительно новая файловая система с использованием CoW, которую не так давно добавили в основную ветку ядра Linux в качестве экспериментальной. Активно разрабатывается, но последнее время возникает ряд разногласий между разработчиком файловой системой и разработчиками ядра.

1.13 Wine / Linux Gaming

1.13.1 Основные составляющие

Переходя к запуску Windows-игр на Linux-системах, стóит иметь в виду, что никаких эмуляторов Windows на Linux не существует, и весь запуск осуществляется с помощью открытой реализации Windows API — Wine, а также средств ретрансляции (перевода) команд DirectX в доступные графические API на Linux (Vulkan, OpenGL) с помощью DXVK или иных ретранслятора кода.

1.13.1.1 Что такое Wine?

Wine - слой совместимости для запуска Windows-приложений (в том числе игр) из под Linux (Unix-подобных систем). Благодаря нему вы по факту сможете поиграть в большинство игр из вашей библиотеки Steam/GOG/Epic Games Store. Исключением являются игры с встроенными анти-чит системами, хотя благодаря усилиям Valve, на текущий момент под Linux есть официальная поддержка как минимум двух анти-чит систем: Easy Anti Cheat (EAC) и Battle Eye. Несмотря на это, чтобы игра с использованием данных античитов заработала нужно чтобы разработчики включили специальный слой совместимости между Proton и игрой, что не всегда приветствуется по тем или иным причинам, и даже те игры, которые потенциально могут отлично работать, все ещё не имеют официальной поддержки Proton со стороны разработчиков. Также следует иметь ввиду, что для запуска и обеспечения работоспособности некоторых программ или игр приходится немного повозиться с настройкой, однако сама возможность запуска уже является незаменимой для Linux пользователей, в частности геймеров.

1.13.1.2 Сборки Wine

Существуют различные сборки Wine. Подобный зоопарк появился ввиду накопления большого количества различных патчей (сторонних изменений) которые по какой-то причине не могут быть приняты в обычный Wine. Кроме того, стоит понимать что, как и в случае с ядрами, обычный Wine это прежде всего свободная реализация Windows API, которая подразумевает запуск любых Windows приложений. При этом он не заточен конкретно под игры или любой другой софт. Именно поэтому в том числе и появились такие вещи как Proton от компании Valve, *являющимся по сути тем же Wine*, но с упором именно на игровую составляющую, исправляющий многие проблемы обычного Wine связанные с играми.

На текущий момент есть две официальные "сборки" Wine, которые поддерживаются непосредственно разработчиками:

- wine - обычная, стабильная версия, содержащая только проверенные изменения от разработчиков, и которая условно универсальна для запуска любых приложений.
- wine-staging - версия, содержащая те изменения, которые пока не могут попасть в обычную по тем или иным причинам, но которые могут помочь исправить определенные баги и улучшить работу конкретных программ и частей Wine.

Существуют также много альтернативных сборок основанных на wine-staging с упором именно на игры, о которых написано далее.

1.13.1.3 Установка wine-pure

Проблема обычных версий Wine в том, что они заточены в первую очередь на предоставление "правильной" и "эталонной" реализации Windows API в Unix. Задачей проекта Wine никогда не было реализовать полноценный запуск всех Windows игр под Linux в ущерб технической каноничности реализуемого API, поэтому когда в

Wine отправляются изменения, которые не полностью удовлетворяют подобным требованиям, но при этом помогающие решить проблему запуска или улучшить работоспособность той или иной игры, то они как правило сразу не принимаются, а либо оседают в `wine-staging`, либо в сторонних сборках как версия Wine используемая Proton.

wine-pure - сборка Wine, сделанная с целью упростить процесс сборки и установки Wine, в отличии от Wine-TKG не требует предварительной настройки, содержит минимальный набор патчей необходимый для комфортной игры. Ключевые особенности:

- По умолчанию использует WoW64, транслятор, который осуществляет перевод 32-битных библиотечных вызовов в 64-разрядные, тем самым не требуя установки большого количества 32-битных зависимостей, сохраняя при этом возможность полноценного запуска 32-битных Windows приложений.
- Добавлен патч для поддержки модуля NTSync, позволяющего получить более точное поведение синхронизации потоков как в Windows в отличии от Esync и Fsync, а также повысить производительность[1].
- Содержит все патчи, применяемые в wine-staigng, кроме Esync из-за наличия NTSync.
- По умолчанию активирована нативная поддержка Wayland. Это позволяет получить хороший уровень производительности, а также заметно более низкую задержку и отклик, чем при использовании Xwayland.
- Установлены флаги `-O3` и `-march=native` для нативной компиляции под ваш процессор.
- Удалены лишние и редко используемые компоненты, такие как поддержка Cups, SANE, V4L, pcap, xxf86vm, xinerama.
- Включена поддержка альтернативного бэкенда для проигрывания видео на базе набора библиотек FFmpeg.

Установка

Сборка и установка пакета выполняется как и всегда тремя командами:

```
git clone https://github.com/ventureoo/PKGBUILDs
cd PKGBUILDs/wine-pure-git
makepkg -sricCf
```

Обратите внимание, что для использования NTSync вам нужно установить ядро версии 6.14 и выше или пакет `ntsync-dkms` из AUR, который содержит модуль ядра `ntsync`, иначе будет использоваться встроенная синхронизация wineserver, которая отличается низкой производительностью в многопоточных приложениях Windows запускаемых через Wine.

Проверить используется NTSync или нет можно при помощи утилиты `lsof` во время работы Wine:

```
sudo pacman -S lsof
lsof /dev/ntsync
```

Если вывод команды не пустой, то всё в порядке.

1.13.1.3.1 Proton-GE-Custom

Proton-GE-Custom это форк проекта Proton для запуска Windows-игр с дополнительными патчами и оптимизациями не вошедшими в основную ветку Proton, а также улучшение совместимости с некоторыми играми (например, Warframe). Позволяет играть во многие проекты которые не заводятся с обычным Wine или Proton.

I. Установка (бинарная версия)::

```
git clone https://aur.archlinux.org/proton-ge-custom-bin
cd proton-ge-custom-bin
makepkg -sric
```

[1] https://lore.kernel.org/lkml/20240131021356.10322-3-zfigura@codeweavers.com/T/

Дабы использовать Proton-GE в качестве альтернативы обычному Proton, после установки Proton-GE-Custom вам нужно перезапустить Steam и зайти в *Свойства* нужной вам игры, прожать в: *Совместность -> Принудительно использовать определенный инструмент совместности Steam Play -> Proton-9.XX-GE-1*. Готово, теперь можно запустить игру.

1.13.1.4 Использование Wine

Использование Wine на деле является довольно простым. Чтобы запустить любое Windows-приложение достаточно использовать простую команду:

```
wine программа.exe
```

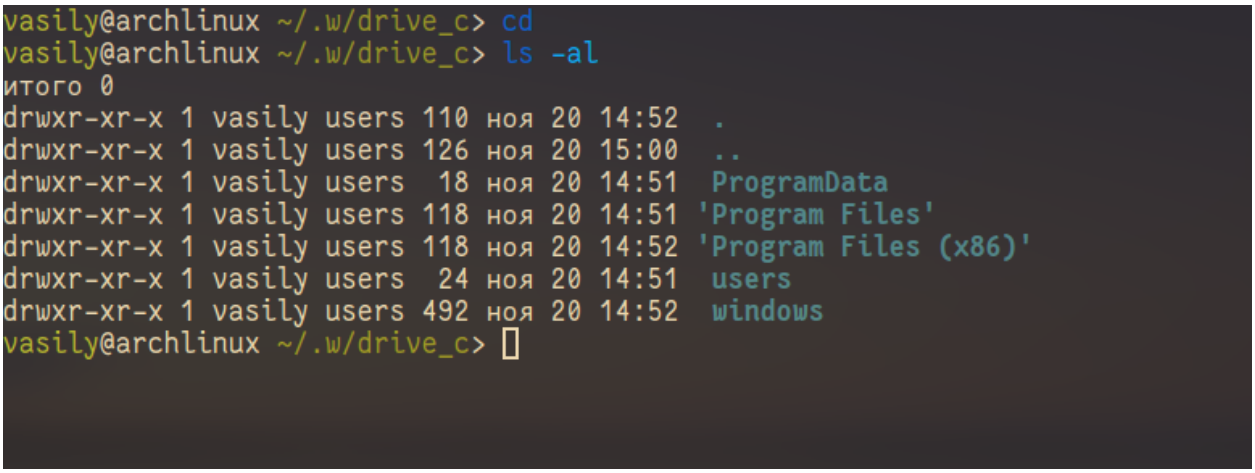
Опасно

Никогда не запускаете wine из под sudo/root! Это поможет вам избежать проблем в будущем, в том числе с безопасностью.

При использовании Wine основополагающим понятием является префикс (в народе его также называют бутылкой). Префикс, это искусственно воссозданное окружение, которое на деле представляет собой обычную директорию, внутри напоминающую собой файловую систему Windows в миниатюре. Внутри префикса будут устанавливаться и работать практически все Windows программы, которые вы будете запускать при помощи Wine. Стоит понимать, что программы запускаемые через Wine по прежнему будут думать, что они работают в Windows, хотя на самом деле это не так. Именно поэтому Wine и понадобилось воссоздать файловую структуру каталогов Windows внутри Linux (Unix). Так называемый префикс по умолчанию - это скрытая директория *~/.wine* в папке вашего пользователя. Если вы её откроете то увидите следующее:

```
vasily@archlinux ~> ls ~/.wine
dosdevices  drive_c  system.reg  userdef.reg  user.reg
vasily@archlinux ~> ls -al ~/.wine
итого 3892
drwxr-xr-x 1 vasily users     126 ноя 20 14:53 .
drwx------ 1 vasily users    1188 ноя 20 14:51 ..
drwxr-xr-x 1 vasily users       8 ноя 20 14:51 dosdevices
drwxr-xr-x 1 vasily users     110 ноя 20 14:52 drive_c
-rw-r--r-- 1 vasily users 3646098 ноя 20 14:53 system.reg
-rw-r--r-- 1 vasily users      12 ноя 20 14:51 .update-timestamp
-rw-r--r-- 1 vasily users    4070 ноя 20 14:52 userdef.reg
-rw-r--r-- 1 vasily users  326449 ноя 20 14:53 user.reg
vasily@archlinux ~>
```

Как мы видим, в префиксе находятся файлы с расширением .reg (файлы реестра Windows), директории *dosdevices* и *drive_c*. Файлы реестра используются Wine для, собственно, воссоздания работы реестра Windows в Linux. К ним также будут обращаться программы запускаемые через Wine. Директория *dosdevices* содержит символические ссылки на примонтированные устройства (разделы) в вашей системе Linux. Это понадобилось для того чтобы представить их в виде MS-DOS томов, ибо Windows приложения опять таки не знают, что они работают под Linux, и им нужны привычные им диски D, E и т.д. Один из таких "виртуальных дисков", а именно главный диск C, располагается в другом каталоге - *drive_c*. Если вы его откроете, то увидите как раз таки "замечательную" и привычную структуру каталогов Windows:



Именно сюда как правило и будут устанавливаться ваши Windows программы, а также сохранятся все их временные и постоянные данные.

Вы можете переназначить используемый префикс через переменную окружения *WINEPREFIX*, указав Wine использовать другую директорию для его расположения вместо *~/.wine*. Например:

```
WINEPREFIX=~/Games wine game.exe # Если директории не было, Wine её создаст.
```

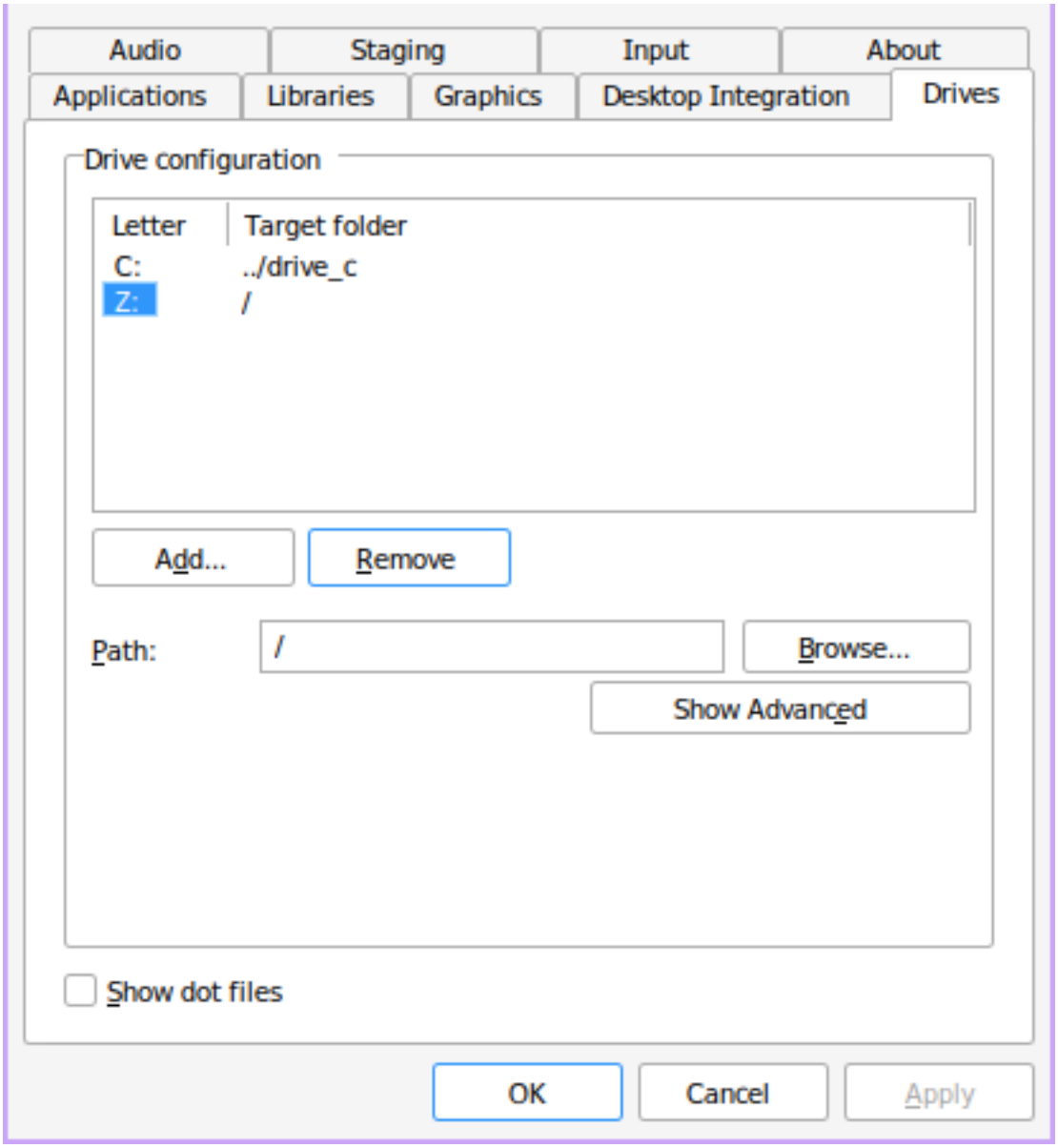
Учитывайте, что при смене префикса через переменную окружения `WINEPREFIX` не переносится его содержимое, т.е. программы установленные в одном префиксе не будут скопированы в новый. Поэтому вам нужно будет вручную их перенести в новый префикс. Это не распространяется на те программы, которые расположены вне используемого префикса, например, если вы запускаете exe-файл из папки *Загрузки*, то он все так же будут запускаться, однако следует учитывать, что установку игр и других программ все же лучше выполнять внутрь префикса, так как некоторые программы (например такие как различные пиратские установщики игр), могут не работать правильно при запуске вне префикса. Кроме того, доступ Wine к файлам, находящимся вне рамок префикса, может быть нежелательным с точки зрения безопасности, так как не следует забывать, что Wine все ещё не является 100% защитой от любых вредоносных Windows программ, которые вы через него запускаете. Чтобы полностью ограничить доступ Wine ко всем файлам, находящимся вне префикса читайте далее.

1.13.1.4.1 Настройка префикса

Как уже было отмечено в самом начале, не все игры и программы работают идеально под Wine, но работу некоторых программ можно улучшить за счёт грамотной настройки префикса, о чем и будет идти речь в данном разделе.

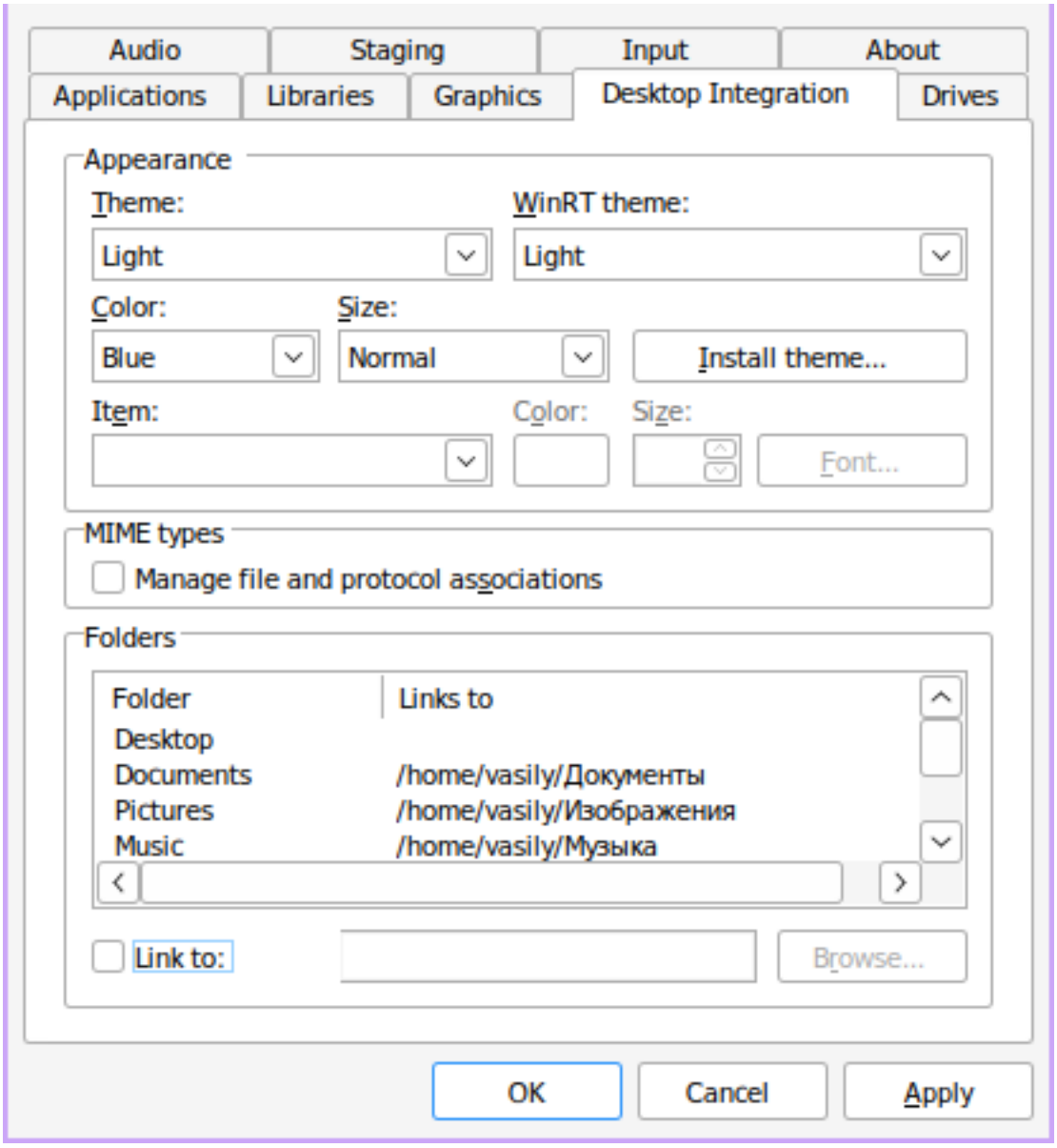
Изоляция Wine программ от доступа к файловой системе

По умолчанию Wine создаёт внутри префикса символические ссылки, по существу представляющие собой "мостик" для доступа приложений ко всей остальной файловой системе. Это необходимо как раз таки для того, чтобы вы могли запускать exe-файлы, которые находятся вне рамок вашего префикса, но это имеет за собой угрозу компрометации ваших личных данных, так как запускаемая программа может оказаться троянским конем, который как раз таки может исследовать префикс и пройти внутрь по всем "виртуальным дискам", которые ведут напрямую к вашей файловой системе Linux. Хотя Wine работает в рамках полномочий вашей учетной записи и не может ни коим образом повлиять на системные файлы, возможность считывать данные внутри вашей домашней директории по прежнему остается. Вот почему важно полностью "изолировать" работу Windows приложений только внутри префикса Wine. Чтобы это сделать достаточно открыть утилиту `winecfg`, перейти во вкладку *"Диски"* (*Drives*), выбрать диск *Z:*, который представляет собой символическую ссылку на ваш корневой раздел, и нажать кнопку *Удалить* (*Remove*):



Кроме диска *Z:* могут присутствовать символические ссылки также и на другие смонтированные носители, их тоже рекомендуется аналогично удалить, так чтобы остался доступ только к диску *C:*, который представляет собой простую директорию внутри префикса.

Уже это значительно увеличит вашу безопасность, однако это ещё всё. Wine так же создает символические ссылки и внутри диска воображаемого диска C:, а точнее в `drive_c/users/имя_вашего_пользователя`, которые связывают имена классических пользовательских папок в Windows, например такими как *"Мои документы"*, *"Загрузки"* с соответствующими аналогами в Linux, такими как *~/Документы* и *~/Загрузки* (если у вас установлена русская локаль), что так же создает определенную угрозу компрометации личных данных. Чтобы предотвратить произвольный доступ к ним нужно не выходя из утилиты `winecfg` перейти во вкладку *"Вид и интеграции"* (*Desktop Integration*), в категории *Папки* (*Folders*) выбрать соответствующую папку и снять галочку с *"Привязать к"* (*Link to*). То же самое нужно проделать так со всеми доступными папками:



После этого не забываем перед выходом прожать кнопку *"Применить"* (*Apply*).

Проблема потери управления при смени фокуса на другое окно

Пользователи Wine могли встретить один очень раздражающий баг, когда при переключении с окна с игрой на другое окно, и последующим возвращением фокуса обратно на окно с игрой, Wine перестает реагировать на нажатия клавиш. Чаще всего эта проблема встречается в играх, созданных на базе движка Unity, но может быть и в других.

Чтобы предотвратить возникновение этой проблемы достаточно изменить значение одного ключа реестра Wine, что можно сделать несколькими способами:

Через терминал

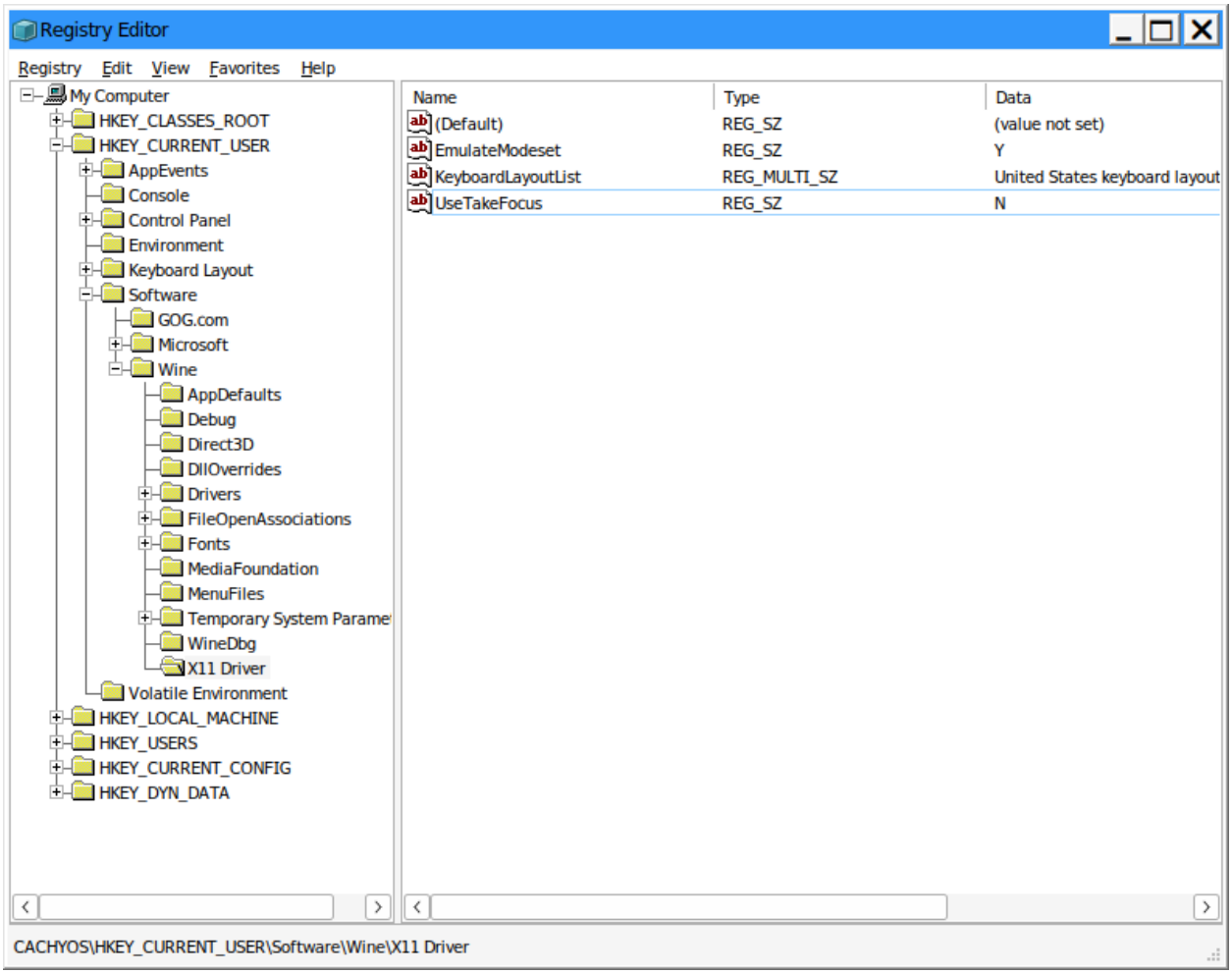
```
wine reg ADD 'HKEY_CURRENT_USER\Software\Wine\X11 Driver' /v UseTakeFocus /d 'N' /f
```

Интерактивный способ

Значение ключа реестра можно изменить и интерактивным способом через встроенную программу "Реестра" в Wine:

```
wine regedit
```

Чтобы создать ключ как показано на скриншоте нужно выбрать в контекстном меню *Правка* (*Edit*) -> *Создать* (*New*) -> *Строковый параметр* (*String value*)-> Ввести имя параметра `UseTakeFocus` -> Установить значение `N`.



Естественно, в случае если вы хотите решить эту проблему не только для игр, находящихся в префиксе по умолчанию (`~/.wine`), то вы должны сопроводить нужную команду (в зависимости от выбранного способа), соответственно установленным значением переменной `WINEPREFIX`, так как любые изменения в реестре Wine локальны только для текущего выбранного префикса.

Исправление чёрных окон лаунчеров на базе .NET

К сожалению, Wine пока ещё очень плохо справляется с правильным отображением различного рода "лаунчеров", в частности тех, которые используют весьма Windows-специфические графические тулкиты, вроде того же 4-го .NET, который часто можно встретить в различных "запускалках" пиратских версий игр. Часто встречаемая проблема - это либо полностью чёрное, либо мерцающее окно лаунчера. Как правило чтобы это исправить достаточно отключить аппаратное ускорение видео для таких окон, отредактировав очередной ключ реестра:

Через терминал

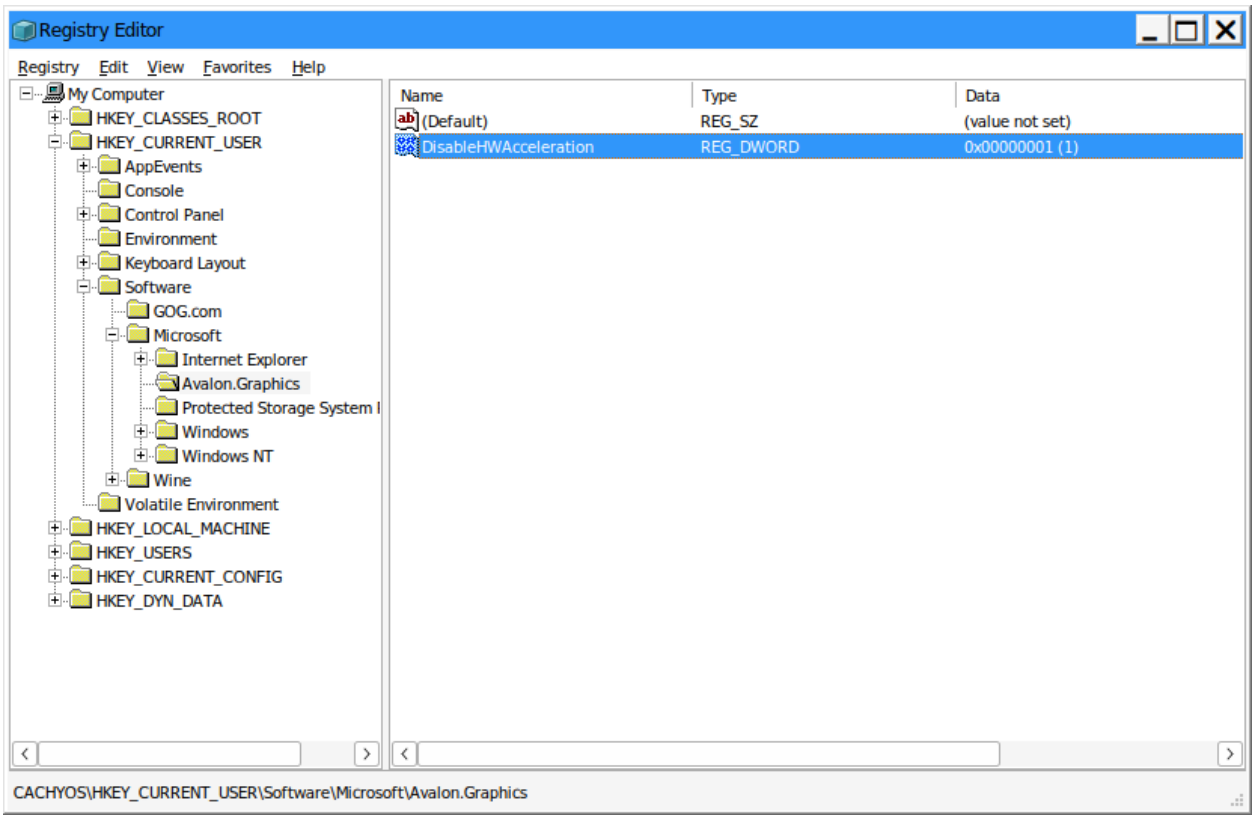
```
wine reg add "HKEY_CURRENT_USER\\SOFTWARE\\Microsoft\\Avalon.Graphics" /v␣
↪DisableHWAcceleration /t REG_DWORD /d 1 /f
```

Интерактивный способ

Значение ключа реестра можно изменить и интерактивным способом через встроенную программу "Реестра" в Wine:

```
wine regedit
```

Чтобы создать ключ как показано на скриншоте нужно сначала выбрать раздел `HKEY_CURRENT_USER\Software\Microsoft`, потом контекстном меню выбрать *Правка (Edit)* -> *Создать (New)* -> *Раздел (Key)* -> `Avalon.Graphics` и уже выбрав данный новый раздел, перейти снова в *Правка (Edit)* -> *Создать (New)* -> *Параметр DWORD (DWORD value)* -> `DisableHWAcceleration` -> Установить значение `1`.



Как и всегда не забываем, чтобы изменения реестра работают только в рамках текущего префикса, поэтому если у вас есть несколько проблемных лаунчеров в разных префиксах, то подставляйте соответствующую директорию префикса в переменную `WINEPREFIX`.

Исправление чёрных окон лаунчеров на базе CEF

Аналогичные проблемы возникают с другими лаунчерами, которые базируются не на .NET, а на CEF (Chromium Embedded Framework). Такие лаунчеры очень часто встречаются среди разработанных большими компаниями: Epic Games Store, Origin, Battle.net, Lesta Game Center, Ubisoft Connect, лаунчер Genshin Impact и многие другие. Одним словом, практически все известные сервисы и платформы для запуска игр. Проблема с их правильным отображением состоит в том, что CEF по умолчанию использует так называемый *Cross Process Rendering*, который, как понятно из названия, позволяет рисовать окно приложения сразу с использованием нескольких процессов, что очень плохо работает в Wine, из-за чего возникают различные артефакты или уже известные чёрные окна.

Исправить проблемы с такими лаунчерами можно только указав для них дополнительные аргументы запуска, а именно: `--single-process --disable-gpu --disable-gpu-compositing --in-process-gpu`.

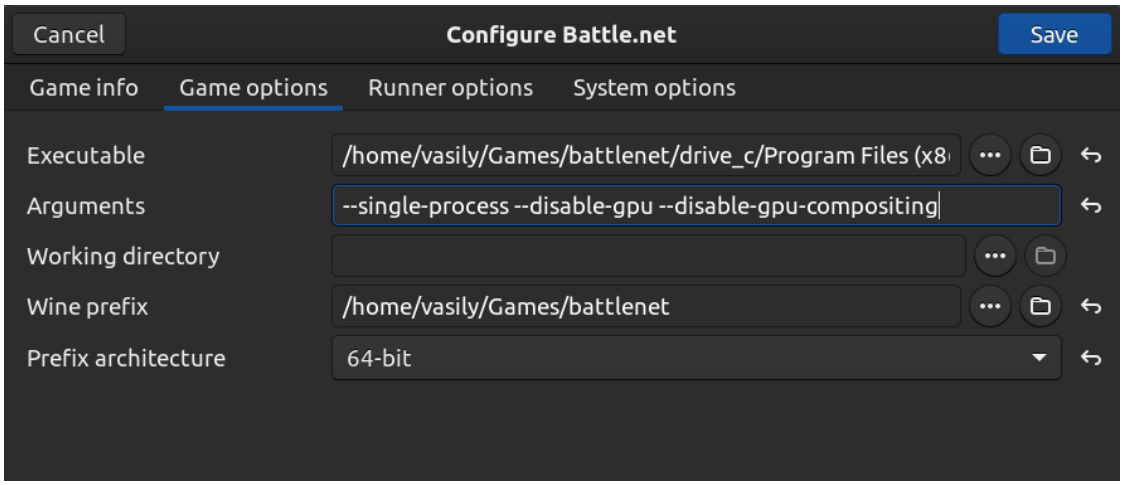
Например, в случае с Battle.net:

Запуск через терминал

```
wine "Battle.net Launcher.exe" --single-process --disable-gpu --disable-gpu-
↪compositing --in-process-gpu
```

Запуск через Lutris

В Lutris аргументы запуска указываются в параметрах игры как показано на скриншоте.



Не забываем сохранять проделанные изменения!

Примечание

Данная проблема особенно актуальна при использовании нативного Wayland драйвера в Wine, так как на текущий момент он вообще не умеет работать с Cross Process Rendering.

1.13.1.5 DXVK

В Linux отсутствует полноценная реализация DirectX по вполне понятным причинам. Но присутствуют альтернативные графические API, работающие под любые платформы. Прежде всего это OpenGL и Vulkan. В следствии этого в Wine есть так называемый ретранслятор кода - wined3d. Он переводит вызовы DirectX в известные любой Linux системе OpenGL вызовы. Однако OpenGL не одно и тоже что и DirectX, поэтому возникают множество проблем. Самая главная из которых - значительно более худшая производительность

OpenGL по сравнению с DirectX. Именно поэтому если вы запустите любую игру через "голый" Wine вы получите ужасный FPS, т.к. она будет работать с использованием wined3d. По этой причине был разработан другой ретранслятор кода - DXVK. Он переводит DirectX вызовы уже не в OpenGL, а в Vulkan - более современный графический API, который достигает паритета по возможностям и производительности с DirectX.

Установка DXVK - это первое что должен сделать любой игрок который собирается запустить Windows-игру под Linux. Хотя при использовании любой версии Proton DXVK уже есть из коробки, для игр запускаемых через обычный Wine его придется устанавливать вручную.

Мы рекомендуем устанавливать dxvk-pure-git для лучшей производительности:

Предупреждение

Последние версии DXVK требуют поддержки драйвером вашего GPU версии по крайне мере API Vulkan 1.3. Если вы являетесь обладателем видеокарты NVIDIA поколения Kepler (Geforce GTX 6xx/7xx), то вы ограничены поддержкой Vulkan 1.2, и вам нет смысла устанавливать DXVK по этой инструкции. Вместо этого используйте последнюю доступную версию DXVK 1.10.3, которую можно выбрать к использованию в настройках Lutris.

Установка::

```
git clone https://github.com/ventureoo/PKGBUILDs
cd dxvk-pure-git
makepkg -sric
```

После установки пакета DXVK не задействуется сразу же. Так как фактически DXVK представляет собой набор DLL библиотек, то мы должны установить их внутрь Wine префикса. Для пользователей графических помощников, таких как Lutris делать это вручную не требуется, но следует указать новую свежеустановленную версию DXVK.

Ручной способ

В классическом варианте установка DXVK внутрь префикса это простой перенос всех DLL библиотек в специальные директории. Однако автор рекомендует создавать символические ссылки на системные пути, чтобы не копировать каждый раз новые DLL библиотеки в префикс (в примере ниже это `~/.wine`) при обновлении пакета:

```
ln -s -f /usr/share/dxvk/x32/*.dll ~/.wine/drive_c/windows/syswow64
ln -s -f /usr/share/dxvk/x64/*.dll ~/.wine/drive_c/windows/system32
```

После чего необходимо специально указать Wine, чтобы вместо встроенной реализации DirectX через wined3d использовались реализация в первую очередь из сторонних DLL библиотек, то есть DXVK:

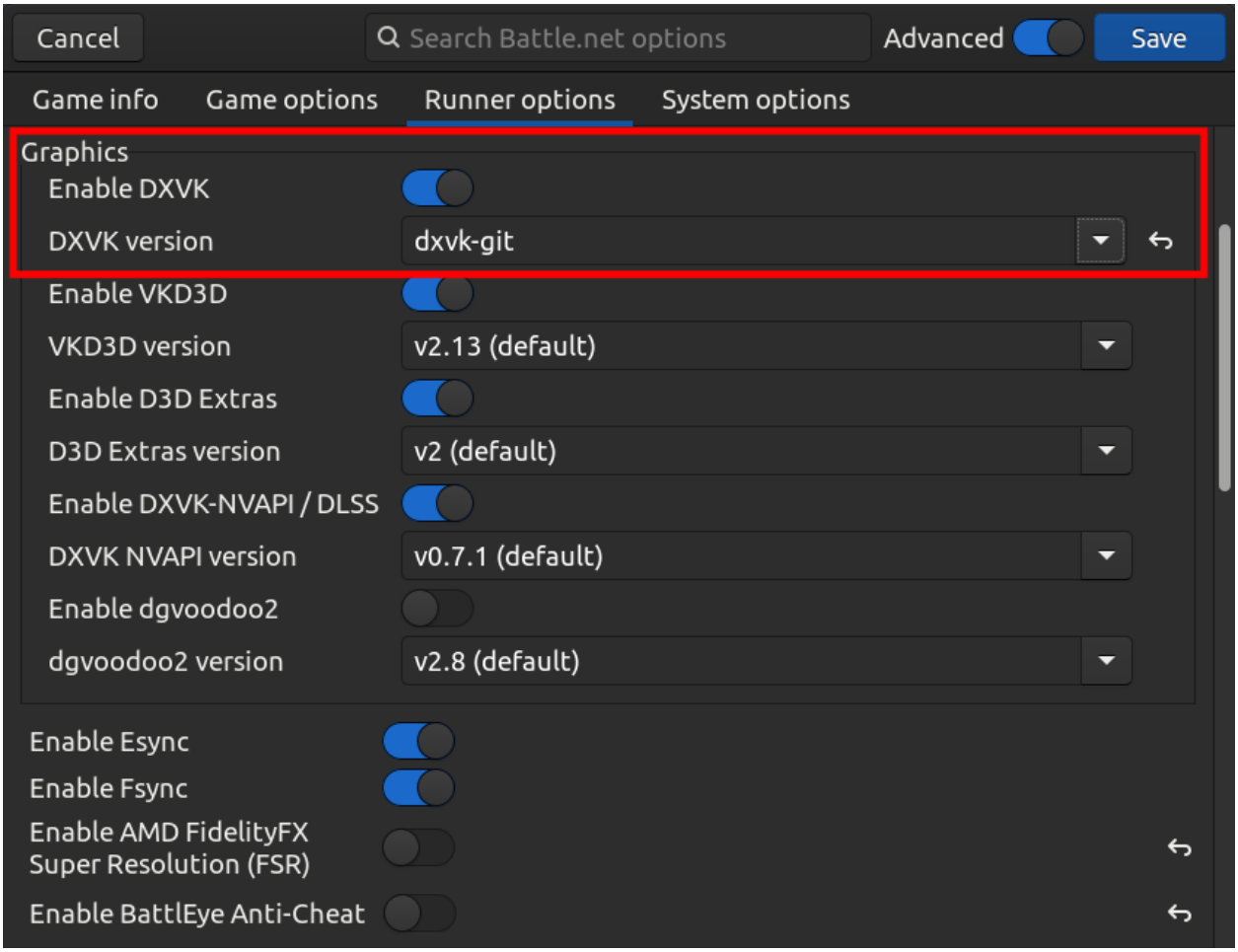
```
wine reg add 'HKEY_CURRENT_USER\Software\Wine\DllOverrides' /v d3d8 /d native,builtin␣
↪/f
wine reg add 'HKEY_CURRENT_USER\Software\Wine\DllOverrides' /v d3d9 /d native,builtin␣
↪/f
wine reg add 'HKEY_CURRENT_USER\Software\Wine\DllOverrides' /v d3d10core /d native,
↪builtin /f
wine reg add 'HKEY_CURRENT_USER\Software\Wine\DllOverrides' /v d3d11 /d native,
↪builtin /f
wine reg add 'HKEY_CURRENT_USER\Software\Wine\DllOverrides' /v dxgi /d native,builtin␣
↪/f
```

При использовании Lutris

Если вы используете графический помощник Lutris, то как уже было сказано выполнять рутинные действия по переносу DLL библиотек внутрь префикса не нужно, однако Lutris хранит все используемые им версии DXVK в специальной директории, куда нужно поместить и только что установленную версию DXVK. Для этого следует создать символическую ссылку, что позволит каждый раз не переносить DLL библиотеки с новыми версиями пакета:

```
ln -s /usr/share/dxvk ~/.local/share/lutris/runtime/dxvk/dxvk-git
```

После перезапуска Lutris в соответствующем поле выбора версии DXVK появится новая версия `dxvk-git`, которая и будет представлять собой только что свеже скомпилированную версию DXVK.



Предупреждение

DXVK осуществляет ретрансляцию вызовов только для игр использующих версии DirectX 8, 9, 10 и 11. Для DirectX 12 понадобиться использовать vkd3d-proton. Подробнее о нем вы можете прочитать ниже.

Примечание

Начиная с версии 2.0 и выше DXVK компилирует шейдеры заранее, так чтобы бы вы не сталкивались с

заиканиями непосредственно вовремя игры. К сожалению, не во всех играх это работает как следует, к таковым относятся некоторые проекты разработанные на движке Unreal Engine. Тем не менее, с помощью определенных опций самого движка все же можно позволить DXVK собирать шейдеры в фоне во время загрузки игры. Для игр на базе Unreal Engine 4/5 нужно отредактировав файл `%LOCALAPPDATA%/game_name/Saved/Config/WindowsNoEditor/Engine.ini` (путь к конфигурационному файлу внутри префикса может отличаться от игры к игре):

```
[/script/engine.rendererсettings]
r.Shaders.Optimize=1
r.CreateShadersOnLoad=1
niagara.CreateShadersOnLoad=1
r.ShaderDevelopmentMode=0
r.CompileShadersForDevelopment=0
```

Аналогично для игр использующих UnrealEngine 3 существует параметр `bInitializeShadersOnDemand=False` (спасибо @Iglu47 для предоставленную информацию).

1.13.1.6 vkd3d

vkd3d - это ретранслятор кода, аналогичный DXVK, но уже конкретно для версии DirectX 12. Стоит отметить, что существует две отдельно разрабатываемые версии vkd3d, одна из которых разрабатывается командой Wine, а другая - Valve. Мы рекомендуем вам использовать ту что от Valve, т.к. она наиболее заточена под современные игры, а также достаточно хорошо поддерживает Raytracing.

Установка vkd3d-proton

Вместо обычных бинарных сборок мы установим vkd3d-proton из AUR, нативно-скомпилировав его под свой процессор:

```
git clone https://aur.archlinux.org/vkd3d-proton-mingw.git # Скачивание исходников
cd vkd3d-proton-mingw                                      # Переход в директорию
makepkg -sric                                              # Сборка и установка
```

Так же как и в случае с DXVK, после установки пакета, если речь про ручную установку, библиотеки vkd3d-proton нужно сначала скопировать в нужный Wine префикс и форсировать их использование, либо указать их в качестве используемых при использовании в Lutris.

Ручной способ

Для использования DLL библиотек vkd3d-proton в рамках выбранного префикса (в примере это `~/.wine`) нужно перенести их внутрь специальных директорий, где находятся все DLL библиотеки известные Wine, соответствующей разрядности. Тем не менее, автор рекомендует создавать символические ссылки в нужные системные пути, чтобы не нужно было каждый раз переносить новые версии DLL библиотек внутрь префикса при обновлении пакета:

```
ln -s -f /usr/share/vkd3d-proton/x86/*.dll ~/.wine/drive_c/windows/syswow64
ln -s -f /usr/share/vkd3d-proton/x64/*.dll ~/.wine/drive_c/windows/system32
```

А также следует указать форсированное использование vkd3d-proton вместо встроенной в Wine реализации vkd3d:

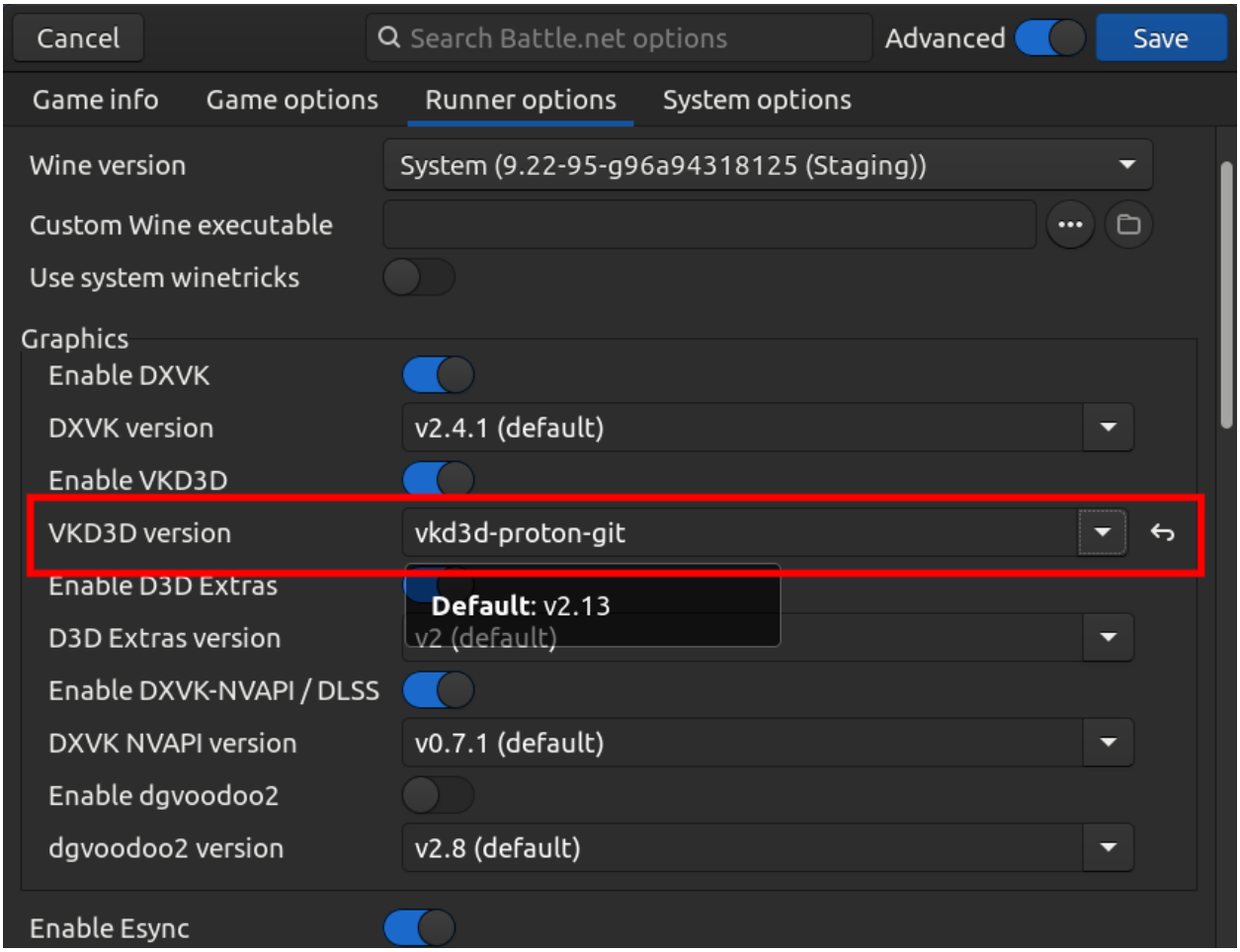
```
wine reg add 'HKEY_CURRENT_USER\Software\Wine\DllOverrides' /v d3d12 /d native,
↪builtin /f
wine reg add 'HKEY_CURRENT_USER\Software\Wine\DllOverrides' /v d3d12core /d native,
↪builtin /f
```

При использовании Lutris

При использовании графического помощника Lutris переносить DLL библиотеки внутрь префикса не нужно, так как он уже делает это за вас, однако нужно поместить DLL библиотеки vkd3d-proton в специальную директорию, чтобы они стали доступными для выбора в качестве версии vkd3d-proton в Lutris:

```
ln -s /usr/share/vkd3d-proton ~/.local/share/lutris/runtime/vkd3d/vkd3d-proton-git
```

После перезапуска Lutris новая версия vkd3d-proton под названием `vkd3d-proton-git` станет доступна в окне выбора версии:



1.13.1.7 Полезные ссылки по теме Wine и DXVK

Скачать готовые сборки Wine и DXVK

https://github.com/Kron4ek/Wine-Builds

https://mirror.cachyos.org/?search=wine

Почитать, что это такое

https://www.newalive.net/234-sborki-dxvk-i-d9vk.html

https://www.newalive.net/231-wine-tk-glitch.html

1.13.2 Дополнительные компоненты

Не являются обязательными, но могут помочь повысить производительность системы или облегчить настройку.

1.13.2.1 Lutris

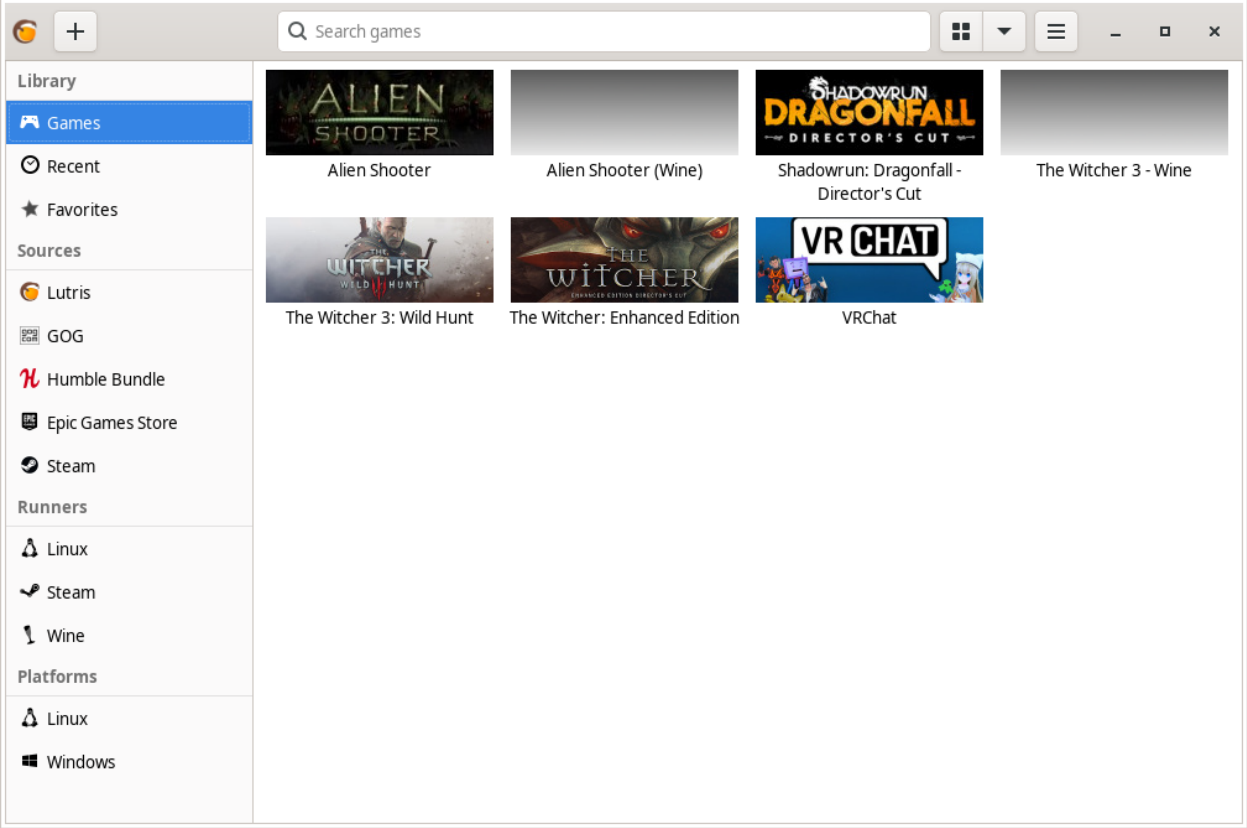
Lutris - это удобный графический интерфейс по обслуживанию всей вашей игровой библиотеки (включая все купленные игры Steam/GOG/Epic Games) в одном приложении. Через него вы сможете достаточно просто запускать нативные игры, игры запускаемые при помощи эмуляторов, и конечно Wine. Все это объединено в одном приложении-комбайне, содержащим много настроек и интеграций с различными сервисами.

Установка

Все проще некуда:

```
sudo pacman -S lutris
```

Тем не менее, стоит удостовериться что вы установили полный набор зависимостей для Wine. Об этом вы можете прочитать в предыдущих разделах.

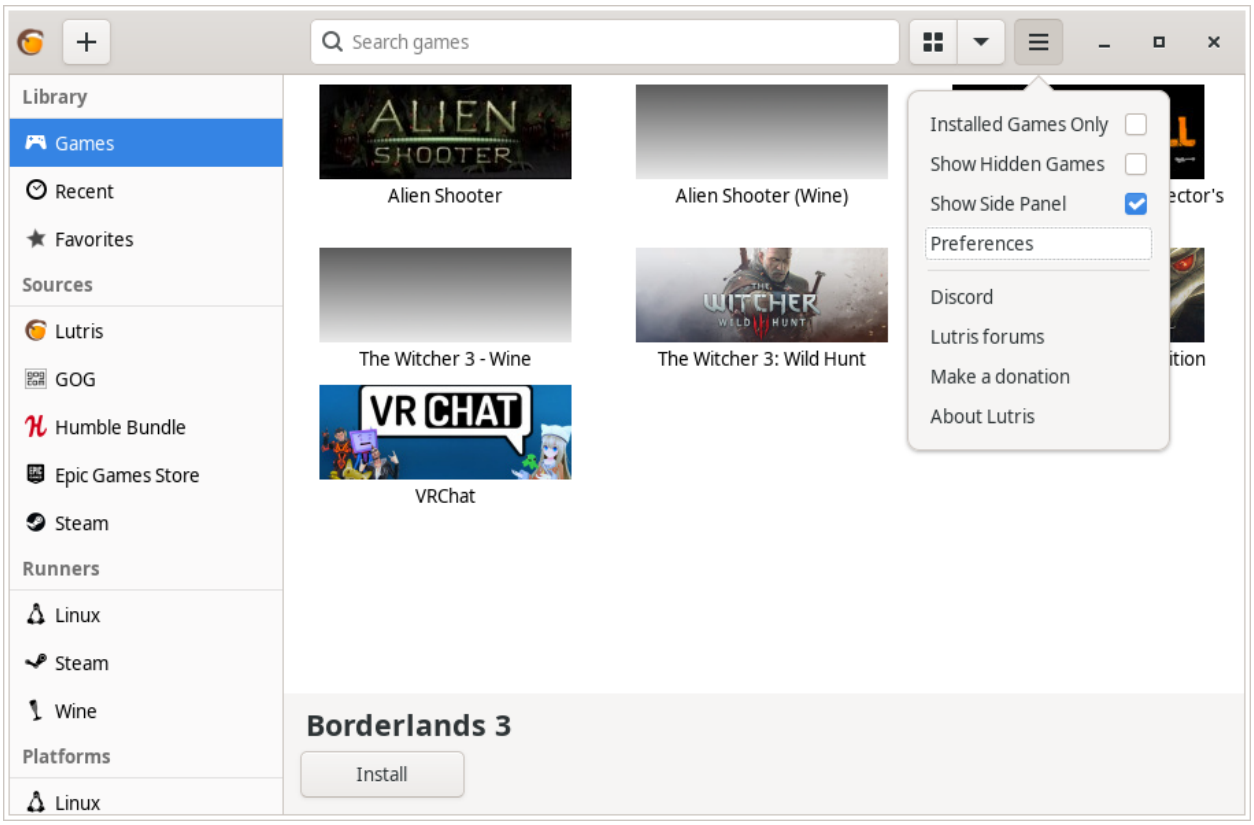


Интеграция с GOG/Epic/Steam

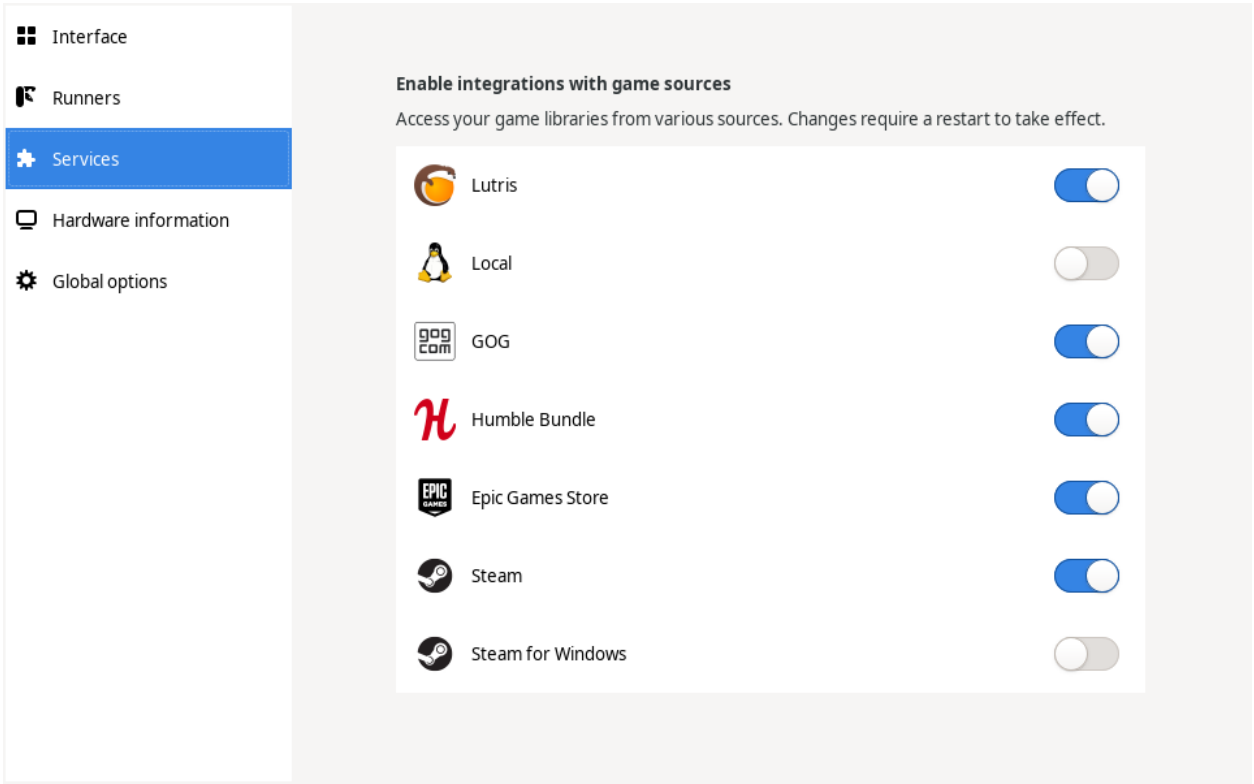
Сразу после установки стоит сделать некоторые базовые вещи. А именно подключить интеграцию с сервисами Steam/GOG/Epic Games. Это позволит синхронизировать локальную библиотеку Lutris'a вместе с перечисленными площадками и выполнять установку игр в два клика. Подключать все конечно не обязательно, так что делайте это если считаете нужным.

1. Зайдем в настройки: В правом верхнем углу найдите три горизонтальные полоски и в контекстном меню выберите *"Preferences"*. После этого выберите *"Services"* и включите те сервисы, которыми вы пользуетесь.

1.1



1.2

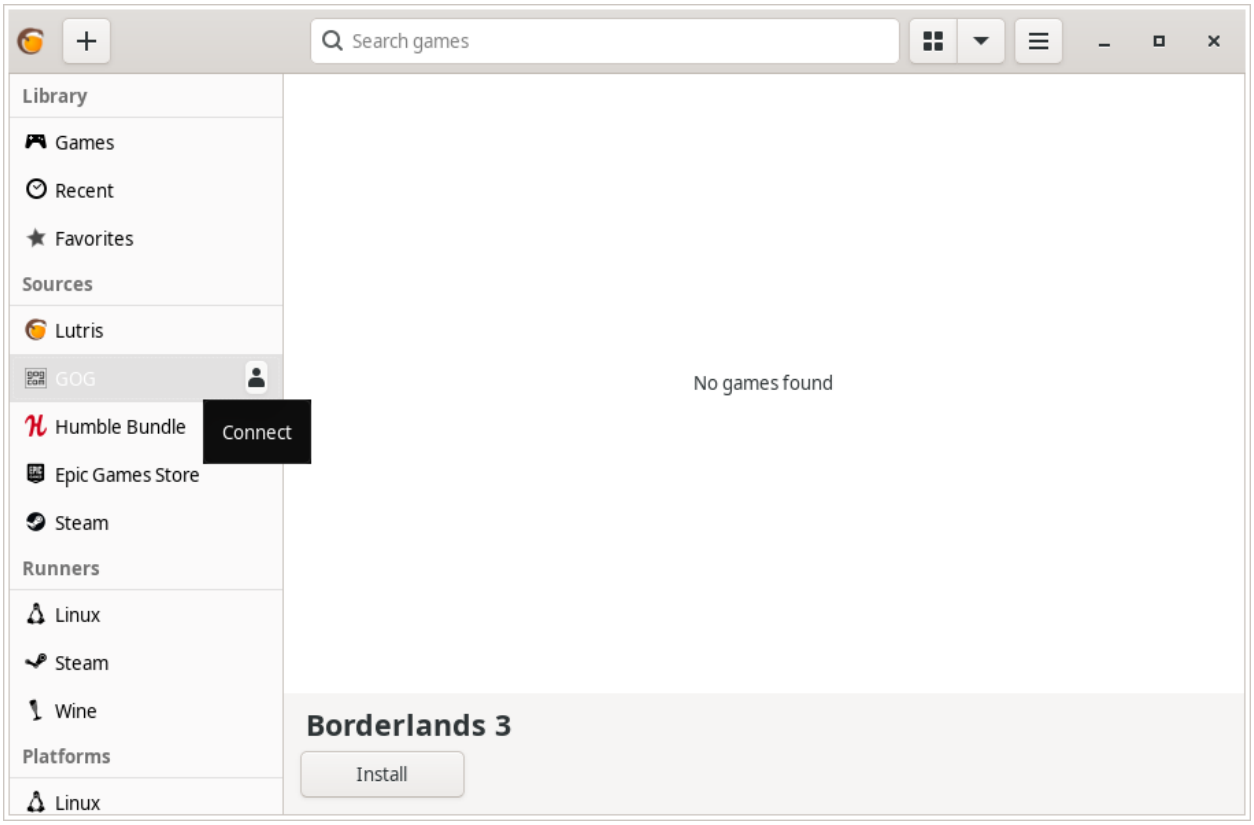


2. Теперь вернитесь в главное окно и наведите курсор на левую панель в графу *"Sources"*, и ниже выберите нужную

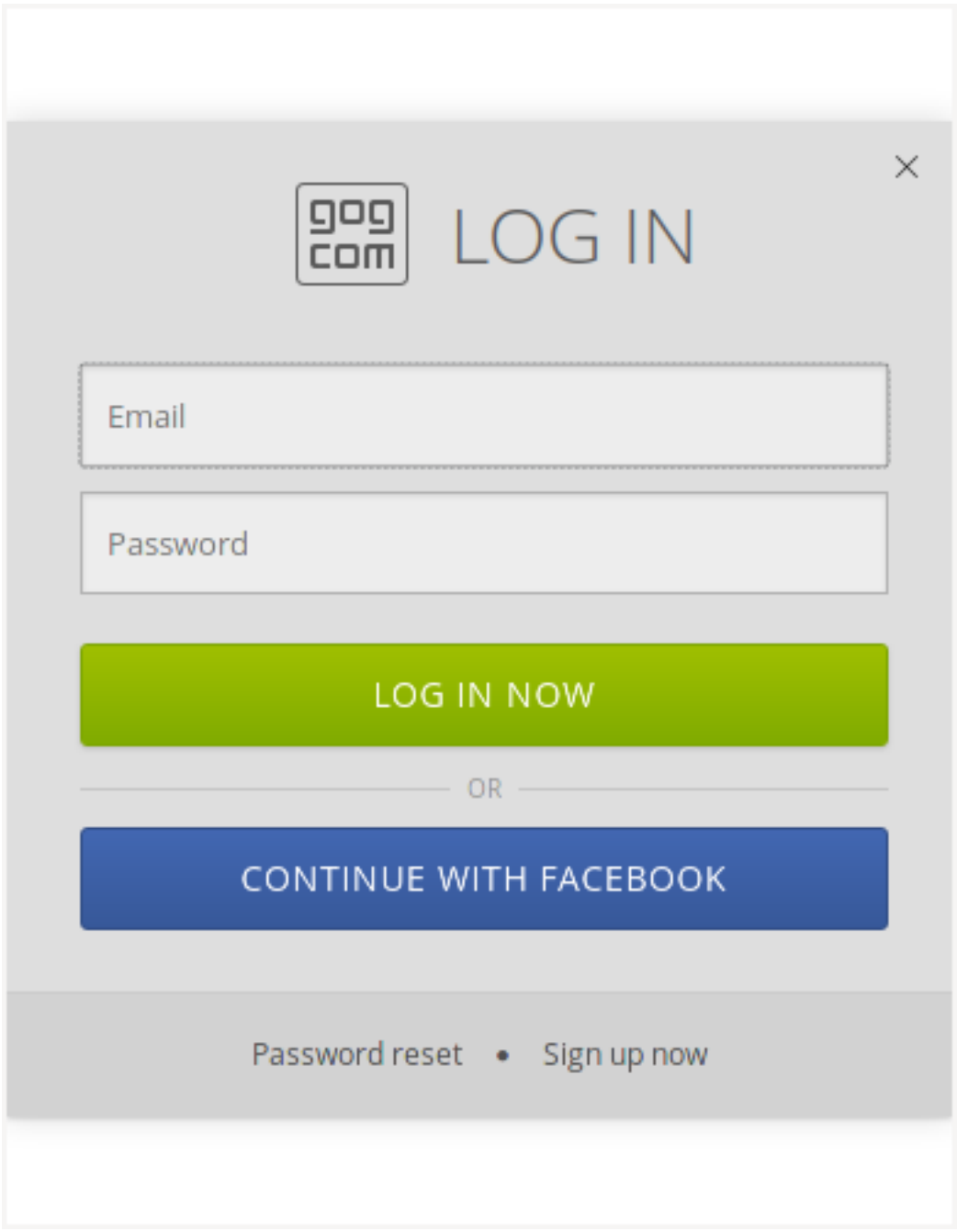
вам платформу. Справа от курсора будет иконка входа. После этого перед вами появится окно авторизации, после прохождения которой у вас появится возможность устанавливать и запускать все игры из вашей внешней библиотеки (Steam/GOG/Epic Games).

Пример подключения аккаунта GOG представлен ниже на скриншотах.

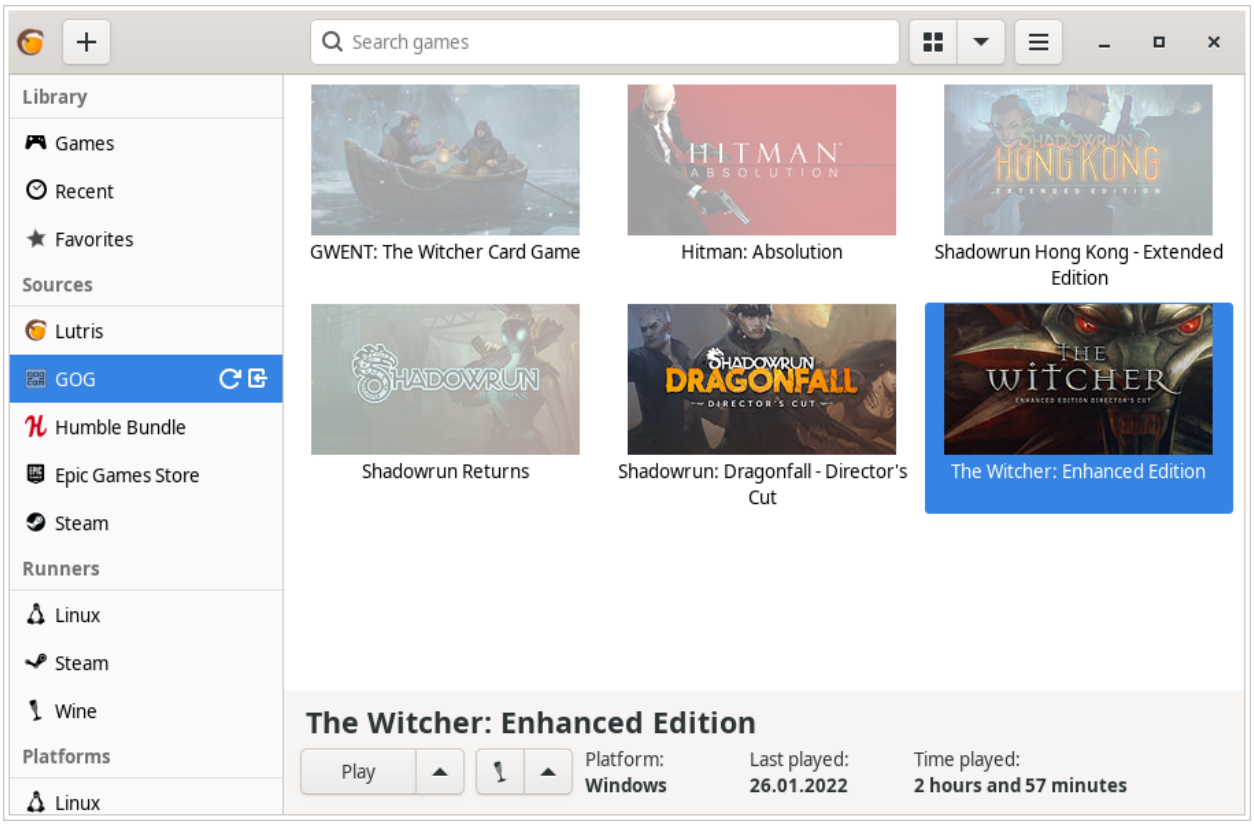
2.1



2.2

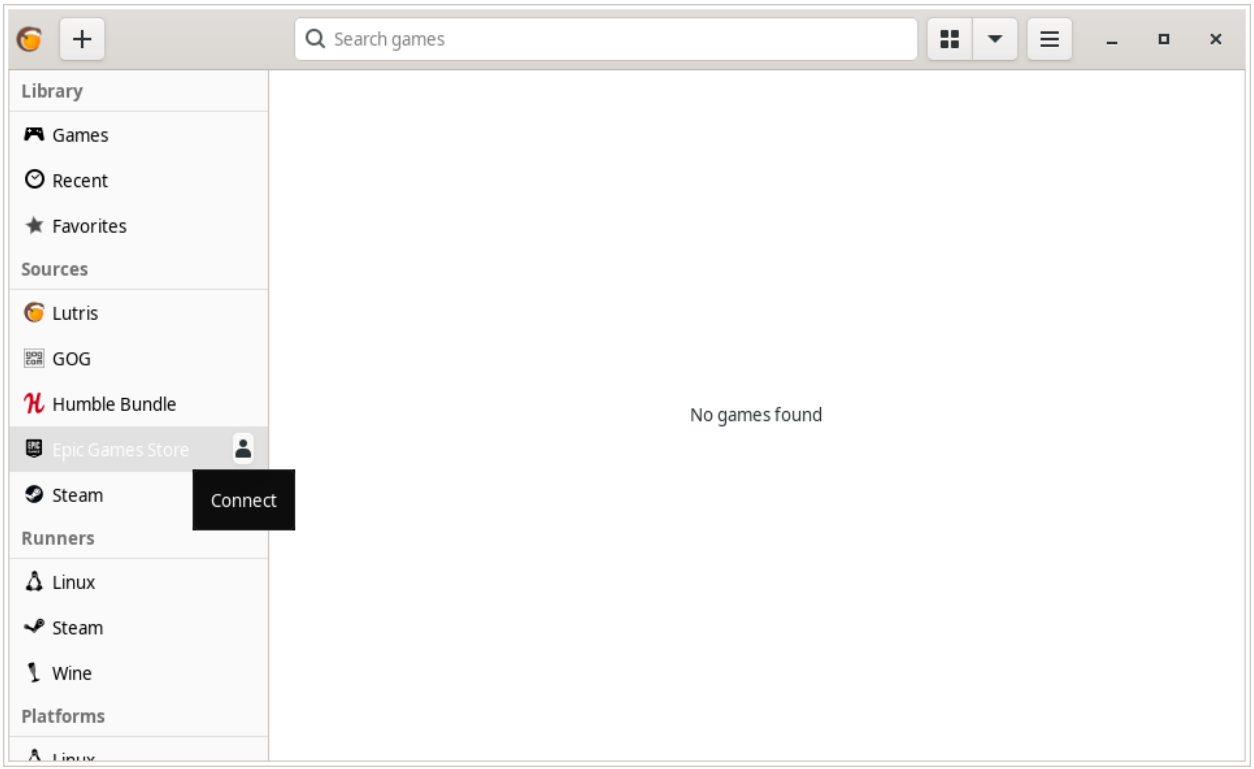

LOG IN
Email
Password
LOG IN NOW
OR
CONTINUE WITH FACEBOOK
Password reset • Sign up now

2.3

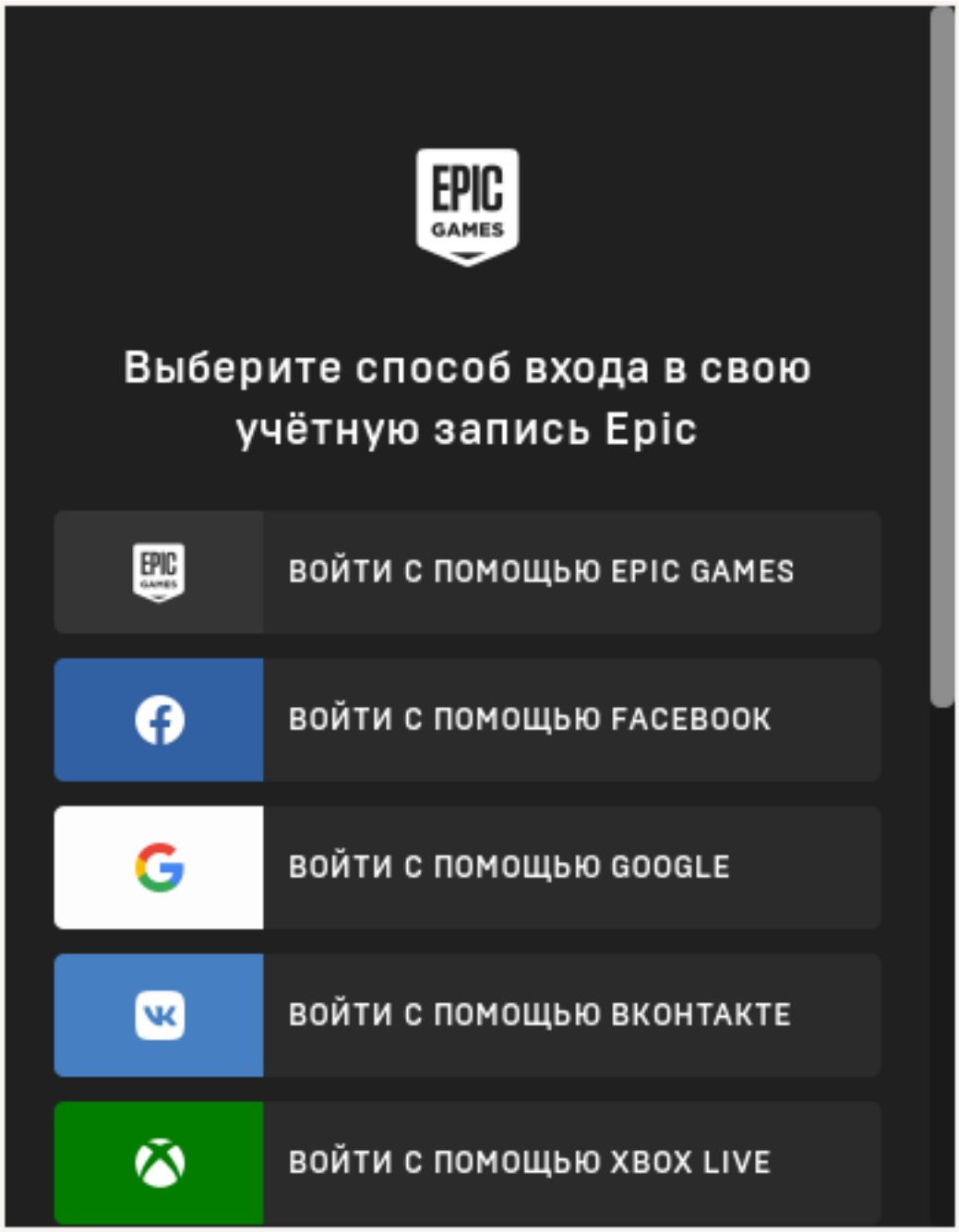


Аналогичная операция проделывается с Epic Games Store:

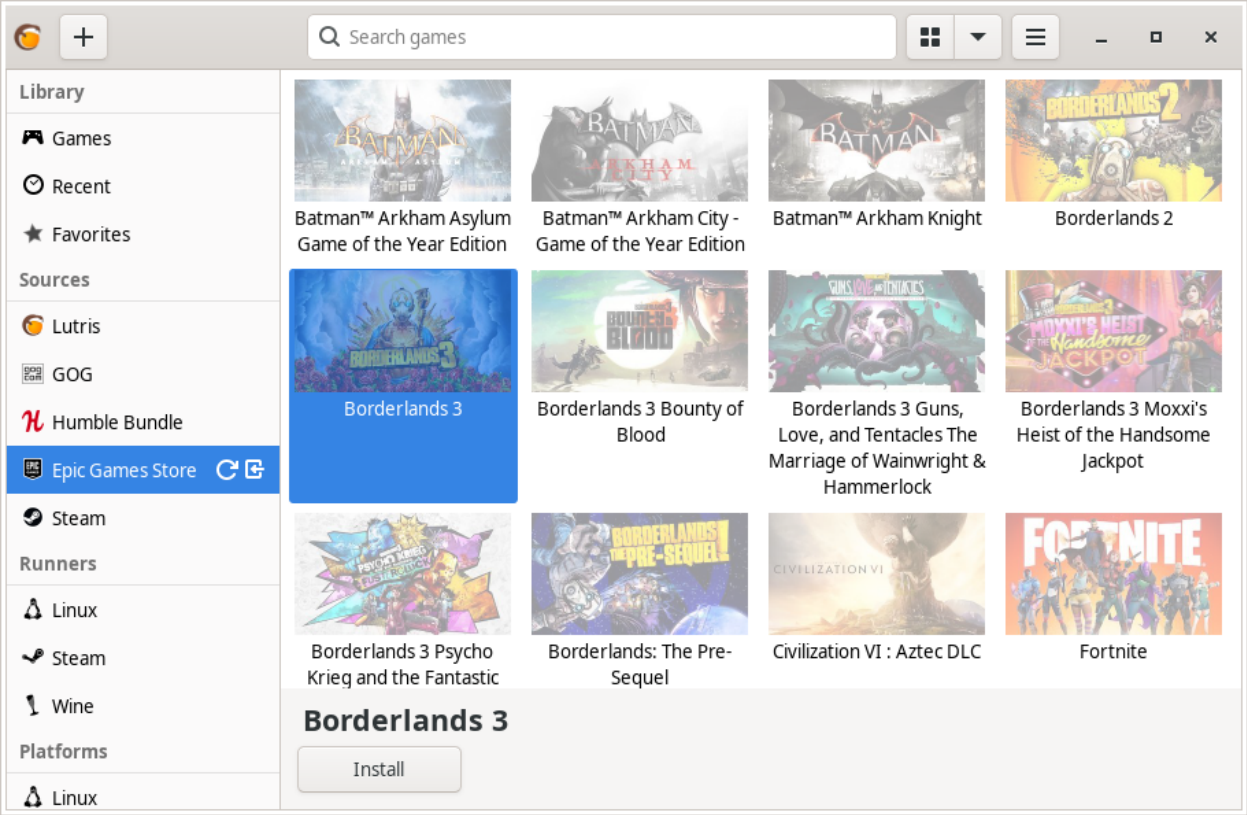
2.4



2.5


EPIC GAMES
Выберите способ входа в свою учётную запись Epic
ВОЙТИ С ПОМОЩЬЮ EPIC GAMES
ВОЙТИ С ПОМОЩЬЮ FACEBOOK
ВОЙТИ С ПОМОЩЬЮ GOOGLE
ВОЙТИ С ПОМОЩЬЮ ВКОНТАКТЕ
ВОЙТИ С ПОМОЩЬЮ XBOX LIVE

2.6



1.13.2.2 Gamemode

Gamemode - утилита для максимальной выжимки системы во время игры. Установку gamemode можно выполнить следующей командой:

```
sudo pacman -S gamemode lib32-gamemode
```

Lutris, как правило использует gamemode по умолчанию (в случае его наличия в системе), однако вы также можете активировать или деактивировать его в параметрах.

Для запуска игры в ручную с использованием gamemode необходимо выполнить команду:

```
gamemoderun ./game
```

Для запуска игр через Steam с использованием gamemode необходимо прописать команду в параметрах запуска игры (находятся в свойствах игры в Steam):

```
gamemoderun %command%
```

Из коробки gamemode применяет не так много оптимизаций, поэтому есть смысл включить использование некоторых параметров, которые отключены по умолчанию. Задействовать их можно создав конфиг для gamemode (комментарии сопровождаются символом `;` в начале):

```
mkdir -p ~/.config/gamemode
nano ~/.config/gamemode/gamemode.ini # Пропишите следующее строчки

[general]
; Повышает приоритет игры до максимума
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
renice=19

; Отключает раздельные блокировки шины памяти.
; Одна инструкция с раздельной блокировкой может занимать шину
; памяти в течение примерно 1 000 тактов, что может приводить к
; кратковременным зависаниям системы в таких играх как God of War.
disable_splitlock=1

; Устанавливает режим работы процессора на максимальную производительность
desiredgov=performance

[gpu]
; Установит профиль вашей видеокарты NVIDIA на максимальную
; производительность на время игры.
nv_powermizer_mode=1

; Аналогично для AMD. Не забывайте следить за температурой вашего GPU!
amd_performance_level=high
```

Предупреждение

Ananicy/Ananicy-cpp и gamemode конфликтуют - не используйте их вместе!

1.13.2.3 Использование DLSS с видеокартами NVIDIA через Proton

Для того чтобы использовать DLSS вам потребуется:

- Видеокарта поддерживающая данную технологию (видеокарты серии RTX и выше).
- Убедиться, что используемая версия Proton не ниже **6.3-8**! (**поддержка DLSS начинается с данной версии!**)
- Указать параметры запуска игры в свойствах игры Steam `PROTON_HIDE_NVIDIA_GPU=0 PROTON_ENABLE_NVAPI=1`
- Некоторые игры, как правило, которые используют DX11, для корректной работы могут также потребовать включения *dxgi.nvapiHack = False* в *dxvk.conf*. Для этого выполните инструкции ниже:

  ```
  mkdir -p ~/.config/dxvk/dxvk.conf
  echo "dxgi.nvapiHack = False" > ~/.config/dxvk/dxvk.conf
  ```

  После этого не забудьте дописать *DXVK_CONFIG_FILE=~/.config/dxvk/dxvk.conf* в приведённом ниже примере перед `%command%`.

Пример для использования в Steam:

```
PROTON_HIDE_NVIDIA_GPU=0 PROTON_ENABLE_NVAPI=1 %command%
```

Внимание

Поскольку для DLSS необходимо специальное машинное обучение, то для запуска необходимо чтобы игра поддерживала DLSS, т.е. в настройках игры должен быть параметр включения данной функции. **Иначе DLSS**

работать не будет!

1.13.2.4 Gamescope

Gamescope - это сессионный композитор, используемый для повышения производительности в играх. По сути, он запускает отдельный менеджер окон специально для вашей игры поверх текущего графического окружения. Преимуществом Gamescope является снижение задержек во время игры и возможность произвольно изменять собственное разрешение окна и разрешение экрана для игры, при этом не меняя исходное разрешение вашего рабочего окружения. У gamescope также есть встроенная поддержка технологий FSR и NVIDIA Image Scaling.

Установка

```
sudo pacman -S gamescope
```

Использование

Прямо перед командой запуска игры (gamescope работает как для Wine, так и для нативных игр) добавьте команду `gamescope`.

Чтобы изменить разрешение в котором будет работать gamescope используйте параметры `-W` и `-H` для ширины и высоты соответственно. Аналогичные параметры есть для указания ширины и высоты окна с игрой `-w` и `-h`.

Используйте параметр `-F` с аргументами `fsr` или `nis` для задействования технологий AMD FSR и NVIDIA Image Scaling соответственно.

Для достижения растягивающего масштабирования используйте `-S stretch`. Например при запуске CS2 с параметрами `gamescope -f -w 2048 -h 1536 -W 3440 -H 1440 -r 165 -S stretch --` получаем картинку 4:3 без черных полос по бокам. `-r` отвечает за герцовку.

Предупреждение

Для правильной работы с закрытым драйвером NVIDIA на последних версиях требуется бета версия Vulkan драйвера 535.43.20.

Предупреждение

Если Gamescope не выводит изображение на видеокартах AMD, используйте переменную окружения `RADV_DEBUG=nodcc` или `R600_DEBUG=nodcc`.

1.13.2.4.1 Запуск gamescope в отдельном tty

Как следует из данного определения, gamescope является сессионным композитором. Это также означает, что он может быть запущен как часть уже запущенной графической сессии (т. е. внутри графического окружения), так и сам представлять собой независимую графическую сессию. Для того, чтобы gamescope смог стать автономным композитором необходимо запустить его в отдельном tty, т.е. вне другой графической среды. По сути такой прием аналогичен запуску отдельного X сервера, но в случае с gamescope это позволяет ему получить некоторые дополнительные возможности, которые нельзя получить при запуске из под графической среды если она сама их не поддерживает (например в случае с GNOME):

- Поддержка VRR (при указании опции `--adaptive-sync`)
- Поддержка прямого отображения кадров минуя дополнительные этапы обработки, что значительно уменьшает задержки ввода (при указании `--immediate-flips`)

Для того чтобы запустить gamescope в таком режиме вам нужно перейти в отдельный, незанятый другой графической сессией, tty, например tty3, при помощи сочетания клавиш `Ctrl+Alt+F3` (последняя цифра указывает номер TTY). После этого перед вами появится приглашение для ввода логина и пароля от вашего пользователя. Если авторизация прошла успешно, то перед вами появится приглашение вашей системной оболочки. В ней вы должны запустить gamescope вместе с указанием приложения, которое вы хотите использовать внутри сессии gamescope. Например:

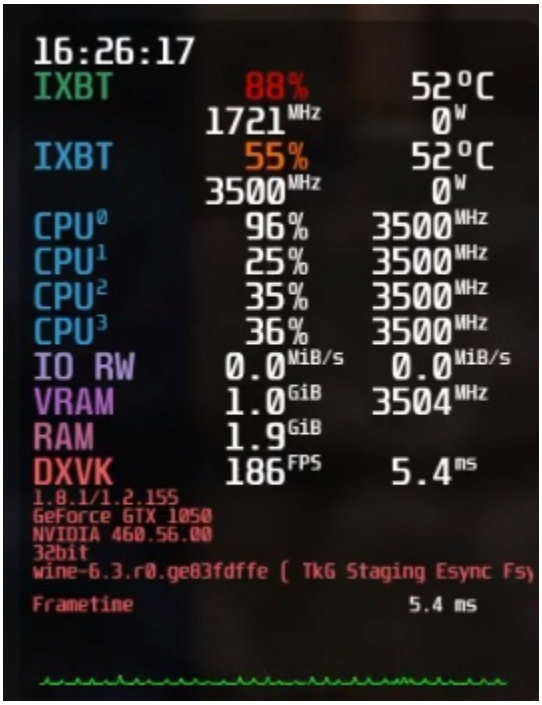
```
gamescope -W 1920 -H 1080 -e -f --xwayland-count 2 --immediate-flips -r 144 -O HDMI-0␣
↪-- wine ~/Games/game.exe
```

Ключи `-W` и `-H` указывают ширину и высоту выбранного ключом `-O` вывода соответственно, а ключ `-r` указывает частоту развертки. После `--` идет команда запуска приложения, которое вы хотите запустить внутри gamescope. Обратите внимание, что вы так же можете запустить клиент Steam внутри gamescope, и все запускаемые вами игры тоже будут работать внутри этой сессии. Специально для Steam также нужно экспортировать переменную окружения (указать перед командой gamescope) `STEAM_MULTIPLE_XWAYLANDS=1` для того чтобы можно было использовать Steam одновременно с вашей обычной графической сессией (возврат на которую осуществляется при помощи сочетания клавиш `Ctrl+Alt+F1`) и gamescope сессией на другом tty.

1.13.2.5 Мониторинг FPS в играх.

1.13.2.5.1 Mangohud

Включение мониторинга в играх как в MSI Afterburner.



Установка

```
sudo pacman -S lib32-mangohud mangohud
```

Графический помощник для настройки вашего MangoHud.

```
sudo pacman -S goverlay
```

Для использования mangohud в играх через Steam необходимо добавить команду в параметры запуска игры (находятся в свойствах игры Steam):

```
mangohud %command%
```

(Для указания нескольких команд необходимо разделять их **пробелом**)

1.13.2.5.2 Альтернатива: DXVK Hud (*Только для игр запускаемых через Wine/Proton*)

Вы также можете использовать встроенную в DXVK альтернативу для мониторинга - DXVK Hud. Он не такой гибкий как MangoHud, но также способен выводить значения FPS, график времени кадра, нагрузку на GPU. Использовать данный HUD можно задав переменную окружения *DXVK_HUD*. К примеру, `DXVK_HUD=fps,frametimes,gpuload` выводит информацию о FPS, времени кадра, и нагрузке на GPU.

Полный список значений переменной вы можете узнать - здесь.

1.13.2.6 Настройка геймпадов Xbox

1.13.2.6.1 Настройка стандартного xpad

К сожалению далеко не все Xbox-совместимые геймпады распознаются встроенным драйвером Xpad при подключении посредством USB, поэтому приходится явно указывать его использование для данных устройств. Делается это при помощи правил Udev (менеджера устройств в Linux). Но перед их написанием нужно определить ID вендора и самого устройства. Это можно сделать через команду `lsusb` (если у вас её нет, то установите пакет `usbutils`):

```
lsusb
```

После вы получите информацию о всех подключенных USB устройствах системе. Нас интересует два числа разделяемых двоеточием, это и будет ID производителя и самого устройства (в примере ниже это `11c1` и `2001` соответственно).

```
...
Bus 001 Device 002: ID 11c1:2001  Controller
...
```

После этого создадим правило udev с произвольным именем файла:

Список 45: `sudo nano /etc/udev/rules.d/10-xbox-gamepad.rules`

```
  # Generic xbox controller
  ACTION=="add", ATTRS{idVendor}=="11c1", ATTRS{idProduct}=="2001", \
      RUN+="/sbin/modprobe xpad", \
      RUN+="/bin/sh -c 'echo %s{idVendor} %s{idProduct} > /sys/bus/usb/drivers/xpad/
↪new_id'"
```

В атрибуты `idVendor` и `idProduct` мы указываем полученные значения из команды lsusb (не забудьте про кавычки!).

Теперь нам нужно загрузить новые правила через следующую команду:

```
sudo udevadm control --reload-rules
```

Переподключите ваш геймпад к компьютеру и он должен стать доступным для использования (проверить можно через наличие файла `/dev/input/js0`).

1.13.2.6.2 Драйвер xpadneo с поддержкой Bluetooth

И драйвер xpad, и драйвер xone не поддерживают работу с геймпадами, работающими через Bluetooth. Xpadneo - это новый драйвер для поддержки работы всех последних контроллеров от Microsoft, например таких как контроллер Microsoft Xbox One S, и их клонов, подключаемых через Bluetooth.

Среди преимуществ драйвера следует отметить следующее:

- Поддержка Trigger Force Feedback (чего даже нет в Microsoft Windows)
- Поддержка подключения более одного контроллера
- Поддержка индикации уровня заряда

Полный список возможностей драйвера можно увидеть здесь.

Установка

```
git clone https://aur.archlinux.org/xpadneo-dkms
cd xpadneo-dkms
makepkg -sric
```

1.14 Оконный менеджер river

Некоторое программное обеспечение (ПО) на первый взгляд имеет сложную документацию. Один из таких представителей оконный менеджер - river.

Данный менеджер окон (WM) - композитор для Wayland, в отличии от других представителей данной области располагает довольно скудной документацией с помощью которой сложно разобраться во многих тонкостях его настройки. Часть информации о настройке river вы можете найти на вики river, но большая часть расположена в мануалах для river, riverctl, rivertile, которые могут быть доступны из системы командой, если у вас установлен **man**:

```
man river
man riverctl
man rivertile
```

Однако в данных мануалах не все так очевидно с первого раза, как бы хотелось, но их использование или ознакомление с ними рекомендуется.

Предупреждение

Данное руководство основано на данных из мануалов и опыте взаимодействия с river. Руководство построено на основе примеров команд - адаптируйте их под свои нужды.

1.14.1 Конфиг river

В отличие от многих других WM, river в качестве своего конфигурационного файла использует исполняемый файл с именем **init**, поэтому настраивать river можно как с помощью скрипта оболочки, так и с помощью программ написанных на других языках программирования. По умолчанию используется скрипт оболочки - в этом можно убедиться по наличию надписи в первой строке **#!/bin/sh** файла **init**. Еще одной особенностью подхода конфигурационного файла river является возможность изменения его параметров непосредственно из работающей системы без необходимости перезапуска WM или перечитывания данных из файла **init**. Данная возможность полезна в случае, если вы хотите проверить некоторое изменение быстро, например - новую комбинацию клавиш для вызова терминала - достаточно ввести в терминале команду:

```
riverctl map normal Super T spawn foot
```

Разберем представленную команду:

- *riverctl* - команда для определения параметров в river,
- *map* - параметр отвечающий за определения комбинаций клавиш,
- *normal* - параметр указывающий на режим работы (в river 2 режима: основной - *normal*, и режим на случай блокировки экрана - *locked*),
- *foot* - здесь отображает программу, которую нужно запустить, также можно задать команду для выполнения, например **'reboot'**. В случае определения команды одиночные кавычки обязательны,
- *spawn* - команда отвечающая за выполнение, *Super* и *T* - указывают на клавиши которые нужно нажать. Если используются несколько управляющих клавиш (*Super, Alt, Shift, Control* и т.п.) то между ними нужно добавлять знак **+**, например для команды в предыдущем примере это будет выглядеть следующим образом:

```
riverctl map normal Super+Shift T spawn foot
```

Предупреждение

Вероятно, что комбинации только из служебных клавиш обозначить нельзя! (Super Shift, Alt Shift и т.п.)

Предупреждение

Не смотря на то что river позволяет вносить изменения в настройки в режиме "реального времени", для того чтобы изменения использовались на постоянной основе, а не только в данной сессии - изменения необходимо добавлять(производить) в файл(е) **init**!

1.14.2 Настройка river

Приступим к настройке river. Для начала стоит отметить, что конфигурационный файл **init** необходимо располагать в:

```
$HOME/.config/river/
```

Пример конфигурационного файла в зависимости от вашей системы может быть расположен:

```
/etc/river/init
/usr/share/river/example/init
```

Если по указанным путям его нет и вы не знаете где его найти, то можно получить его из репозитория river. Если вы хотите использовать другой формат для файла **init**, то есть примеры на wiki river - мы их пока не затрагиваем.

1.14.2.1 Назначение клавиш

Чтобы определить комбинацию клавиш уже был представлен пример команды, однако в случае если у вас используется несколько раскладок клавиатур, например английская и русская, то на русской раскладке часть комбинаций не будет работать. Это происходит потому что по умолчанию river использует *keysym*, а не *keycode* (у англоязычных и русскоязычных букв *keysym* - разный, а *keycode* - одинаковый) Чтобы решить данную проблему необходимо использовать *-layuot*:

```
riverctl map -layuot 0 normal Super T spawn foot
#                    |
#              номер раскладки
```

Предупреждение

Данная комбинация не включает использование *keycode*, она просто говорит, что в качестве привязанных клавиш нужно использовать раскладку 0, т.е. первую указанную в "us, ru". Будьте внимательны!

Стоит отметить, что те клавиши, которые не зависят от *keysym* (*Super, Alt, Control, Return* и прочие служебные клавиши) не нуждаются в добавлении *-layuot*, например следующая команда будет работать в любой раскладке:

```
riverctl map normal Super Return spawn foot
```

В river, как и в других WM, можно привязывать к клавишам целые команды, для этого нужно добавить одиночные кавычки:

```
riverctl map normal Super Return spawn 'foot --app-id=foobar'
```

Для того чтобы убрать привязку клавиш в работающей системе, используйте *unmap*, например:

```
riverctl unmap normal Super D
```

Предупреждение

Данная команда введенная в терминале сработает только для текущей сессии, если вы хотите внести изменения на постоянной основе - редактируйте **init** для последующих запусков.

В river для определения взаимодействия с окнами с помощью мыши используется *map-pointer*, например:

Для перемещения окна с помощью мыши:

```
riverctl map-pointer -layuot 0 normal Super BTN_LEFT move-view
```

Предупреждение

Окна в river определены как *виды* - **view**.

Предупреждение

При перемещении не "плавающего" окна, т.е. из **rivertile** на 13.02.2025 заданные размеры окна игнорируются - берутся те размеры, что сейчас на экране!

Для изменения размера окна с помощью мыши:

```
riverctl map-pointer -layuot 0 normal Super BTN_RIGHT resize-view
```

Для переключения режима "плавающего" окна с помощью мыши:

```
riverctl map-pointer -layuot 0 normal Super BTN_MIDDLE toggle-float
```

Чтобы удалить привязку, например:

```
riverctl unmap-pointer normal Super BTN_RIGHT
#                          |      |         |
#                       режим  Модификатор Кнопка мыши
```

1.14.2.2 Настройка раскладки клавиатуры

Для настройки смены раскладок клавиатуры:

```
riverctl keyboard-layout -model pc105 -variant qwerty -options "grp:win_space_toggle"
↪"us, ru"
# или
riverctl keyboard-layout -options "grp:win_space_toggle" "us, ru"
```

Разберем допустимые параметры:

- *rules* - не присутствует в данном примере, не является обязательным, позволяет определять правила, на данный момент имейте в виду что такой параметр есть.
- *keyboard-layout* - определение слоев клавиатуры,
- *model* - здесь задается тип клавиатуры - сколько клавиш (указывать не обязательно данный параметр),
- *variant* - форматов раскладок несколько, но все привыкли к формату *qwerty* (указывать не обязательно данный параметр),
- *options* - тут указывается как переключать раскладку клавиатур, согласно *xkb*,
- *"us, ru"* - в кавычках указываются все нужные раскладки.

Предупреждение

После обновления предположительно **setxkbmap**, вариант с указанием всех параметров может перестать работать, обращайте на это внимание! Что касается сокращенного варианта - похожих случаев не зафиксировано и позволяет решить описанную проблему с большим количеством параметров, но имейте в виду.

Предупреждение

По умолчанию в river каждая клавиатура расценивается как индивидуальное устройство ввода и раскладка клавиатуры переключается для каждой клавиатуры в отдельности, конкретной клавиатурой.

Если вам нужно чтобы на всех клавиатурах раскладка переключалась одновременно, то для этого есть *группы*, которые нужно создать и добавить туда все нужные устройства ввода. Для работы с *группами* устройств ввода есть несколько действий:

1. Создать группу:

   ```
   riverctl keyboard-group-create _имя_группы_
   ```

2. Добавить устройство в группу:

```
riverctl keyboard-group-add _имя_группы_ _имя_устройства_
```

Для того чтобы узнать _имя_устройства_ ввода:

```
riverctl list-inputs
```

3. Для удаления устройства из группы:

```
riverctl keyboard-group-remove _имя_группы_ _имя_устройства_
```

4. Для удаления группы:

```
riverctl keyboard-group-destroy _имя_группы_
```

Настроить раскладки для всех клавиатур можно через файл согласно документации XKB:

```
riverctl keyboard-layout-file _путь_к_файлу_
```

1.14.2.3 Настройка медиа-клавиш или устройств

Для настройки медиа клавиш и устройств в конфиге river есть подобный фрагмент:

```
for mode in normal locked
do
  # Eject the optical drive (well if you still have one that is)
  riverctl map $mode None XF86Eject spawn 'eject -T'

  riverctl map $mode None XF86AudioRaiseVolume  spawn 'wpctl set-volume @DEFAULT_
↪SINK@ 2%+'
  riverctl map $mode None XF86AudioLowerVolume  spawn 'wpctl set-volume @DEFAULT_
↪SINK@ 2%-'
  riverctl map $mode None XF86AudioMute         spawn 'wpctl set-mute @DEFAULT_SINK@␣
↪toggle'

  # Control MPRIS aware media players with playerctl (https://github.com/altdesktop/
↪playerctl)
  riverctl map $mode None XF86AudioMedia spawn 'playerctl play-pause'
  riverctl map $mode None XF86AudioPlay  spawn 'playerctl play-pause'
  riverctl map $mode None XF86AudioPrev  spawn 'playerctl previous'
  riverctl map $mode None XF86AudioNext  spawn 'playerctl next'

  # Настройка яркости экрана через brightnessctl (https://github.com/Hummer12007/
↪brightnessctl)
  riverctl map $mode None XF86MonBrightnessUp   spawn 'brightnessctl set +5%'
  riverctl map $mode None XF86MonBrightnessDown spawn 'brightnessctl set 5%-'
done
```

Если вы не используете **pipewire** и **wireplumber**, то для регулировки звука замените **wpctl** на **pamixer** или **pactl** согласно документации их использования. (По умолчанию в конфиге river указан **pamixer**)

Как можно видеть из примера в случае специальных медиа клавиш или крутилок или других спец. клавиш в обозначении появляется конструкция **$mode None XF86Audio...** - данный формат специально предназначен для подобных настроек.

1.14.2.4 "Тайлинг" в river

Для реализации "тайлинга", т.е. расположения окон на экране в river, используется **rivertile** - это так называемый генератор слоев. По умолчанию **rivertile** работает в стековом режиме, т.е. есть одна главная область и вторая второстепенная, куда "складываются" остальные окна в своеобразный "столбик".

Особенность river в том, что он позволяет заменить rivertile на другой, тем самым позволяет с легкостью изменить формат разметки "тайлинга". На wiki river предлагаются следующие варианты:

- riverguile
- owm
- kile
- stacktile
- rivercarro
- river-luatile
- river-bsp-layout
- river-dwindle
- filtile
- wideriver
- river-ultitile
- river-spiral-extended

Чтобы указать какой генератор макетов использовать по умолчанию для всех экранов:

```
riverctl default-layout rivertile   # rivertile используется по
умолчанию в river - замените на другой
```

Чтобы указать другое размещение окон на активном экране используйте:

```
riverctl output-layout rivertile    # rivertile замените на любой
другой
```

Для отправки команд генератору разметки есть подобный пример:

```
# Команда с привязкой к "горячим" клавишам
riverctl map -layout 0 normal Super K send-layout-cmd rivertile "main-count +1"

# Для команды из терминала
send-layout-cmd rivertile "main-count +1"
```

rivertile - тут в качестве примера генератора, а выражение в "" команда которую нужно обработать - должна соответствовать документации генератора

1.14.2.4.1 Настройка параметров rivertile

В **rivertile** можно изменить расстояние между окнами:

```
rivertile -view-padding N     # N - количество пикселей
```

Для изменения расстояния до краев окна:

```
rivertile -outer-padding N    # N - количество пикселей
```

Для выбора места - где будет располагаться основной сегмент:

```
rivertile -main-location PAR   # PAR = [top | bottom | left | right] - указывается␣
↪один из представленных
```

Чтобы изменить соотношение "полезного" пространство для основного сегмента:

```
rivertile -main-ratio N        # N - значение от 0.1 до 0.9
```

Определить количество видов в основном пространстве:

```
rivertile -main-count N        # N - целое значение, по умолчанию 1
```

1.14.2.4.2 Взаимодействие с окнами

Для переключения активного окна используются:

1. Выбрать следующее окно:

   ```
   riverctl map -layout 0 normal Super J focus-view next
   ```

2. Выбрать предыдущее окно:

   ```
   riverctl map -layout 0 normal Super K focus-view previous
   ```

Для переноса окна в стек и обратно:

```
riverctl map -layout 0 normal Super+Shift Return zoom
```

Переключение режимов окна:

1. Переключение окна в "плавающий режим" и обратно:

   ```
   riverctl map -layout 0 normal Super Space toggle-float
   ```

2. Переключение окна в полноэкранный режим и обратно:

   ```
   riverctl map -layout 0 normal Super F toggle-fullscreen
   ```

Для перемещения окон:

1. Переместить окно следующую позицию:

   ```
   riverctl map -layout 0 normal Super+Shift J swap next
   ```

2. Переместить окно на предыдущую позицию:

   ```
   riverctl map -layout 0 normal Super+Shift K swap previous
   ```

Предупреждение

Поддерживаются также значения *up, down, left, right*.

Для того чтобы перемещать "плавающее" окно:

```
riverctl map -layout 0 normal Super+Alt H move left 100     # переместит окно влево
riverctl map -layout 0 normal Super+Alt J move down 100     # переместит окно вниз
riverctl map -layout 0 normal Super+Alt K move up 100       # переместит окно вверх
riverctl map -layout 0 normal Super+Alt L move right 100    # переместит окно вправо
```

100 - показывает шаг перемещения в пиксилях

Предупреждение

Если окно не было "плавающим", то оно перейдет в данный режим!

"Плавающие" окна можно переместить к одной из сторон экрана например:

```
riverctl map -layout 0 normal Super+Alt+Control H snap left     # левый край
riverctl map -layout 0 normal Super+Alt+Control J snap down     # нижний край
riverctl map -layout 0 normal Super+Alt+Control K snap up       # верхний край
riverctl map -layout 0 normal Super+Alt+Control L snap right    # правый край
```

Для перемещения окна в угол экрана выполните две команды последовательно, например для левого угла переместите окно к левой границе экрана и вверх, либо наоборот. Также вы можете обозначить данные действия на одну клавишу с помощью *&&*.

1.14.2.4.3 Управление размерами окон

Для изменения размеров окон используются следующие команды:

1. Уменьшение размера основного окна:

   ```
   riverctl map -layout 0 normal Super H send-layout-cmd rivertile "main-ratio -0.05"
   ```

2. Увеличение размера основного окна:

   ```
   riverctl map -layout 0 normal Super H send-layout-cmd rivertile "main-ratio +0.05"
   ```

Предупреждение

Увеличение / уменьшение основного сектора приводит к уменьшению / увеличению размеров стека соответственно!

Для "плавающих" окон:

1. Уменьшение по горизонтали:

   ```
   riverctl map -layout 0 normal Super+Alt+Shift H resize horizontal -100
   ```

2. Увеличение по вертикали:

   ```
   riverctl map -layout 0 normal Super+Alt+Shift J resize vertical 100
   ```

3. Уменьшение по вертикали:

```
riverctl map -layout 0 normal Super+Alt+Shift K resize vertical -100
```

4. Увеличение по горизонтали:

```
riverctl map -layout 0 normal Super+Alt+Shift L resize horizontal 100
```

100 - показывает шаг изменения размера в пиксилях

Предупреждение

Изменение размеров "плавающих" окон происходит с двух сторон!

1.14.2.5 Настройка tags (рабочих столов)

В river как и в dwm рабочие столы определены не как **workspace**, а как **tag**. Как утверждается **tags** более гибки в настройке и одному приложению можно определять несколько **tags**. Также одновременно на экран можно выводить несколько разных **tags**.

В конфиге river есть блок, который определяет клавиши для выбора и управления **tags**, если вы хотите изменить клавиши, то вам поможет следующий пример:

```
# Определяем клавиши для переключения
tg=('q' 'w' 'e' 'r' 't' 'y' 'u' 'i' 'o')

# Определяем команды для каждого tag
for i in $(seq 1 9)                # задается количество tags, по умолчанию до 9 штук
↪на экран
do
  tags=$((1 << ($i - 1)))          # определяются tags

  # Super+[q-o] выбор tag [0-8]
  riverctl map -layout 0 normal Super ${tg[$i-1]} set-focused-tags $tags

  # Super+Shift+[q-o] перемещение окна на tag [0-8]
  riverctl map -layout 0 normal Super+Shift ${tg[$i-1]} set-view-tags $tags

  # Super+Control+[q-o] выбор - окна какого tag [0-8] добавить для отображения на
↪экране
  riverctl map -layout 0 normal Super+Control ${tg[$i-1]} toggle-focused-tags $tags

  # Super+Shift+Control+[1-9] дублировать окно на tag [0-8]
    riverctl map -layout 0 normal Super+Shift+Control ${tg[$i-1]} toggle-view-tags
↪$tags
done
```

Предупреждение

Данный пример приведен, если вы хотите изменить клавиши для **tags** на буквы, в конфиге river по умолчанию используются числа от 1 до 9.

В river есть команды для взаимодействия с окнами на всех **tags**:

```
all_tags=$(((1 << 32) - 1))

# Вывести на экран все окна со всех tags
riverctl map -layout 0 normal Super 0 set-focused-tags $all_tags

# Определить для активного окна все tags - окно будет перемещаться вместе с переходом␣
↪на другой tag
riverctl map -layout 0 normal Super+Shift 0 set-view-tags $all_tags
```

В river, в отличие от sway и части других WM, по умолчанию **tags** не переносятся на другие мониторы, т.е. для каждого экрана создаются свои независимые **tags**, взаимодействия с которыми требуют переключения на другой монитор:

```
riverctl map -layout 0 normal Super L focus-output next       # переключиться на␣
↪следующий монитор
riverctl map -layout 0 normal Super H focus-output previous   # переключиться на␣
↪предыдущий монитор
```

Помимо предыдущего (*previous*) и следующего (*next*) мониторов доступны также - *up, right, down, left* или имена выводов - *HDMI-1, DP-1 и т.п.*

Для переноса активных окон с одного монитора на другой используется команда:

```
riverctl map -layout 0 normal Super+Shift L send-to-output next      # перенести на␣
↪следующий монитор активное окно
```

Поддерживаются также - *previous, up, down, right, left* и имена мониторов - *HDMI-1, DP-1 и т.п.*.

Предупреждение

По умолчанию при перемещении активных окон сохраняется № использованного *tag* и после перемещения на другой монитор активное окно будет на *tag* под тем же номером.

1.14.2.6 Настройка экранов(мониторов)

В river для управления экранами (мониторами) используется внешний софт, на wiki river рекомендуются следующие:

- wlopm
- wlr-randr
- kanshi
- way-displays

На данный момент будет описано использование **wlr-randr**. Для начала необходимо знать какие выводы у вас используются. Для этого достаточно запустить команду:

```
wlr-randr
```

Данная команда покажет вам всю доступную информацию о подключенных мониторах и настройках, например:

```
DP-2 "AOC Q27G2WG4 0x0000DAB3 (DP-2)"
  Make: AOC
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
  Model: Q27G2WG4
  Serial: 0x0000DAB3
  Physical size: 600x340 mmSwayNotificationCenter
  Enabled: yes
  Modes:
    2560x1440 px, 59.951000 Hz (preferred)
    2560x1440 px, 143.912003 Hz (current)
    2560x1440 px, 119.998001 Hz
    2560x1440 px, 99.945999 Hz
    1920x1080 px, 119.878998 Hz
    1920x1080 px, 60.000000 Hz
    1920x1080 px, 59.938999 Hz
    1920x1080 px, 50.000000 Hz
    1280x1440 px, 59.912998 Hz
    1280x1024 px, 75.025002 Hz
    1280x1024 px, 60.020000 Hz
    1440x900 px, 59.901001 Hz
    1280x720 px, 59.943001 Hz
    1280x720 px, 50.000000 Hz
    1024x768 px, 119.988998 Hz
    1024x768 px, 99.972000 Hz
    1024x768 px, 75.028999 Hz
    1024x768 px, 70.069000 Hz
    1024x768 px, 60.004002 Hz
    800x600 px, 119.972000 Hz
    800x600 px, 99.662003 Hz
    800x600 px, 75.000000 Hz
    800x600 px, 72.188004 Hz
    800x600 px, 60.317001 Hz
    800x600 px, 56.250000 Hz
    720x576 px, 50.000000 Hz
    720x480 px, 59.939999 Hz
    640x480 px, 119.517998 Hz
    640x480 px, 99.768997 Hz
    640x480 px, 75.000000 Hz
    640x480 px, 72.808998 Hz
    640x480 px, 59.939999 Hz
    640x480 px, 59.929001 Hz
  Position: 0,0
  Transform: normal
  Scale: 1.000000
  Adaptive Sync: disabled
DP-1 "AOC Q27G2SG4 XFXQ7HA001584 (DP-1)"
  Make: AOC
  Model: Q27G2SG4
  Serial: XFXQ7HA001584
  Physical size: 600x340 mm
  Enabled: yes
  Modes:
    2560x1440 px, 59.951000 Hz (preferred)
    2560x1440 px, 155.000000 Hz
    2560x1440 px, 143.912003 Hz (current)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
    2560x1440 px, 119.998001 Hz
    1920x1080 px, 119.878998 Hz
    1920x1080 px, 60.000000 Hz
    1920x1080 px, 59.938999 Hz
    1920x1080 px, 50.000000 Hz
    1280x1440 px, 59.912998 Hz
    1280x1024 px, 75.025002 Hz
    1280x1024 px, 60.020000 Hz
    1280x720 px, 59.943001 Hz
    1280x720 px, 50.000000 Hz
    1024x768 px, 119.988998 Hz
    1024x768 px, 99.972000 Hz
    1024x768 px, 75.028999 Hz
    1024x768 px, 70.069000 Hz
    1024x768 px, 60.004002 Hz
    800x600 px, 119.972000 Hz
    800x600 px, 99.662003 Hz
    800x600 px, 75.000000 Hz
    800x600 px, 72.188004 Hz
    800x600 px, 60.317001 Hz
    800x600 px, 56.250000 Hz
    720x576 px, 50.000000 Hz
    720x480 px, 59.939999 Hz
    640x480 px, 119.517998 Hz
    640x480 px, 99.768997 Hz
    640x480 px, 75.000000 Hz
    640x480 px, 72.808998 Hz
    640x480 px, 59.939999 Hz
    640x480 px, 59.929001 Hz
  Position: 2560,0
  Transform: normal
  Scale: 1.000000
  Adaptive Sync: disabled
```

Для настройки экранов **wlr-randr** использует ряд параметров:

```
--output name             # выбор экрана для настройки (name - HDMI, DP и т.п.)
# Следующие параметры указываются после --output name для конкретного монитора
  --mode WxH@XHz          # определяет разрешение и частоту экрана (W - ширина, H -␣
↪высота в px) и частоту (XHz - частота в Hz. X - должен точно соответствовать␣
↪имеющимся значениям)
  --pos X,Y               # расположение одного экрана относительно другого X -␣
↪смещение по X, Y - смещение по Y
  --adaptive-sync X       # включение режима переменной частоты обновления монитора␣
↪(X - enabled или disabled)
  --scale X               # масштабирование изображения (X - 1.0, 1.5 и т.п.)
  --transform X           # поворот изображения на экране (90, 180, normal, flipped,␣
↪flipped-90 и т.п.)
  --on                    # включение
  --off                   # выключение
  --toggle                # переключение (вкл. / выкл.)
```

Пример настройки экранов через **wlr-randr**:

```
wlr-randr --output DP-2 --mode 2560x1440@143.912003Hz --pos 0,0 --adaptive-sync␣
↪enabled --output DP-1 --mode 2560x1440@143.912003Hz --pos 2560,0 --adaptive-sync␣
↪disabled
```

В Wayland вывод изображения синхронизируется с частотой обновления монитора, что может создавать задержку ввода. Для решения данной проблемы был реализован протокол **tearing**. Для его включения в полноэкранных программах:

```
riverctl allow-tearing enabled
```

1.14.2.7 Настройка управления программами / приложениями

Организация работы с программами / приложениями в river производится через *правила (rules)*. Данный компонент позволяет определять: на каком мониторе будет открываться программа, на какой *tag* будет выводиться по умолчанию, будет ли использоваться *tearing* - независимо от заданного глобального параметра, задавать размер окна программы и его расположения, будет ли окно открываться как *плавающее*, будет ли программа открываться в полноэкранном режиме - в общем отвечает за все основные параметры. Взаимодействовать с *rules* можно тремя способами:

1.14.2.7.1 Добавить правило

Предупреждение

В river добавлять правило можно только по одному. Не стоит пытаться объединять все в одну строку, если конечно вы не пишите несколько команд через *&&*.

Правило определяющее *tag* для программы

Чтобы *firefox* открывался на третьем *tag* нужно указать правило:

```
riverctl rule-add -app-id "firefox" tags $((1 << 2))
```

Предупреждение

Стоит отметить, что в river, как в части языков программирования, нумерация начинается с 0, а не с 1, поэтому в выражение **$((1 << N))**, **N** - должно быть на 1 меньше чем номер *tag* на который вы хотите определить отображение программы по умолчанию. Зная данный факт можно заменить выражение на **$((1 << N - 1))** - это позволит указывать **N** согласно номеру *tag*.

Для того чтобы river понимал какую программу он должен обработать и отобразить на определенном *tag* используются такие параметры как **app-id** или **title**. Данные параметры можно задавать при запуске программы через *riverctl*:

```
riverctl spawn 'foot -app-id=terminal'
riverctl spawn 'foot -title=terminal'
```

Предупреждение

Если **title** не задано пользователем, то оно не задается системой самостоятельно, поэтому лучше ориентироваться на **app-id**.

Предупреждение

Во многих случаях **app-id** соответствует названию программы, но далеко не всегда так: например blender дополнительно в **app-id** требует указание версии, например **blender-4.3**, иначе на него правило работать не будут!

Для того, чтобы определить **title** и **app-id** запущенной программы, можно использовать lswt. Пример результата:

```
state:   app-id:                  title:
----     libreoffice-startcenter  LibreOffice
--a-     foot                     foot
----     foot                     foot
----     firefox                  "river/wiki: The river Wayland compositor wiki -␣
↪Codeberg.org — Mozilla Firefox"
----     foot                     foot
```

Правило определяющее монитор для программы

Чтобы указать монитор на котором стоит отображать программу по умолчанию, можно использовать следующий пример:

```
riverctl rule-add -app-id "steam" output DP-2
```

Для определения имени "вывода" (экрана) воспользуйтесь одной из программ, упомянутых в **Настройка экранов (мониторов)**.

Правило регулирующее отображение программы в полноэкранном режиме

Для отображения по умолчанию программы в полноэкранном режиме достаточно использовать параметр *fullscreen*:

```
riverctl rule-add -app-id "foot" fullscreen
```

Для того чтобы сделать исключение используется *no-fullscreen*:

```
riverctl rule-add -app-id "foot" no-fullscreen
```

Правило для плавающего окна

Для запуска программы в режиме "плавающего" окна по умолчанию - используйте *float*:

```
riverctl rule-add -app-id "firefox" float
```

Для отмены или исключения:

```
riverctl rule-add -app-id "firefox" no-float
```

Правила размера и расположения

Чтобы задать размер окна программы:

```
riverctl rule-add -app-id "foot" dimensions W H    # W - ширина, H - высота
```

Предупреждение

Стоит отметить, что размер окна работает в первую очередь для "плавающих" окон, поскольку "не плавающие" окна регулируются макетом **rivertile**.

Чтобы задать позицию окна программы:

```
riverctl rule-add -app-id "foot" position W H      # W - координата по ширине, H -␣
↪координата по высоте
```

Предупреждение

Стоит отметить, что "начало координат" (точка X = 0, Y = 0) находится в верхнем левом углу. И в первую очередь указание размера необходимо для "плавающих" окон.

Правило отвечающее за tearing

Для того, чтобы задать конкретной программе использовать *tearing* вне зависимости от глобального параметра:

```
riverctl rule-add -app-id "firefox" tearing
```

Для отключения *tearing* для конкретной программы:

```
riverctl rule-add -app-id "firefox" no-tearing
```

Правило указывающее на управляющего по декорациям окна

В Wayland многое в графике программ переложено на сами клиенты, а не композиторы. В river есть два параметра которые определяют кто должен отвечать за внешний вид окон программ:

1. Клиент сам занимается декорацией окна, заголовка и т.п.:

   ```
   riverctl rule-add -app-id "foot" csd
   ```

2. river занимается декорацией окна, заголовка и т.п.:

   ```
   riverctl rule-add -app-id "foot" ssd
   ```

Предупреждение

river не отвечает за то что и как рисуется в самом окне, но может например скрыть заголовок окна терминала, если он является управляющим. По умолчанию используется режим ssd для многих программ.

Настройка внешнего вида river

В river настройка внешнего вида ограничена несколькими параметрами:

Чтобы задать цвет фона:

```
riverctl background-color 0x000000ff
```

Для указания цвета в river используется один из двух вариантов обозначений:

1. 0xRRGGBBAA - данный формат описывает все цвета на основе комбинации трех цветов - красный (RR), зеленый (GG) и синий (BB) и также задает прозрачность (AA). Все четыре параметра могут изменяться от 00 до 99 и от aa до ff, где aa - является следующим за 99,т.е. ff - наибольшее значение, а 00 - наименьшее.
2. 0xRRGGBB - данный формат такой же как и первый, но без прозрачности, т.е. AA всегда равен ff

Задать цвет границ активного окна:

```
riverctl border-color-focused 0xffffffff
```

Задать цвет границ неактивного окна:

```
riverctl border-color-unfocused 0x555555ff
```

Задать цвет "срочных представлений":

```
riverctl border-color-urgent 0x660022ff
```

Предупреждение

"Срочные представления" на данный момент не совсем понятны. Мы работаем над тем, чтобы дать более точную информацию.

Задать ширину рамки окна:

```
riverctl border-width N                    # N - целое значение пикселей
```

Чтобы разместить обои на экраны необходимо использовать сторонний софт, на wiki river рекомендуют:

- swaybg
- wbg
- sww

Стоит отметить, что в Wayland мониторы не объединяются в один единый рабочий стол или виртуальный экран, поэтому для вывода разных изображений на разные экраны нужно использовать программы, которые это поддерживают, иначе изображение будет повторятся на всех экранах. Также вероятно вам придется разрезать изображение под каждый экран, если вы хотите разместить изображение размером с объединенные экраны. Приведем пример использования *swaybg*:

```
riverctl spawn 'swaybg -o DP-1 -i /path_to_image1/img1.jpg -o DP-2 -i /path_to_image2/
↪img2.png'

# или

exec swaybg -o DP-1 -i /path_to_image1/img1.jpg -o DP-2 -i /path_to_image2/img2.png
```

Настройки для курсора

Изменение активного окна с помощью курсора мыши:

```
riverctl focus-follow-cursor _параметр_
```

Допустимые значения _параметр_:

- *disabled* - не менять активное окно при перемещении курсора,
- *normal* - перемещение приводит к переключению активного окна, но не переключается на данное окно, если курсор перемещается внутри окна непокидая его,
- *always* - всегда переключать активное окно вслед за курсором мыши.

Способы скрыть курсор мыши:

1. Скрывать курсор мыши через T милисекунд:

   ```
   riverctl hide-cursor timeout T
   ```

2. Скрывать курсор мыши при нажатии не специальных клавиш:

   ```
   riverctl hide-cursor when-typing disabled     # enabled | disabled
   ```

Положение курсора при перемещении на другой экран или другое окно:

```
riverctl set-cursor-warp disabled
```

Вместо *disabled* доступны:

- *on-output-change* - расположить курсор в центре экрана, при переключении с клавиатуры на другой экран
- *on-focus-change* - расположить курсор в центре активного окна, при переключении с клавиатуры на другой экран или окно

Задать тему и размер курсора:

```
riverctl xcursor-theme _тема_курсора_ _размер_курсора_    # _размер_курсора указывать␣
↪не обязательно
```

1.14.2.7.2 Удаление правил

Любое правило можно удалить из запущенной системы:

```
riverctl rule-del -app-id "foot" ssd # ssd для примера, параметры частично описаны␣
↪выше, но также их получение описано далее.
```

Для указание на программу можно использовать **app-id** или **title**, главное чтобы они были определены.

1.14.2.7.3 Просмотр заданных правил

Если правила были заданы, то вполне логично что их нужно как-то просмотреть и проверить. Для этого есть команда:

```
riverctl list-rules _параметр_
```

*_параметр_* может принимать следующие значения - *dimensions, float, fullscreen, output, position, ssd, tags*

Предупреждение

Параметры для *tearing* по какой-то причине проверить нельзя!

1.14.2.8 Особые режимы / события в river

В river предусмотрены такие события как: *lid* - крышка и *tabled* - планшет. Для их определения используется команды:

1. При закрытии крышки:

```
riverctl map-switch normal lid close _любая команда через riverctl_
```

2. При открытии крышки:

```
riverctl map-switch normal lid open _любая команда через riverctl_
```

3. При включении:

```
riverctl map-switch normal tabled on _любая команда через riverctl_
```

4. При выключении:

```
riverctl map-switch normal tabled off _любая команда через riverctl_
```

Для удаления заданных параметров, например:

```
riverctl unmap-switch normal lid close

riverctl unmap-switch normal tabled off
```

Не смотря на то, что в river по умолчанию только два режима: *normal* и *locked*, есть возможность определять / создавать другие режимы и переключаться между ними, например в конфиге river есть такой фрагмент:

```
# Объявления проходного режима. Этот режим имеет только одно отображение для возврата␣
↪в обычный режим. Это делает его полезным для тестирования вложенного композитора␣
↪wayland
riverctl declare-mode passthrough

# Super+F11 войти в режим passthrough
riverctl map normal Super F11 enter-mode passthrough

# Super+F11 вернутся в режим normal
riverctl map passthrough Super F11 enter-mode normal
```

1.14.2.9 Параметры для настройки устройств ввода

В river есть ряд параметров которые предназначены для настройки специализированных значений или поведений устройств ввода (*input*).

Чтобы настроить будут ли "события" устройства ввода использоваться в river:

```
riverctl input _имя_устройства_ events enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить,
3. *disabled-on-external-mouse* - выключить тачпад при подключении внешней мыши.

Предупреждение

Для определения _имя_устройства_ используйте **list-inputs**. Пример использования был описан выше в **Настройка раскладки клавиатуры**.

Чтобы указать профиль ускорения указателя устройства ввода:

```
riverctl input _имя_устройства_ accel-profile none
```

Допустимые значения:

1. *none* - нет профиля,
2. *flat* - плоский профиль - ко всем дельтам устройства ввода применяется постоянный коэффициент, независимо от скорости движения,
3. *adaptive* - адаптивный профиль - профиль, который учитывает текущую скорость устройства при определении ускорения.

Чтобы определить коэффициент ускорения указателя устройства ввода:

```
riverctl input _имя_устройства_ pointer-accel 1.0
```

Возможные значения от -1.0 до 1.0.

Режим работы тачпада / кликпада:

```
riverctl input _имя_устройства_ click-method none
```

Допустимые значения:

1. *none* - нет режима,
2. *button-areas* - нижняя область тачпада делится на область левой, средней и правой кнопок,
3. *clickfinger* - количество пальцев на тачпаде определяет тип кнопки.

Для активиции или отключения функции перетаскивания тачпадом:

```
riverctl input _имя_устройства_ drag enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить.

Для включения или отключения функции блокировки перетаскивания тачпадом:

```
riverctl input _имя_устройства_ drag-lock enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить.

Для включения или отключения функции отключения тачпада во время набора текста:

```
riverctl input _имя_устройства_ disable-while-typing enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить.

Для включения или отключения функции отключения тачпада при использовании трекпоинта:

```
riverctl input _имя_устройства_ disable-while-trackpointing enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить.

Для включения эмуляции средней кнопки мыши на тачпаде:

```
riverctl input _имя_устройства_ middle-emulation enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить.

Чтобы включить инверсию прокрутки:

```
riverctl input _имя_устройства_ natural-scroll enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить.

Чтобы определить скорость прокрутки:

```
riverctl input _имя_устройства_ scroll-factor 2
```

Допускаются значения больше 0. Значения < 1 приведут к замедлению скорости прокрутки, когда значения > 1 - к ускорению.

Режим работы "для левшей" для мыши, тачпада и т.п.:

```
riverctl input _имя_устройства_ left_handed enabled
```

Допустимые значения:

1. *enabled* - включить,
2. *disabled* - выключить.

Включить или отключить функцию "тап" для тачпада:

```
riverctl input _имя_устройства_ tap enabled
```

Допустимые значения:

1. *enabled* - включить,

2. *disabled* - выключить.

Настройка типов нажатий на тачпаде:

```
riverctl input _имя_устройства_ tap-button-map left-right-middle
```

Допустимые значения:

1. left-right-middle - нажатие на тачпад одним пальцем - левая кнопка мыши, двумя - правая, тремя - средняя,
2. left-middle-right - нажатие на тачпад одним пальцем - левая кнопка мыши, двумя - средняя, тремя - правая.

Чтобы задать способ прокрутки на тачпаде:

```
riverctl input _имя_устройства_ scroll-method none
```

Допустимые значения:

1. *none* - не использовать тачпад для прокрутки,
2. *two-finger* - прокрутка с помощью двух пальцев,
3. *edge* - пролистывание, используя край тачпада,
4. *button* - пролистывание с помощью движения указателя при нажатой кнопке.

Чтобы задать клавишу для пролистывания с помощью четвертого варианта:

```
riverctl input _имя_устройства_ scroll-button _код_события_
```

Режим работы клавиши прокрутки:

```
riverctl input _имя_устройства_ scroll-button-lock enabled
```

Допустимые значения:

1. *enabled* - клавишу не нужно удерживать. Первое нажатие войти в режим прокрутки, а второе нажатие - выйти,
2. *disabled* - клавишу нужно удерживать для прокрутки.

Чтобы указать устройству ввода работать только на определенном экране, используйте:

```
riverctl input _имя_устройства_ map-to-output HDMI-1
```

Допустимые значения:

1. *disabled* - отключить привязку устройства ввода к определенному экрану,
2. Все возможные варианты *output*, которые вы можете получить через тот же **wlr-randr** (DP-1, HDMI-1 и т.п.).

В river можно задать скорость обработки нажатий и задержку:

```
riverctl set-repeat rate delay
```

О параметрах:

1. *rate* - скорость повторной обработки нажатия (зажатия) клавиши в милисекундах (50 - по умолчанию),
2. *delay* - задержка повторной обработки нажатия (зажатия) клавиши в милисекундах (300 - по умолчанию).

1.14.2.10 Настройка уведомлений

Как и во многих других WM в river используются сторонние средства для отображения уведомлений. На wiki river рекомендуются следующие:

- mako
- salut
- fnott
- dunst
- wired
- SwayNotificationCenter

Для запуска вместе с river нужно в **init** указать команду:

```
riverctl spawn fnott    # fnott тут в качестве примера, замените на любой другой
```

1.14.2.11 Панель (status bar)

В качестве панели (status bar) на вики river рекомендуют использовать:

- waybar
- yambar
- levee
- creek
- i3bar-river
- zelbar
- dam
- sandbar

Для запуска панели достаточно указать команду в **init**:

```
riverctl spawn waybar     # вместо waybar можете использовать любую другую панель.
```

Предупреждение

Панели могут потребовать дополнительной настройки для отображения элементов river (tags, window, mode, layuot и т.п.).

Предупреждение

На 11.02.2025 г. панели не имеют модуля вывода текущей раскладки языка в river!

Как можно заметить для запуска приложения "в фоне" достаточно просто использовать *riverctl spawn _имя_программы_*, данная команда просто запустит программу - работает как через файл **init** так и из терминала.

1.14.2.12 Меню запуска приложений

Для запуска программ нужен так называемый *launcher* (лаунчер). На wiki river рекомендуют:

- fuzzel
- bemenu
- mew
- wofi
- LavaLauncher
- nwg-drawer
- tofi
- wmenu

Для использования достаточно определить комбинацию клавиш, что было описано вначале на примере *foot*.

1.14.2.13 Блокировка экрана

Для блокировки экрана на wiki river рекомендуют использовать:

- waylock
- swaylock

Заблокировать экран можно назначив запуск программы на "горячую" клавишу или используя swayidle:

```
riverctl spawn 'swayidle -w timeout 300 "swaylock --show-failed-attempts -e"'
```

Немного пояснения:

- *-w* - говорит о необходимости ожидания события,
- *timeout* - событие, которое ожидает swayidle - в данном случае ожидание бездействия в 300 секунд.

Через **swayidle** можно выполнять разные команды. Например выключать и включать мониторы. В мануале **swayidle** есть подобное события:

```
swayidle -w \
    timeout 300 'swaylock -f -c 000000' \
    timeout 600 'swaymsg "output * dpms off"' \
          resume 'swaymsg "output * dpms on"' \
    before-sleep 'swaylock -f -c 000000'
```

Предупреждение

Будьте внимательны при настройки отключения мониторов, поскольку после пробуждения могут часть настроек river "слететь", особенно в случае с несколькими мониторами, а в случае с видеокартами nvidia возможны проблемы с пробуждением!

Алфавитный указатель

Не-алфавитный

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z