1. Аннотация

Целью данной работы является создание среды исполнения для гетерогенной вычислительной системы.

Для этого необходимо изучить гетерогенные вычислительные системы, предъявить требования к среде исполнения процессов в ней. Далее выбрать подходящую для нашей задачи операционную систему. Операционную систему нужно портировать на гетерогенную вычислительную систему и отладить на симуляторе.

Полученные результаты. Изучена специфика гетерогенных вычислительных систем. В качестве основы для среды исполнения Выбрана операционная система NetBSD. Значительная часть модулей требующих портирования адаптированы. Успешно пройдена большая часть запуска операционной системы. Однако остались и открытые вопросы и задачи.

2. Оглавление

3. Введение

В современном мире, высокопроизводительные вычисления занимают значительную нишу рынка IT технологий. Крупные компании производители электроники конкурируют на этом рынке разрабатывая новые технологические решения. Не обошла этот рынок стороной и компания Intel.

Был предложен новый подход к увеличению производительности «графической» карты, а именно гетерогенная вычислительная архитектура, эффективно использующая разные уровни параллелизма.

Такой сложной вычислительной системе нужна развитая среда исполнения, вот ее то в этой работе мы и будем разрабатывать.

Сформулируем требования к нашей работе:

1. Изучить архитектуру и специфику гетерогенной вычислительной системы Intel.
2. Выбрать операционную систему для построения на ее базе среды исполнения.
3. Изучить структуру выбранной операционной системы.
4. Адаптировать операционную систему для наших задач.
5. Запустить и отладить полученную операционную систему на симуляторе вычислительной системы.

4. Гетерогенные архитектуры и их особенности

4.1. Предпосылки создания гетерогенных вычислительных систем. Разноуровневый параллелизм.

В целом параллелизм в исполнении программ можно поделить на два уровня

1. Параллелизм уровня алгоритма. Это параллелизм, который позволяет сам алгоритм вычислений, например, в задаче умножения матриц, перемножение все столбцов со строками можно выполнять параллельно так как отсутствуют зависимости по данным. При этом на вичисление каждого элемента задачи требуется достаточно большое количество инструкций процессора.
2. Параллелизм уровня инструкций. Этот параллелизм который возникает при исполнении программы, когда некоторые элементарные операции не имеют зависимостей по данным и могут быть выполненны процессором одновременно, хотя в алгоритме написанном на языке высокого уровня происходят последовательно. Такой параллелизм может обнаруживать и оптимизировать компилятор, однако современные суперскалярные процессора делают это автоматически в некотором окне инструкций.

Предпосылкой к созданию гетерогенной вычислительной системы является узкое место в производительности, «бутылочное горлышко», возникшее в результате развития архитектуры, повсеместно использующейся при построении персональных, а также серверных компьютеров. В ней центральный процессор (Host CPU) обменивается данными с вычислительной картой (графической картой, GPU), представляющей из себя по сути массив процессоров и память, при помощи шины PCI/PCI-e. С ростом количества и скорости графических процессоров в массиве, а также увеличения производительности центрального процессора, постепенно шина PCI стала узким местом, вызывающим значительные задержки в обработке данных.

Так, например, параллелизуемые части вычислительных задач (векторные вычисления) происходят в массиве графических процессоров, процессора которого имеются в большом количестве, однако имеют достаточно простое устройство, не позволяющее осуществлять эффективный параллелизм на уровне инструкций. На них рационально выполнять задачи, имеющие параллелизм на уровне алгоритма. Скалярная часть вычислений

(например, анализ полученных векторных данных) происходит на суперскалярном CPU, что приводит к необходимости передать большой полученный объем данных из графического ускорителя в основную память компьютера.

Если обратить внимание на задержки, возникающие при передаче данных по шине PCI то станет понятно, что скорость передачи по ней значительно меньше пропускной способности шины памяти, что означает что при передаче данных из памяти графической карты в память компьютера будут иметь место значительные накладные расходы.

Так, например, пропускная способность шины памяти в графических картах NVidia последнего поколения насчитывает более 1Терабайта в секунду при задержках на доступ в память порядка 100 наносекунд, которые, к тому-же, в значительной степени компенсируются многоуровневыми кешами графической карты.

В то же время скорость передачи данных по шине PCI насчитывает порядка 100 Гигабайт в секунду и задержками порядка 150 наносекунд которые невозможно компенсировать кешированием.

На рисунке ниже схематично изображена структура современных графических карт.

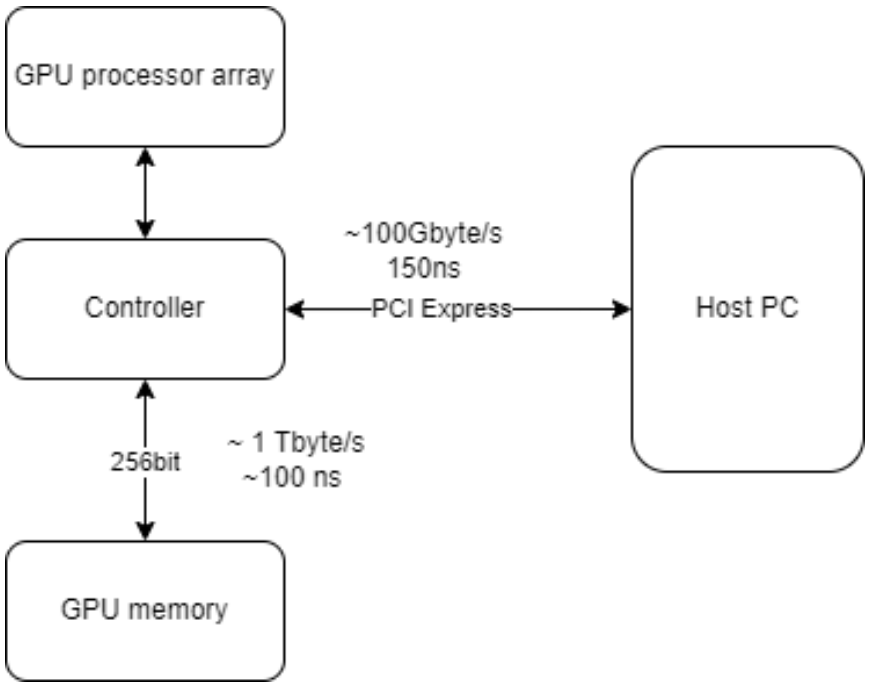

Рис 1. Структура современной вычислительной карты.

Решением данной проблемы может стать установка на графическую карту дополнительного скалярного процессора, позволяющего осуществлять эффективный параллелизм на уровне инструкций и имеющего прямой доступ в память и когерентностный кеш с графическим процессором карты, с целью произвести на нем все скалярные вычисления. В таком случае передача данных в основную память компьютера будет происходит только по завершению всей вычислительной задачи, что значительно увеличит скорость вычислений и снизит нагрузку на канал передачи данных.

Такая структура изображена на рисунке ниже.

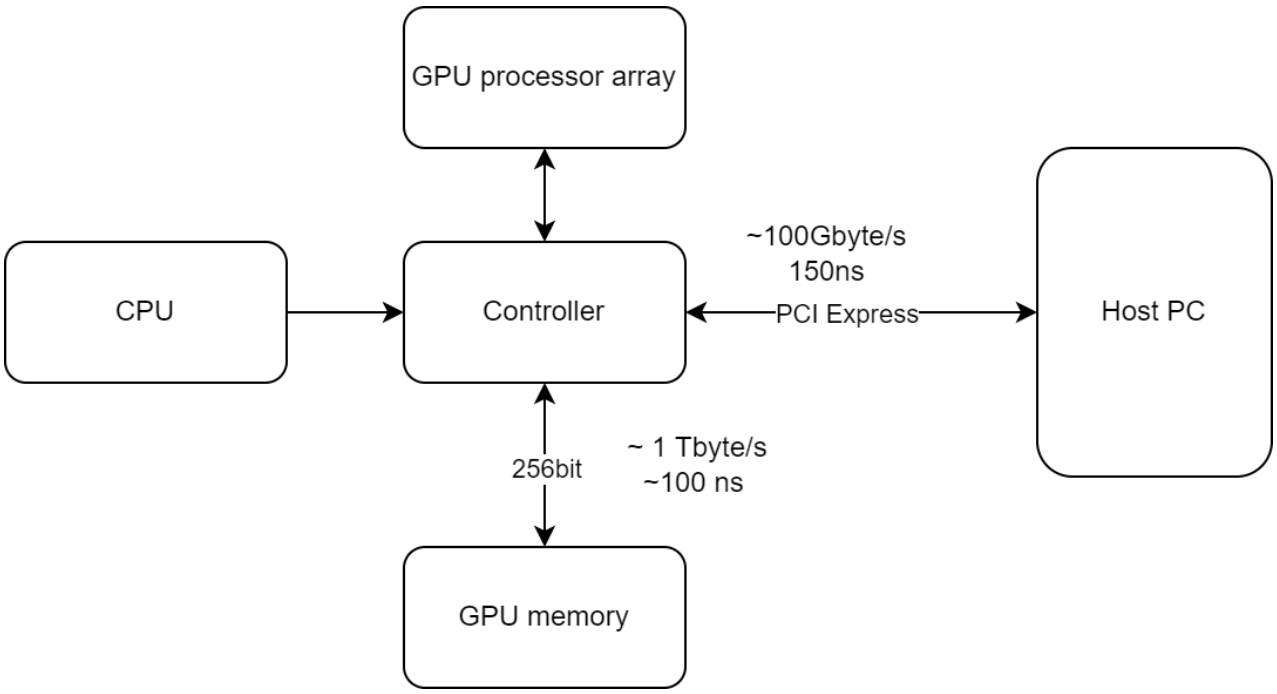

Рис. 2. Упрощенная структурная схема гетерогенной вычислительной системы.

4.2. Требования к системной части архитектуры

Задача установить на графическую карту скалярный процессор сталкивается с проблемой – скорость с которой растут количество памяти и процессоров графической карты а также их производительность, значительно превышают скорость с которой наращивается производительность «больших» процессоров на базе архитектуры x86, тем временем производительности скалярного и графического процессоров должны находиться в

некотором постоянном соотношении которого требуют исполняемые вычисления, дабы не создавался перевес в одну из сторон и ресурсы обоих процессоров использовались оптимально. Отсюда вывод – нужна новая архитектура, имеющая возможность легко быть расширенной, например, за счет увеличения количества процессорных ядер.

Отсюда же вытекает требование и к среде исполнения на таком скалярном процессоре – должна поддерживаться возможность многопоточного исполнения, а значит должна быть и операционная система, управляющая исполнением и переключением потоков исполнения.

4.3. Особенности разработки и исполнения приложений в гетерогенной среде

Особенность приложения для гетерогенной вычислительной системы состоит в том, что образ такой программы содержит в себе исполняемый код сразу и для графического, процессора и для скалярного. Следовательно, код программы на языке высокого уровня должен содержать информацию о том, на каком процессоре исполнять данный код.

5. Вычислительная архитектура с разноуровневым параллелизмом

5.1. Структура параллельной гетерогенной системы.

Гетерогенная вычислительная система состоит из:

- Массива графических процессоров, выполняющих задачи в которых возможен высокий параллелизм исполнения.
- Скалярного сопроцессора, выполняющего задачи в которых параллелизм на уровне алгоритма невозможен, однако способный к параллелизму на уровне инструкций, т.н. CPU.
- Общей памяти к которой имеют доступ и массив графических процессоров и скалярный сопроцессор.
- Контроллера задач, который осуществляет обмен данными с хостовым компьютером, получает новые задачи для исполнения, а также передает данные выполненных задач. Он также отвечает за отладку приложений выполняющихся на вычислительной системе.

Общая схема такой вычислительной системы представлена на рисунке ниже.

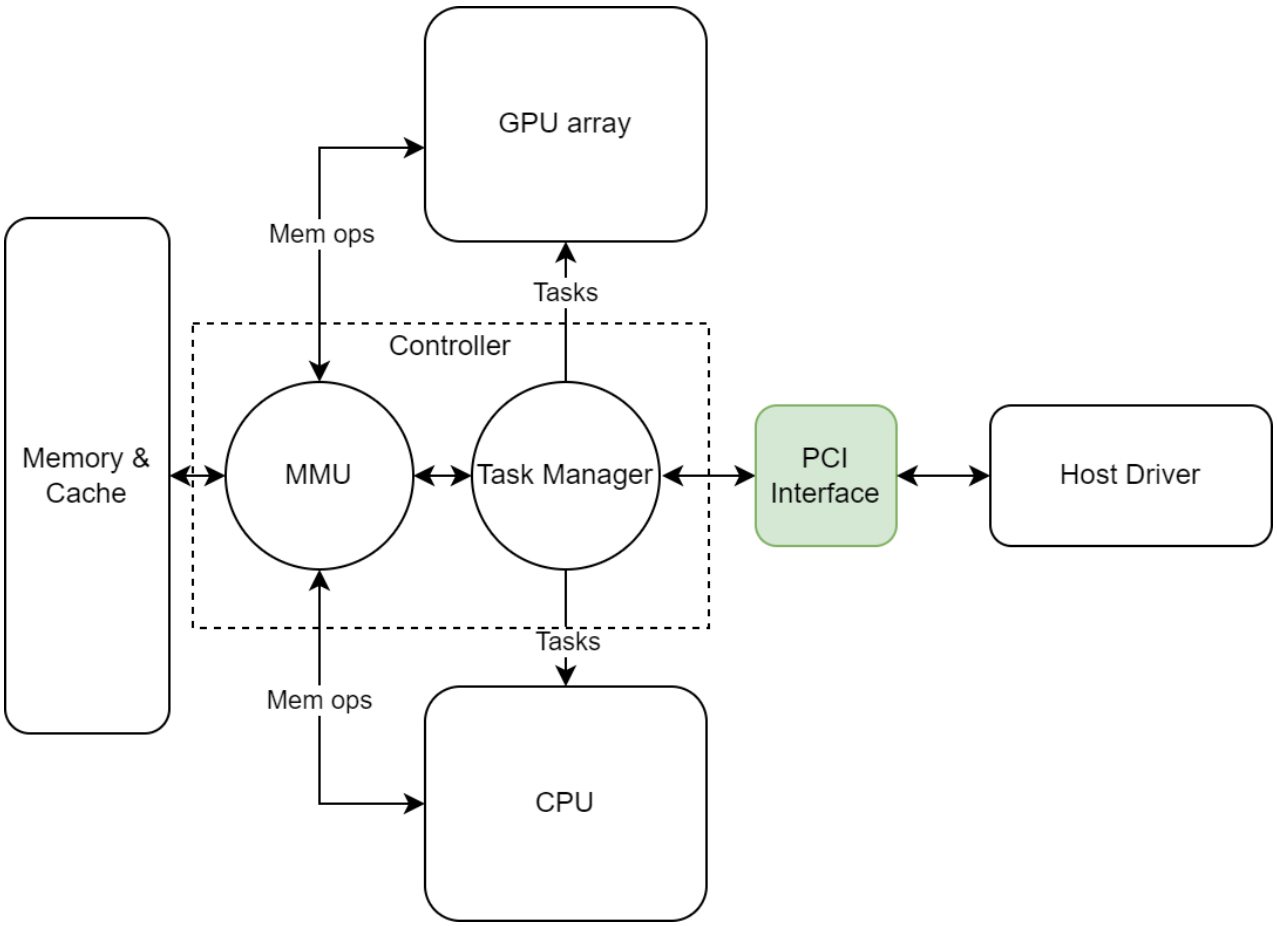

Рис. 3. Устройство гетерогенной вычислительной системы.

5.2. Операционная система и ее задачи

Операционная система в такой гетерогенной вычислительной среде необходима по той причине, что необходимо снять функции планирования исполнения задач на графической карте с хостовой системы чтобы не занимать канал передачи данных и не терять время на ожидание данных и команд. К тому же желательно иметь возможность выполнения нескольких задач на вычислительной карте одновременно, чтобы полностью утилизировать ее производительность. Это означат, что управляющая система должна иметь понятие потоков исполнения внутри одной задачи, а также процессов предназначенных для одновременного выполнения нескольких задач.

Помимо этого, в перспективе, такое устройство может стать самостоятельным устройством работающим без хостовой системы, а следовательно должен быть предусмотрен функционал работы с файловыми системами и различными устройствами ввода-вывода.

Все перечисленные выше требования выполняются в операционных системах общего назначения таких как Linux и системы семейства BSD.

5.3. Критерии выбора операционной системы.

Основными критериями выбора операционной системы для нашей задачи таким образом являются:

- Легкая портируемость. Что означает хорошую модульность кода, а также возможность так называемой кросс-компиляции.
- Лицензия, позволяющая использовать измененный исходный код операционной системы в коммерческих целях.
- Полнота функционала ОС, позволяющая расширение функционала устройства.

Для адаптации к гетерогенной системе была выбрана операционная система NetBSD т.к. она удовлетворяет всем перечисленным пунктам.

6. Операционная система NetBSD

6.1. Структура и преимущества NetBSD

Одним из главных преимуществ операционной системы NetBSD является ее модульная структура. Исходный код операционной системы строго делится на две части – машинно-зависимую и машинно-независимую. При этом очевидно пропадает необходимость модифицировать машинно-независимый код при создании адаптации (порта).

Все операционные системы типа UNIX в целом имеют следующую иерархическую структуру:

- На первом слое идет аппаратура. Это собственно и есть та самая «платформа», на которую операционная система портируется.
- Далее идет ядро операционной системы. В него входят все системные функции такие как файловые системы, управление памятью, драйверы устройств и прочее.
- В третьем слое операционной системы находятся системные утилиты и службы, это так называемый userland.
- Пользовательские программы. Эта часть уже определяется пользователям системы.

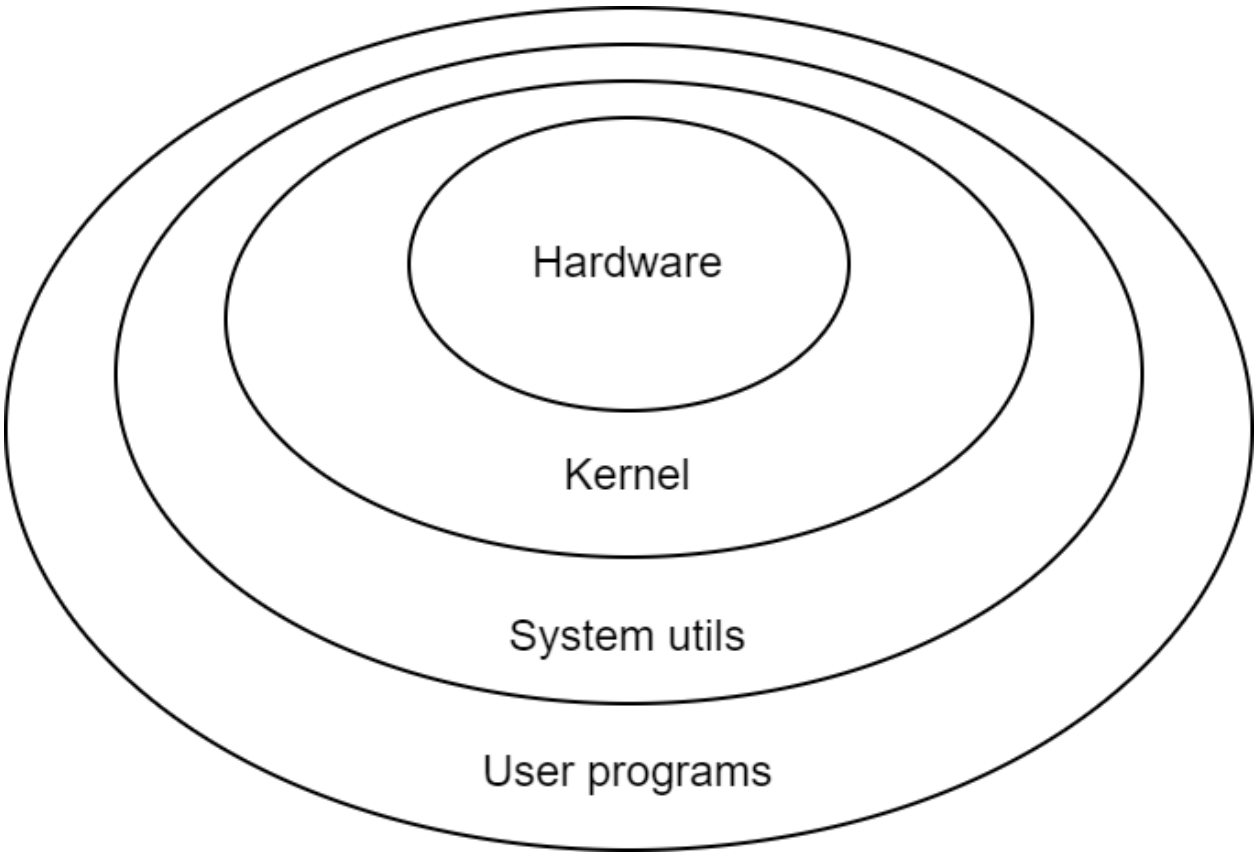

Рис 4. Иерархия в современных ОС

Рассмотрим теперь более подробно модульную структуру unix-подобной операционной системы.

На рисунке ниже она представлена в виде трех слоев:

- Уровень приложений. На этом уровне исполняются системные утилиты и пользовательские программы. Процессы на этом уровне исполняются в виртуальном адресном пространстве и изолированы друг от друга. Это делается с целью безопасности и универсальности системы. Системные утилиты запускаются во время запуска системы, но работают тем не менее на уровне приложений. В большинстве процессоров есть разделение инструкций на так называемые «кольца» исполнения. В кольце приложений, например, недоступны некоторые системные регистры, как например регистры управления виртуальной памятью, также недоступен ввод-вывод устройств. Таким образом все эти функции исполняются в другом кольце исполнения – так называемом

«системном». Смена кольца исполнения происходит как правило при обработке прерываний, а также при исполнении инструкции “syscall” эта инструкция прерывает исполнение приложения и отдает управление ядру для выполнения запрашиваемой операции (например, операции с файлом). При помощи системных вызовов приложения пользуются всеми сервисами предоставляемыми операционной системой.

- Уровень ядра операционной системы. Здесь выполняются все функции операционной системы, которые требуют доступа к аппаратуре и управления памятью. В целом ядро операционной системы состоит из следующих функций:
  - Интерфейс системных вызовов. Это та часть операционной системы, которая отвечает на syscall. Здесь происходит распределение задач для остальных модулей системы.
  - Файловая система. Это виртуальная прослойка над устройствами ввода вывода. Она может отображать как реальные устройства хранения данных, такие как диски, карты памяти и им подобные блочные устройства, так и любые другие устройства ввода-вывода, например, USB, COM порты и.т.д. Необходим этот модуль для абстракции Различных файловых систем, таких как FAT, NTFS и.т.д, в также устройств в единый вид. Также эта система имеет механизмы буферизации и контроля за ресурсами ввода-вывода.
  - Опирается виртуальная файловая на драйверы устройств. Драйверы устройств - это модули ядра каждый из которых имеет унифицированный интерфейс управления, получения и отправки данных, а также инициализации и обработки прерываний.
  - Модуль управления памятью отвечает за распределение физической памяти для процессов, а также учетом потребления памяти, буферизации памяти в файле подкачки, созданием таблиц страниц и.т.п.
  - Для получения доступа к непосредственно аппаратуре управления памятью имеется модуль управления MMU. Он оборачивает универсальные операции управления памятью в реальные операции с регистрами MMU.
  - Для реализации многозадачности в ядре операционной системы имеется система управления процессами. В этой системе имеется планировщик задач, который распределяет задачи по ядрам центрального процессора, управляет их остановкой, снятием и.т.п.
  - Доступ к аппаратуре модуля управления процессами осуществляется при помощи интерфейса системной архитектуры центрального процессора. Эта прослойка

осуществляет необходимые манипуляции с системными регистрами, а также создание сохранение и восстановление контекста процессов и потоков исполнения.

- Управление сетью. Этот модуль создает системную абстракцию сети, так называемые сокеты (socket). Он содержит интерфейс сокетов, а также реализацию стандартных сетевых протоколов, таких как TCP и UDP. Также он выполняет функции маршрутизации и коммутации и ответа на различные запросы протоколов типа ARP и ICMP.
- Для ввода-вывода в реальную сеть требуются драйверы сетевых адаптеров таких как WIFI и Ethernet. А также реализация протоколов специфичных для различных стандартов передачи данных.

- Уровень аппаратуры. Это непосредственно «железо» компьютера. Здесь находятся диски, сетевые устройства, и прочие устройства ввода-вывода. Особенно стоит остановиться на устройствах памяти. В современных компьютерах процессы выполняются в так называемой «виртуальной» памяти. Для организации такой виртуальности необходим аппаратный блок управления памятью, который отображает страницы виртуальной памяти в страницы физической памяти (MMU).

Общую схему операционной системы изображена на рисунке ниже.

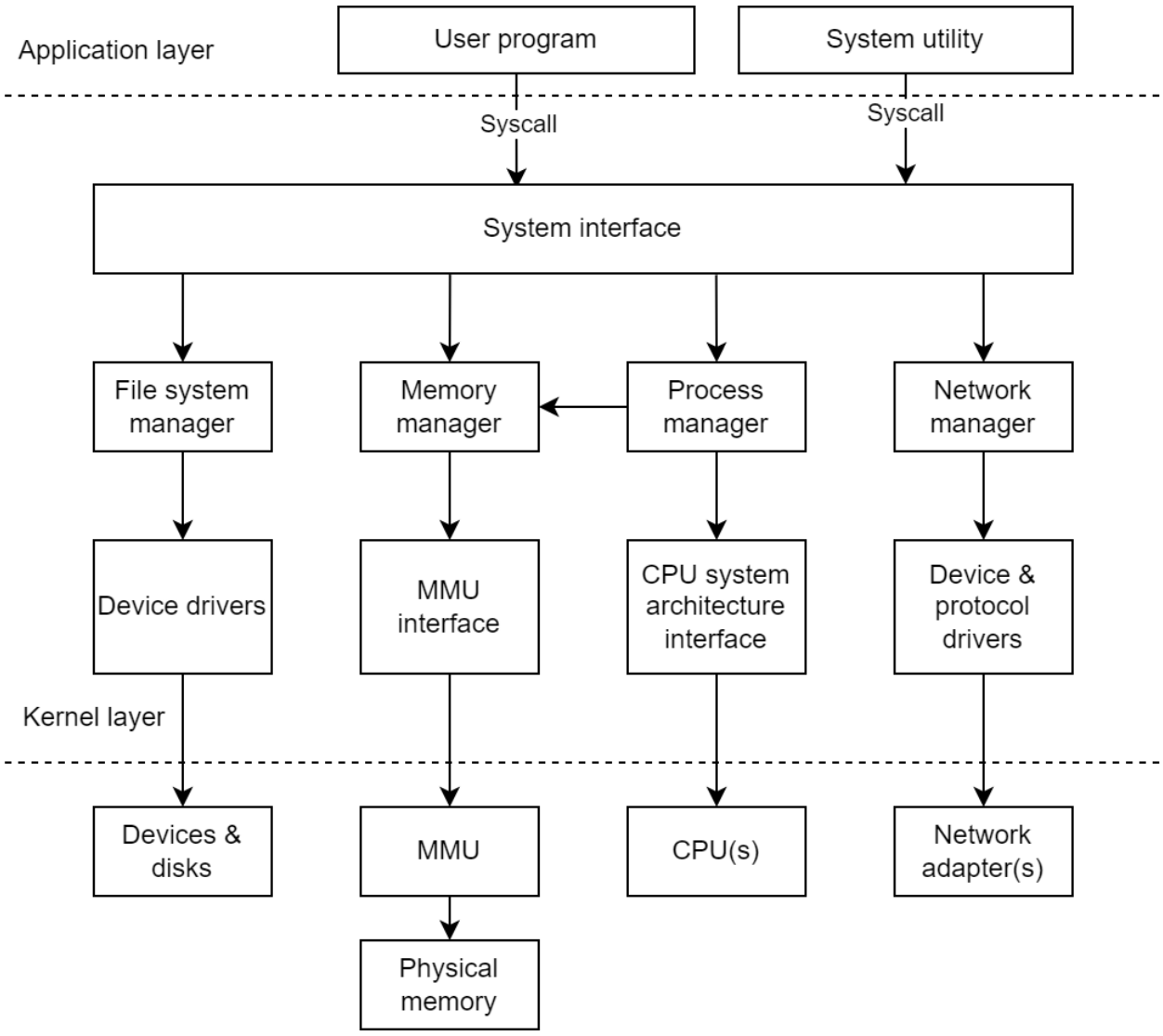

Рис 5. Модульное строение современной ОС

6.2. Особенности адаптации NetBSD для задач новой архитектуры

В этом разделе мы рассмотрим особенности адаптации операционной системы к нашей гетерогенной вычислительной аппаратуре. Основной особенностью, а точнее основным отличием гетерогенной вычислительной карты от обычного компьютера (ПК или встраиваемой системы) является то, что наша карточка является ведомым устройством, то есть не запускается сама и скалярный сопроцессор не имеет возможности управлять всеми процессами, происходящими в системах карты. Так, например, сопроцессор не имеет

возможности управлять MMU который находится в GUC. Это чрезвычайно осложняет процесс управления памятью в операционной системе, работающей на CPU.

Также нет необходимости некоторых функциях. Так, например, на первом этапе создания гетерогенной вычислительной системы, когда она запускается с хостового ПК, нет необходимости в поддержании полноценных файловых систем на различных носителях с ПЗУ. Ввод-вывод будет осуществляться через интерфейс хостового драйвера.

6.3. Возможности NetBSD для построения унифицированной архитектуры

Операционная система NetBSD имеет ряд полезных особенностей для адаптации ее к гетерогенной вычислительной архитектуры.

- Машинно-независимые драйверы устройств. Драйверы в операционной системе NetBSD могут быть написаны так, что не задействуют функции из платформенно-зависимой части ядра непосредственно, это следствие строгой модульности кода NetBSD. Это облегчает портирование, т.к. для запуска NetBSD на реальной аппаратуре нужны драйверы устройств ввода-вывода, а наличие их в машинно-независимой части ядра ускоряет разработку порта и его отладку т.к. нет необходимости разрабатывать и отлаживать драйверы вместе с отлаживаемой системой. Это особенно важно если код ядра еще не отлажен до конца т.к. ускоряет поиск ошибок. Эта особенность также значительно облегчает разработку NetBSD для унифицированной архитектуры т.к. при смене, например, только системы команд процессора, адаптировать систему под новую платформу будет очень просто.
- Поддержка любого количества процессоров. NetBSD имеет поддержку многопроцессорности в машинно-независимой части ядра, и малого количества функций для поддержки мультипроцессорности в машиннозависимой. Более того не накладывается ограничений на количество процессоров, для запуска всех процессоров операционная система определяет их количество при старте, а далее пользуется всеми. Это означает что для скалярного сопроцессора вычислительной карты можно создать ОС, которая будет самостоятельно адаптироваться к растущей производительности карты, то есть система будет легко масштабируемой.
- Внедрение дополнительных возможностей. При будущем развитии графической карты будут необходимы остальные функции операционной системы общего назначения такие как сеть и файловая система, а в NetBSD они уже реализованы, что также ускорит разработку.

В целом NetBSD является одним из лучших вариантов операционной системы для гетерогенной вычислительной системы.

7. Модульная система NetBSD

Операционная система NetBSD имеет модульное строение, в данной главе мы рассмотримего подробно, особенное внимание обратим на те модули, которые пришлось реализовать для адаптации NetBSD к гетерогенной системе.

Исходный код операционной системы разделен на машинно-зависимые и машинно-независимые файлы или файлы портов. Все порты хранятся в специальной директории в дереве кода NetBSD. Модульная система NetBSD проиллюстрирована на рисунке 4.1.

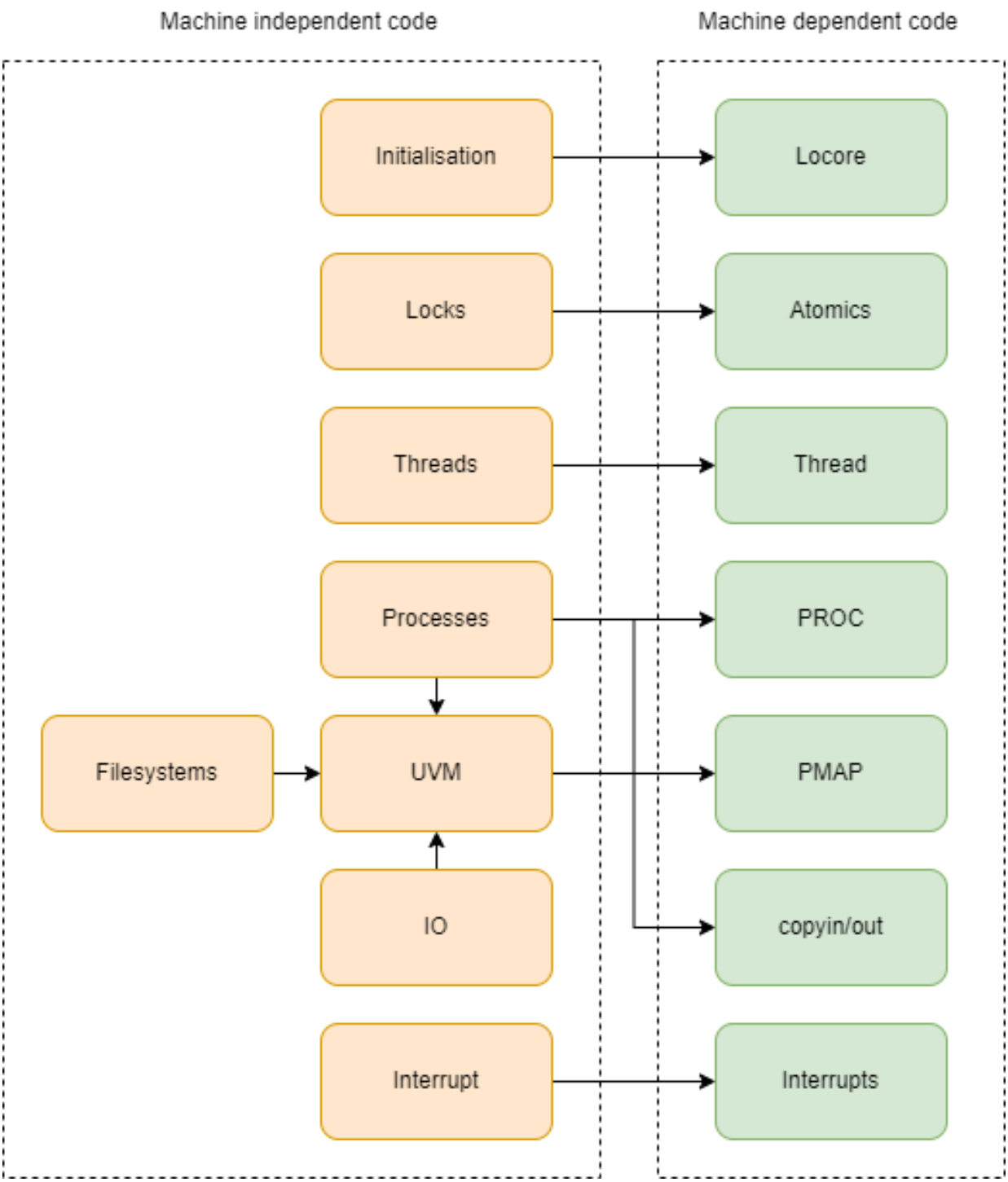

Рис 6. Модульная система NetBSD.

Далее рассмотрим модули более подробно.

7.1. Модуль низкоуровневой поддержки ядра (Locore)

Этот модуль самый низкоуровневый функционал необходимый для работы остальных функций ядра. Он же отвечает за первичную инициализацию ядра.

Например процесс инициализации выглядит следующим образом:

3. Настройка ведущего процессора для работы операционной системы (установка таблицы прерываний, включение нужного режима исполнения)
4. Подготовка среды исполнения (установка указателя стека и.т.п)
5. Создание пространства памяти ядра (создание таблицы страниц).
6. Переопределение вектора прерывания PageFault.
7. Включение MMU.
8. Происходит прерывание по PageFault.
9. Вектор переопределяется на штатный адрес обработчика.
10. Управление передается в main ядра.

Необходимой функцией модуля locore является определение рабочих диапазонов физической памяти доступной для ядра.

7.2. Инициализация ядра. (Init).

Этот модуль уже машинно-независимый. В него передается управления из Locore, здесь находится main ядра. Самая главная функция этого модуля в том чтобы в строго нужном порядке инициализировать изапустить все остальные с учетом всех зависимостей.

7.3. Модуль Блокировок (Locks).

Этот модуль необходим для синхронизации процессов и потоков в системе с параллелизмом. Он аппаратно независим, и опирается на модуль о котором речь пойдет далее. В нем реализуются основные примитивы синхронизации:

- Мьютексы (mutex). Это примитив синхронизации который позволяет войти в некоторую область кода одновременно только одному процессу.
- Барьеры доступа в память – система необходимая если нужно синхронизировать состояние памяти для двух процессоров.
- Условные переменные. Это механизм для реакции процессов на события происходящие в системе.

7.4. Модуль атомарных операций (atomics).

Этот модуль отвечает за атомарные операции с переменными в памяти.

Атомарная операция с переменной это такая операция которая не может быть прервана переключением на исполнение другого процесса. Это важно потому что операция с переменной как правило состоит из трех действий

1. Чтение. Значение считывается из памяти в регистр процессора.
2. Модификация. Значение модифицируется в регистре.
3. Запись. Значение записывается из регистра в память.

Если во время модификации значения другой процесс считает значение переменной из памяти и тоже изменит его, то изменения сделанные одним процессом не внесут изменений в значение переменной после записи в память. Это приведет к ошибке в программе.

Поскольку нет необходимости реализовывать вообще любую атомарную операцию, а можно обойтись лишь реализацией некоторых, а их уже использовать для построения примитивов синхронизации, то требуется реализация семи классов атомарных операций:

1. Атомарное сложение (atomic_add). Атомарно прибавляет к переменной в памяти число.

2. Атомарное И (atomic_and). Атомарно производит битовую операцию И между переменной в памяти и переданным параметром.

3. Атомарная сравнение с заменой (atomic compare-and-swap). Производит смену значений переменной из памяти и параметра в случае если первое меньше второго.

4. Атомарный декремент (atomic_dec). Атомарно уменьшает переменную на единицу.

5. Атомарный инкремент (atomic_inc). Атомарно увеличивает переменную на единицу.

6. Атомарное ИЛИ (atomic_or). Атомарно производит битовую операцию ИЛИ между переменной в памяти и переданным параметром.

7. Атомарная замена (atomic_swap). Заменяет значение переменной в памяти переданным параметром а старое значение возвращает.

Данные атомарные операции были реализованы для адаптации операционной системы к архитектуре скалярного сопроцессора. В данной архитектуре атомарность достигается тем, что можно проверить, не было ли доступа по данному адресу памяти с момента чтения. Таким образом если во время исполнения атомарной операции произошло изменение значение в памяти, то нужно повторить чтение значения и повторить выполнение операции.

Рассмотрим реализацию атомарного сложения, алгоритм выполнения такой атомарной операции представлен на картинке ниже.

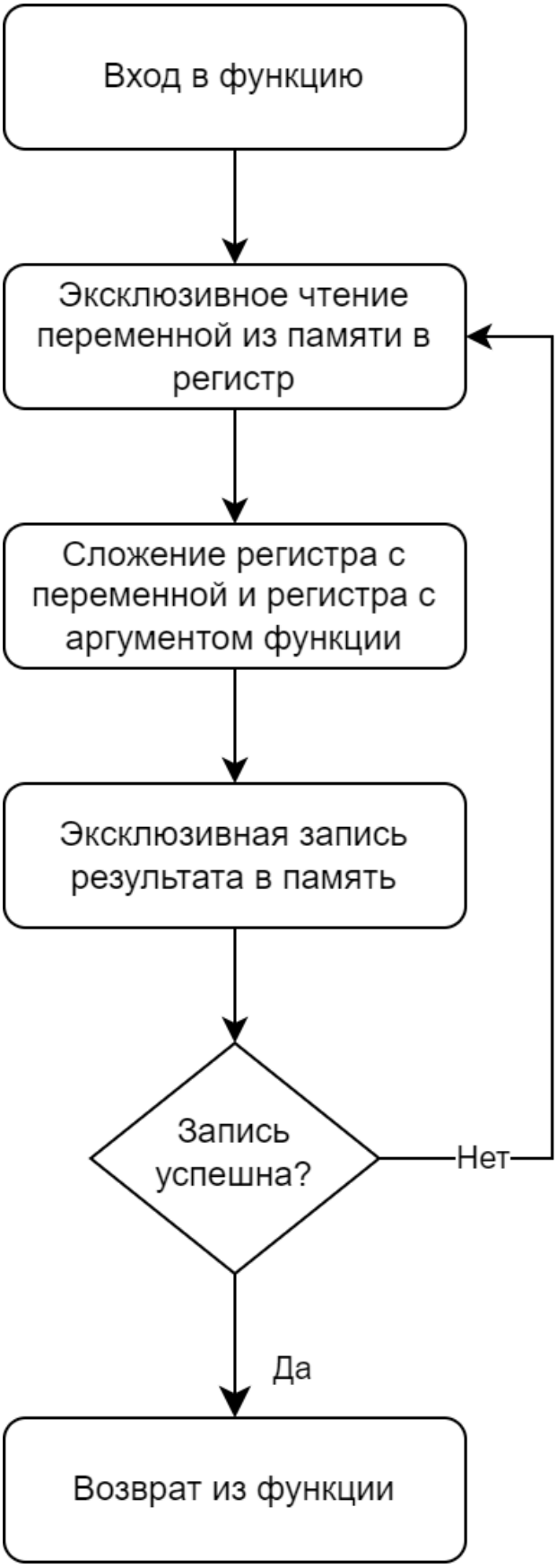

Рис 7. Реализация атомарного сложения.

Эксклюзивный доступ в память осуществляется при помощи специального аппаратного блока в процессоре нашей архитектуры – монитора доступов в память. Реализован он в виде специальных инструкций загрузки (ldrex) и сохранения (strex).

7.5. Потоки (Threads).

Модуль управления потоками отвечает за создание и управление потоками исполнения. Модуль имеет части как в машинно-зависимой так и в машинно-независимой части кода.

Рассмотрим функции которые необходимо реализовать в машинно-зависимой части для работы данного модуля ОС:

- Создание потока. Необходимо выделить место для стека потока, создать контекст потока а также правильным образом этот контекст наполнить, а конкретно – установить указатель стека на созданный для потока стек.
- Сохранение контекста потока. Эта функция сохраняет значение регистров процессора во время переключения потока в структуру описывающую контекст.
- Восстановление контекста потока. Извлекает значение регистров процессора из структуры хранящей контекст и записывает их в регистры. После чего отдает управление потоку.
- Также необходимо описать структуру хранящую контекст процесса. Она хранит значения регистров ядра.

7.6. Процессы (proc).

Модуль управления процессами. На самом деле включает в себя множество более мелких частей а также опирается на другие модули. Так что этот модуль скорее логический чем

программный. Однако в аппаратно-зависимой части системы требуется реализация поддержкки управления процессами.

- Машинно-зависимая часть контекста процесса. Это структура которая описывает машинно-зависимую часть контекста процесса, например состояние каких-нибудь регистров.
- Операция fork в модуле управления памятью копирует адресное пространство процесса.
- Операции с машинно-зависимой частью контекста процесса.

7.7. Универсальный модуль управления памятью (UVM).

Это абстрагированная от аппаратуры система управления виртуальной памятью, очень большой и важный модуль операционной системы, и на него завязаны многие остальные модули. Так например система ввода-вывода использует виртуальную память для буферизации. Системе управления процессами необходимы манипуляции с памятью для управления адресным пространством процессов. Даже планировщик в операционной системе NetBSD находится в системе управления памятью. О портировании этого модуля речь пойдет в следующей главе.

7.8. Машинно-зависимое управление виртуальной памятью (PMAP).

Этот модуль является слоем абстрагирующим UVM от аппаратуры. Он предназначен для простейших манипуляций с таблицами страниц и операциями с MMU. О нем также пойдет речь в следующей главе.

7.9. Модуль копирования между адресными пространствами ядра и приложения (copy).

С целью защиты ядра и приложений при исполнении приложения оно не может получить доступ в адресное пространство ядра, а ядро не имеет доступа в область памяти приложения. Для того чтобы передавать данные из ядра и в ядро такие как например буферы для вывода при обработке системного вызова записи в файл. В разных платформах эти функции имеют различную реализацию, так как защита имеет различную аппаратную реализацию. В нашей

системе для доступа из ядра в адресное пространство процесса предусмотрены специальные инструкции чтения и записи данных в регистры.

По принципу действия функции копирования аналогичны функции memcpy стандартной библиотеки языка C однако реализовывать их придется на ассемблере т.к. нужно использовать специализированные инструкции. Очень важна оптимизация этого копирования, т.к. от этого зависит производительность ввода-вывода. Для оптимизации будем использовать копирование блоками максимальной доступной ширины и выравнивать адреса по ширине доступа.

7.10. Файловая система (VFS).

Операционная система NetBSD имеет унифицированную виртуальную файловую систему. Она абстрагирует файловые операции от реализаций реальных файловых систем, а также от устройств. В этом слое также происходит учет статистики использования ресурсов, а также управление процессами, связанное с вводом-выводом.

При создании нашей адаптации поддержка файловых систем была не нужна, поэтому реализации реальных файловых систем были отключены, а VFS впоследствии значительно изменена в связи с модификациями в системе управления памятью.

7.11. Система ввода-вывода.

Ввод-вывод в операционной системе NetBSD осуществляется модулем UIO это абстракция позволяет создавать буферы и очереди буферов с данными для ввода и вывода, а также управлять процессами в зависимости от стратегии ввода-вывода.

К этой системе можно отнести также систему драйверов.

Драйвер это совокупность стандартного набора функций управляющих устройством, реальных или виртуальным, структуры содержащей указатели на функции интерфейса устройства при его автоконфигурации при запуске системы, а также структуры содержащей указатели на стандартные функции ввода-вывода. Такая структура драйверов позволяет сделать их универсальными, а также проводить автоматическое монтирование устройств при их подключении и при запуске системы.

Для описания иерархии устройств платформы, операционной системе передается из загрузчика специальная структура, так называемое дерево устройств (device tree, dtb).

7.12. Система прерываний.

Система прерываний, как и остальные состоит из машинно-зависимой и машинно-независимой частей.

В машинно-зависимой части системы находится таблица прерываний в которую передается управление, когда происходит прерывание или исключение.

В машинно-независимой части системы находится система обработки прерываний. Обработчики регистрируются в системе, например, драйверами устройств, и состоят из двух частей

- «Верхняя половина» это часть обработчика которая предназначается для работы с данными которые нужно обработать быстро. Вызывается сразу после возникновения прерывания.
- «Нижняя половина» прерывания ставится в очередь и вызывается уже специальным потоком, обрабатывающим прерывания позднее. В ней происходит уже менее срочная обработка данных.

В системе также имеется механизм иерархии прерываний, позволяющий в критические моменты работы отключить малозначительные прерывания.

Для передачи сигналов между модулями системы, например из одного драйвера в другой имеется система «программных» прерываний. То есть прерываний которые вызываются не аппаратурой а программно.

Для портирования операционной системы на нашу гетерогенную архитектуру была создана таблица прерываний перехватывающая прерывания от аппаратуры процессора и передающая управление системному обработчику прерываний.

На рисунке ниже изображен процесс обработки прерывания в системе NetBSD.

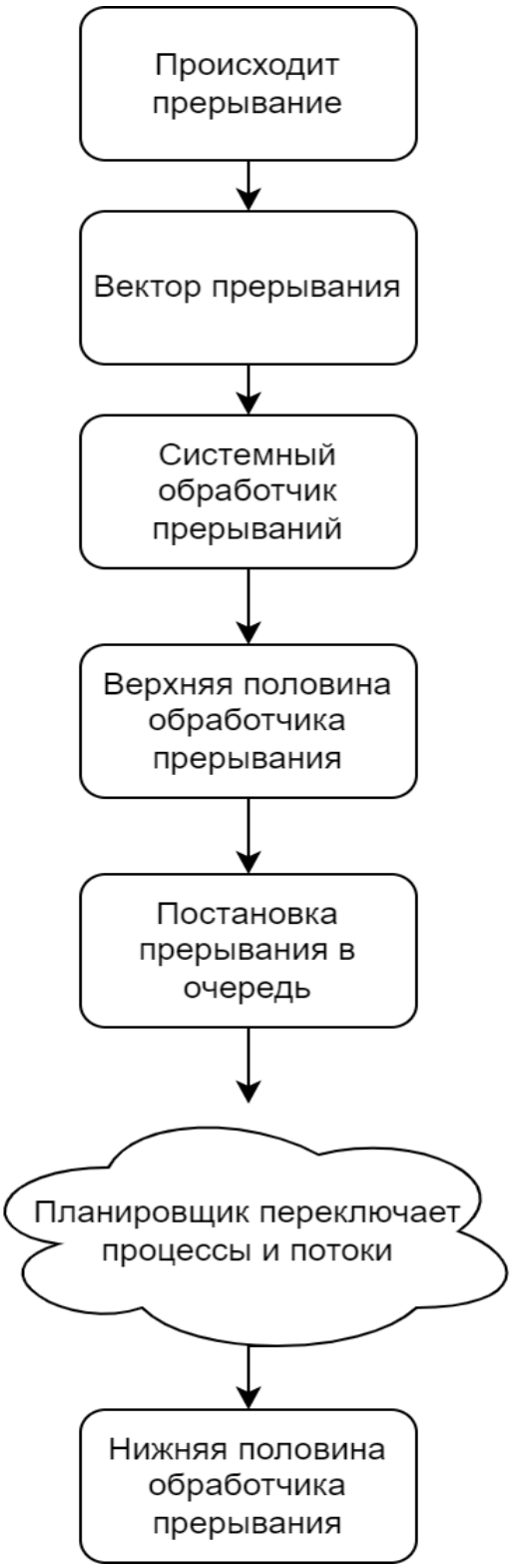

Рис. 8. Процес обработки прерывания в системе NetBSD.

7.13. Система сборки NetBSD.

Одним из важных модулей NetBSD, хоть и не будучи непосредственно исходным кодом, является система сборки.

Система сборки NetBSD и заточена под легкое портирование и кросс-компиляцию.

Система сборки выполняет четыре основных функции

1. Сборка инструментов необходимых для кросс-компиляции.
2. Сборка ядра операционной системы.
3. Сборка системного окружения (userland).
4. Генерация загружаемых образов, образов дисков и.т.п.

Подробнее рассмотрим систему сборки ядра. Она, как и исходный код строго поделена на машинно-зависимую и машинно-независимую части.

Происходит сборка в три этапа

1. Сборочный скрипт определяет параметры сборки такие как целевая архитектура и то, какую часть ситемы требуется собрать.
2. Автоконфигурация. Части makefilов генерируются специальной утилитой по конфигурационным файлам.
3. Сборка при помощи утилиты make по Makefilам.

Весь этот процесс изображен на следующей иллюстрации:

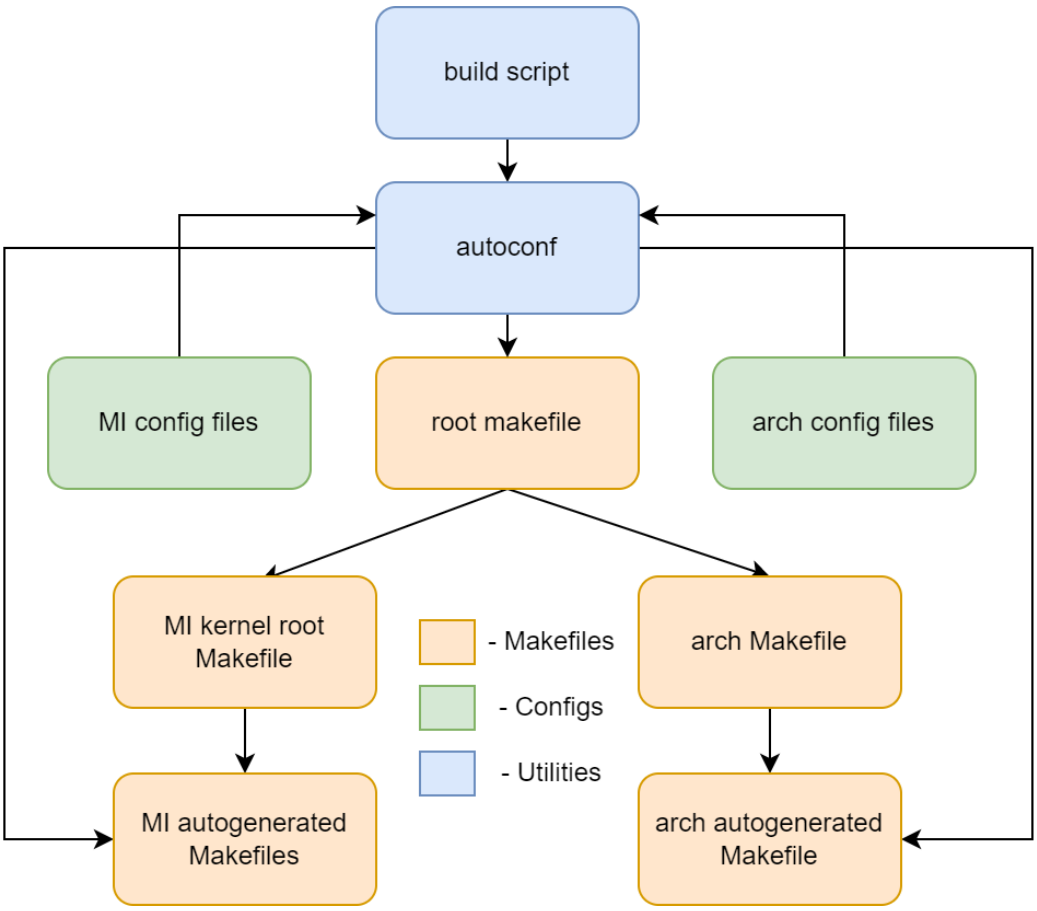

Рис 9. Система сборки ядра NetBSD.

8. Подсистема памяти

В этой главе речь пойдет о системе управления памятью в гетерогенной системе.

Центральным элементом любой операционной системы является виртуальная память. Существует стандартный подход к организации виртуальной памяти, когда центральный процессор имеет в себе блок управления памятью или MMU.

Выглядит схематически это следующим образом

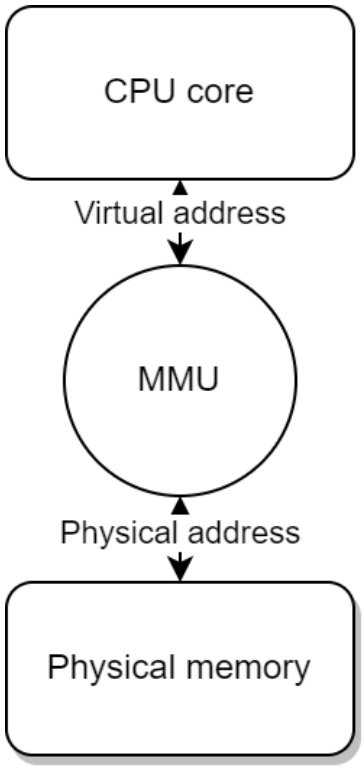

Рис 10. Стандартная организация виртуальной памяти.

8.1. Виртуальная память в гетерогенной системе.

Поскольку исторически управление виртуальной памятью графической карты происходило в контроллере, а собственно адресное пространство создавалось драйвером на хосте, этот способ организации виртуальной памяти остался и в гетерогенной вычислительной системе. То есть CPU не имеет возможности напрямую управлять MMU графической карты.

Это означает что в системе управления виртуальной памяти NetBSD придется внести значительные изменения.

Для управления виртуальной памятью будет использоваться интерфейс хостовой системы. Запросы на управления будут отправляться в хостовый драйвер. Он уже в свою очередь сформирует таблицы страниц и запишет их в память, а также правильным образом настроит MMU.

Этот интерфейс пока только имеет наметки, так ничего конкретного о реализации сказать нельзя. Однако можно точно схематично изобразить как это будет работать в будущем.

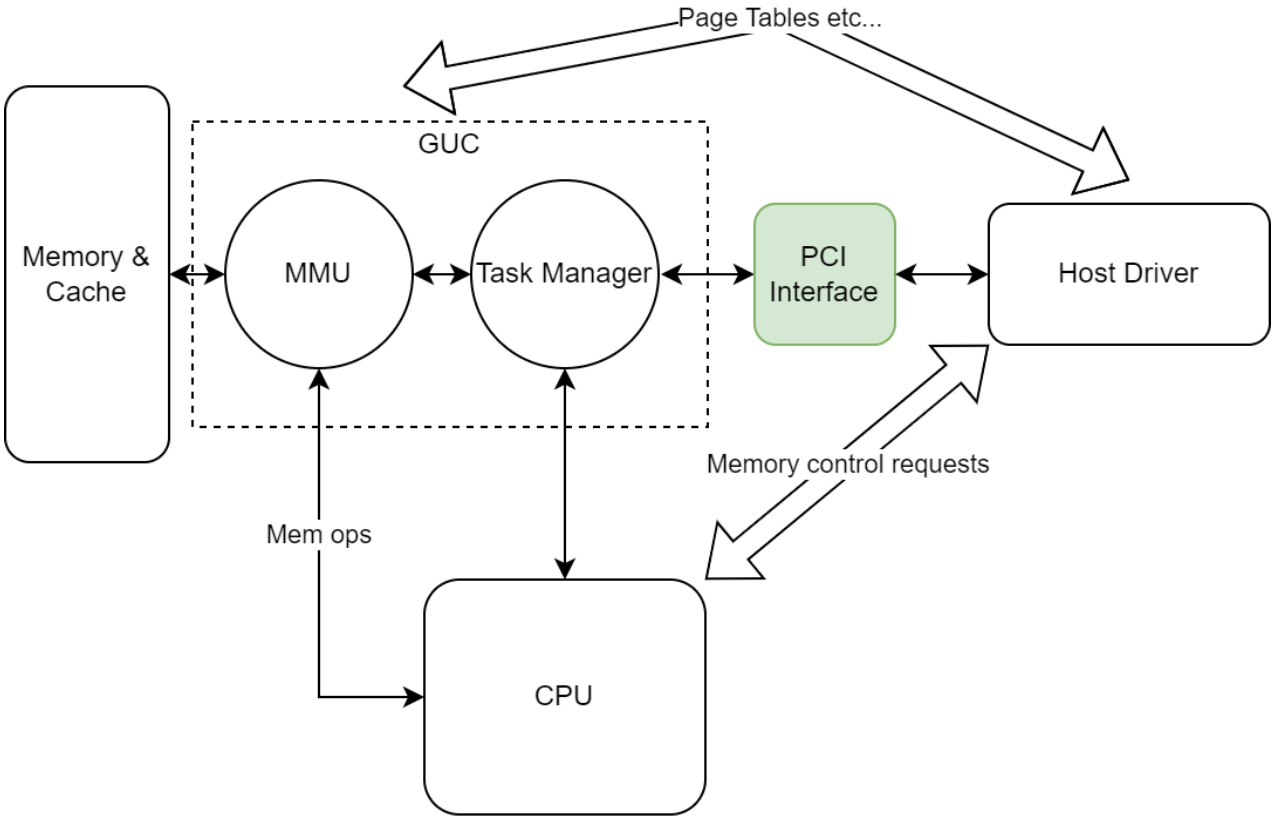

Рис 11. Специфика управления виртуальной памятью в гетерогенной системе.

8.2. Менеджер памяти NetBSD UVM/PMAP

UVM – абстрагированный от аппаратуры менеджер памяти. Он используется в системе буквально везде. Для ввода-вывода, для буферизаци файлов, для управления процессами и.т.п

PMAP – прослойка между UVM и аппаратурой. Эта та часть котора в первую очередь модифицируется при портировании. Поскольку мы не имеем возможности управлять виртуальной памятью, поскольку доступа доступа к MMU нет, а протокол управления еще не разработан, в этом модуле были оставлены только заглушки которые никак не влияют на виртуальную память, но «обманывают» UVM создавая видимость, что манипуляция выполнена.

Это конечно временный подход и на самом деле на что, чтобы пройти инициализацию виртуальной памяти было затрачено огромное количество времени, так как при таком подходе многие системные функции работают неправильно.

8.3. Адаптация менеджера памяти

В будущем необходимо в модуле pmap при помощи интерфейса с хостом выполнять те операции которые обычно выполняются модификацией таблиц страниц и управлением MMU. Такая реализация несет большие накладные расходы, а значит управление памятью при программировании приложений должно быть очень аккуратным во избежание частого обращения к хосту.

9. Взаимодействие приложений с ядром NetBSD

В этой главе будет рассмотрена тема общения приложений в операционной системе управляющей гетерогенной системой

9.1. Способ построения защиты приложений и системы.

Самый надежный способ защитить операционную систему от вредоносного воздействия приложений – аппаратный. В первую очередь это означает разместить код операционной системы в отдельном защищенном участке физической памяти. В нашей гетерогенной системе так и будет сделано. Запись кода операционной системы вообще не будет доступна для CPU. То есть при любых обстоятельствах, сам сопроцессор не сможет переписать код операционной системы и выполнить вредоносный код.

Также для защиты данных ядра от воздействия приложений, передача данных из адресного пространства приложения в адресное пространство ядра осуществляется только при помощи специальных функций, которые вызываются ядром, но обычного доступа на чтение ядро в пространство приложения не имеет.

9.2. Системные вызовы.

Интерфейс системных вызовов предназначен для использования приложениями сервисов предоставляемых операционной системой.

В нашей гетерогенной системе системные вызовы CPU происходят как обычные системные вызовы, с той разницей, что запросы сервисов, которые операционная система не может предоставить в силу строения гетерогенной системы перенаправляются на хостовый компьютер.

Так, например, для записи или чтения файла данные будут переданы в/из хостовую систему и там уже записаны в файл.

Перенаправляются на хостовую систему и запросы требующие манипуляций с виртуальной памятью, такие например как fork или mmap.

При ошибках системные вызовы возвращают код ошибки и устанавливают специальную переменную errno.

9.3. Особенности реализации copyin/copyout.

Система copy предназначена для передачи данных между ядром и приложением. Особенностью в нашей гетерогенной системе является то, каким образом происходит передача. Для копирования используются специальные инструкции, которые не вызывают исключение при доступе из ядра в пространство процесса.

Функции копирования работают последовательным чтением – записью, с оговоркой, что при возможности для копирования используется максимальная ширина слова, которое читается и записывается, используемая ширина зависит от выравнивания адресов копирования.

9.4. Методы синхронизации

Межпроцессная синхронизация в операционной системе NetBSD происходит при помощи примитива синхронизации называемого мьютекс (mutex). Реализованы мьютексы при помощи атомарных операций описаных выше.

Особенно стоит рассмотреть синхронизацию между графическими процессорами и скалярным сопроцессором. Синхронизация происходит через механизмы графического контроллера посредством передачи сообщений. Для этого в графическом контроллере имеются так называемые «почтовые ящики(mailbox)». При получении сообщения в CPU происходит прерывание и необходимый процесс разблокируется.

Эта часть адаптации является теоретической разработкой т.к. на данный момент разработка не дошла до взаимодействия CPU с графическими процессорами.

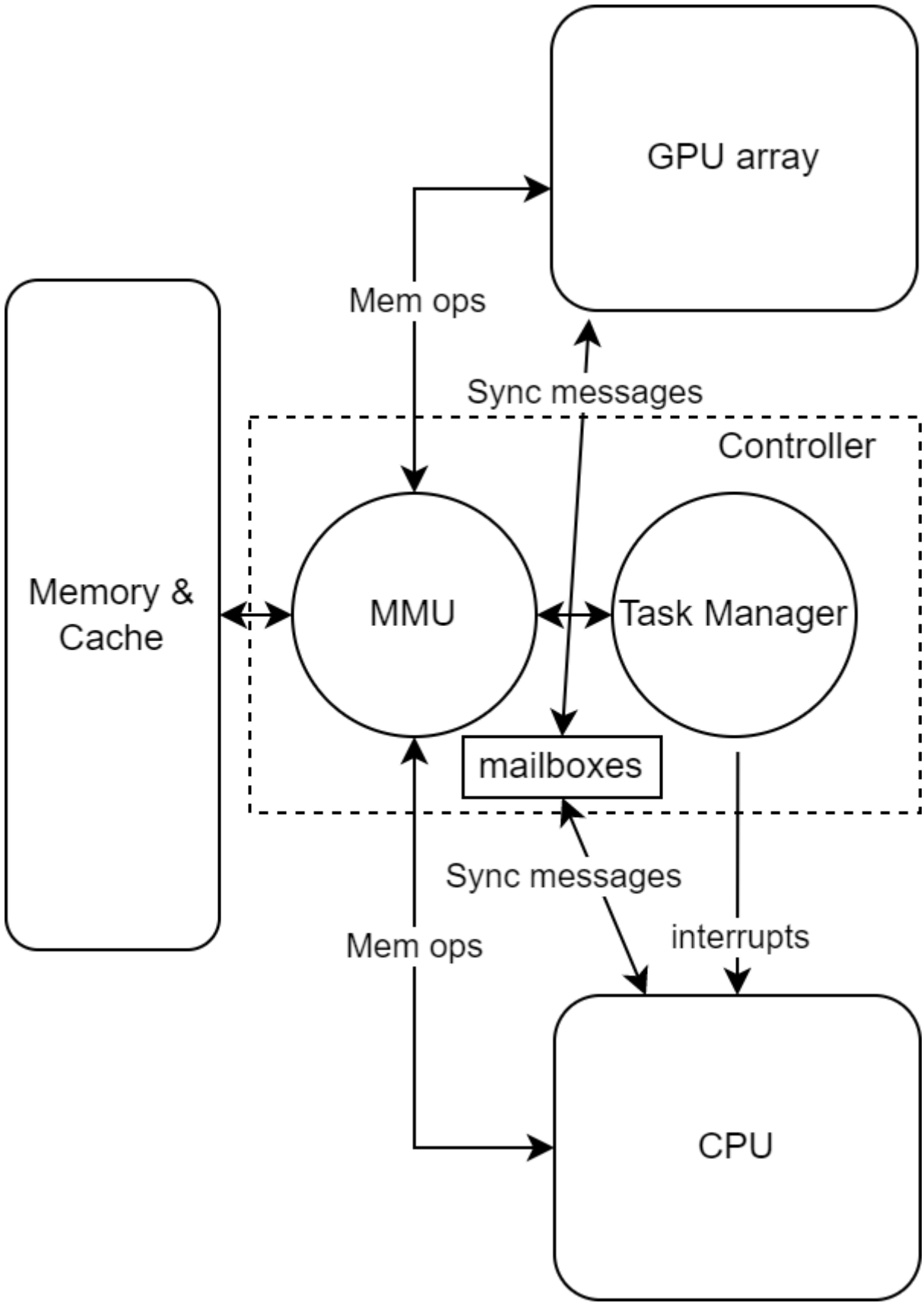

Рис. 12. Процесс синхронизации в гетерогенной системе.

10. Планировщик в NetBSD

10.1. Процессы NetBSD в гетерогенной системе

Процессы в гетерогенной системе состоят не только из образа программы для центрального процессора но и из образа для исполнения на массиве графических процессоров. При этом эти две программы могут напрямую взаимодействовать через общую память, что намного упрощает и ускоряет процесс передачи данных.

Однако планирование таких задач усложнено тем что, исполнение происходит не только на том процессоре которым может управлять планировщик.

Структура гетерогенного процесса показана на картинке ниже.

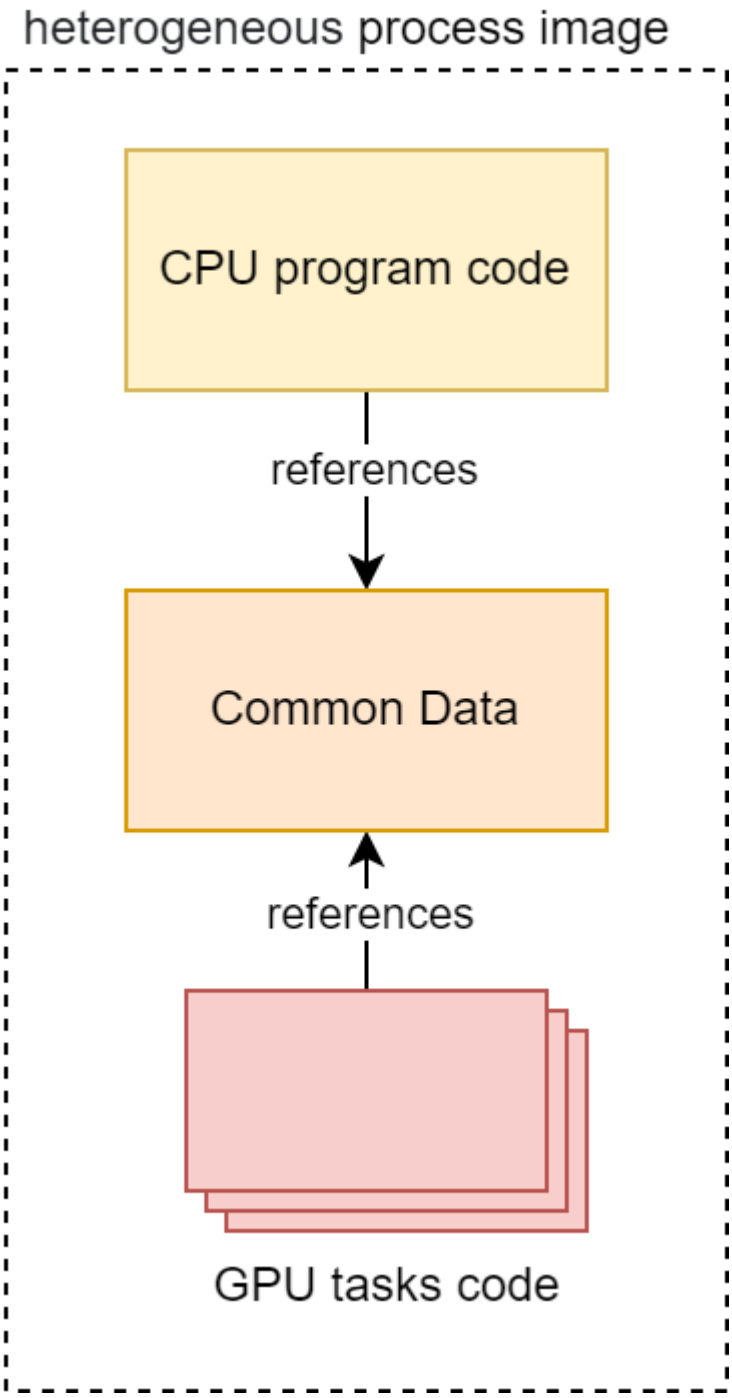

Рис 13. Структура процесса в гетерогенной вычислительной системе.

10.2. Планирование потоков.

Планирование потоков в NetBSD на гетерогенной системе будет работать нетипичным образом.

Оптимальная производительность будет достигаться если каждый поток в CPU будет ожидать завершения своей небольшой части задачи, исполняемой на GPU, чтобы обеспечить непрерывную загрузку GPU и вычислительные потоки CPU. Это так называемые асинхронные вычисления.

На рисунке ниже изображен процесс асинхронных вычислений. Каждому потоку на CPU соответствует задача, исполняющаяся на GPU. Как только задача на GPU завершилась процесс на CPU разблокируется и может сразу же работать с полученными данными.

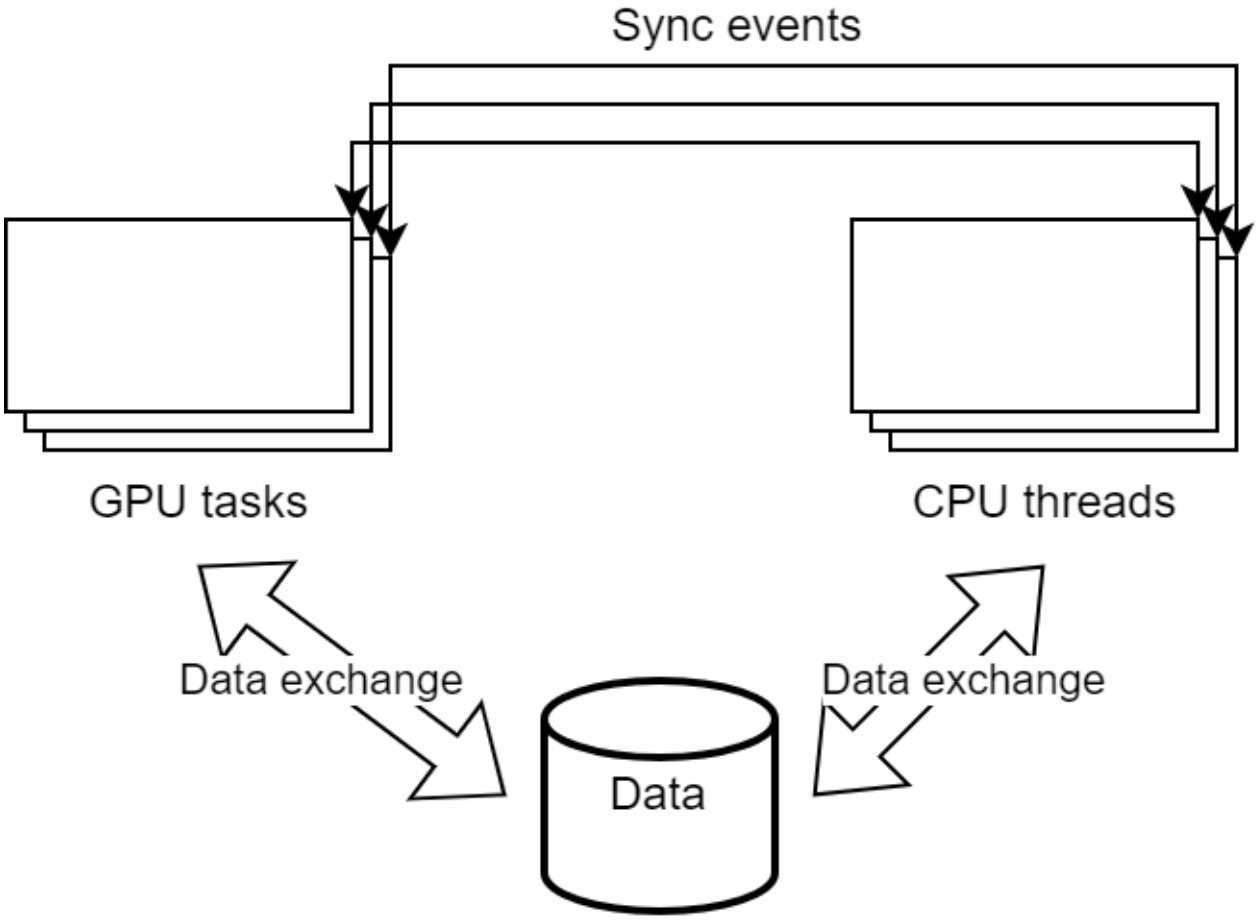

Рис. 14. Асинхронные вычисления.

Асинхронный характер вычислений позволяет равномернее нагружать каналы передачи данных между памятью и процессорами что снижает задержки доступа к данным.

10.3. Многопроцессорность в NetBSD

NetBSD поддерживает работу в мультипроцессорном режиме, при этом количество процессоров определяется динамически при запуске системы, таким образом использование NetBSD позволяет максимально утилизировать ресурсы CPU.

Функции для определения колиства и управления процессорами находятся в модуле locore машинно-зависимой части системы. Основные функции:

1. curcpu – возвращает указатель на дескриптор текущего процессора
2. mount_procfs – монтирует файловый интерфейс процессора
3. cpuset – модуль абстрагирующий набор процессоров.

1. Взаимодействия NetBSD с Хост операционной системой

Для выполнения некоторых функций операционной системы в гетерогенной вычислительной системы необходима работа с хост системой.

Для работы с операционной системой хоста был разработан дополнительный протокол канала взаимодействия драйвера хоста с операционной системой CPU (uKernel).

Протоколы взаимодействия между хост системой и CPU имеют следующую иерархию:

1. Шина PCI это протокол физического и канального уровня. По нему происходит передача данных содержащих другие протоколы взаимодействия.
2. Протокол контроллера GPU. Сложный протокол, обеспечивающий передачу всех необходимых данных на GPU, загрузку данных в память, а также протокол сообщений.
3. Протокол взаимодействия с uKernel. Был разработан специально для обеспечения работы операционной системы. Использует выделенный тип сообщений протокола нижнего уровня. Передает запросы она реализацию системных вызовов, отладочные данные, осуществляет

Схематически схема взаимодействия хостовой системой с uKernel показана на рисунке

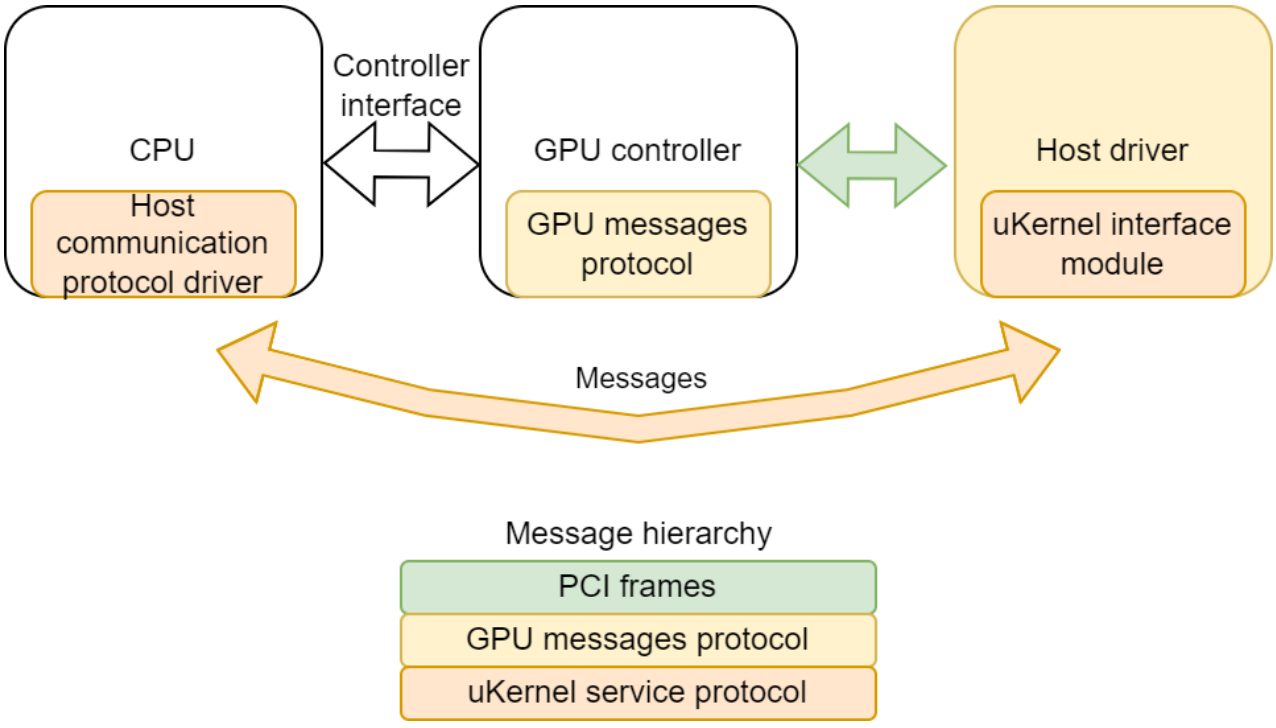

Рис. 15. Схема взаимодействия хостовой системой с операционной системой CPU.

11.1. Перенаправление системных вызовов.

Многие из системных вызовов происходящих во время работы приложений в операционной системе NetBSD не могут быть выполнены на CPU в силу того что гетерогенная вычислительная система накладывает ограничения на функционал ОС.

Вот основные из них:

- fork. В связи с тем, что uKernel не имеет возможности управлять виртуальной памятью, то и создавать копии таблицы страниц для процесса он не может. При получении запроса на такой системный вызов драйвер хоста выполняет необходимые манипуляции с адресным пространством.

- mmap, mprotect… Также невозможно выполнить из-за специфики управления памятью. При получении этих запросов драйвер производит нужные манипуляции с адресными пространствами и данными отображаемых файлов.

- Read, write, open… Также иногда перенаправляются на хост из-за того что в ядре отстутствует поддержка файловых систем. Манипуляции с файлами происходят на стороне хостовой операционной системы а данные передаются в/из графической карты.

Перенаправление системных вызовов безусловно очень накладная по времени операция, таким образом разрабатываемое программное обеспечение должно быть разработано с учетом того чтобы минимально ими пользоваться. Например исполняемые программы не должны вызывать fork, а данные которые будут обрабатываться алгоритмом передаваться сразу в образе памяти процесса, а не считываться из файла.

11.2. Дополнительные сервисы.

Дополнительные сервисы необходимы для отладки и функционировани я программ в гетерогенной вычислительной среде. Основными такими сервисами являются:

- Отладочный интерфейс ядра. Реализован в виде GDB stub в ядре ОС CPU.
- Отладочный интерфейс приложений работающих в среде uKernel.
- Отладочный символьный ввод-вывод. Необходим для отладки ядра ОС и приложений.

2. Особенности запуска и исполнения NetBSD в гетерогенной системе.

12.1. Загрузка, инициализация и раскрутка

В гетерогенной среде процесс запуска операционной системы несколько отличается от обычного запуска операционной системы.

Так, например, ядро оказывается в памяти уже полностью загруженным драйвером хоста, таким образо драйвер хоста выполняет роль загрузчика.

Также хостовый драйвер сразу создает таблицу страниц ядра и включает MMU, поэтому не требуется на ранних этапах запуска системы переходить из исполнения по физическим адресам в исполнение в виртуальном адресном пространстве.

К сожалению, помимо упрощений, отсутствие возможности управлять виртуальной памятью приносит множество проблем при работе, так, например, очень долгое время ушло на то, чтобы «обмануть» UVM, и пройти этап инициализации системы виртуальной памяти.

В конечном итоге, работа над запуском ОС остановилась на этапе монтирования корневой файловой системы.

Поскольку поддержка файловых систем была отключена в ядре, то не было возможности смонтировать корневую файловую систему и запустить из нее init процесс.

Процесс запуска системы представлен на иллюстрации.

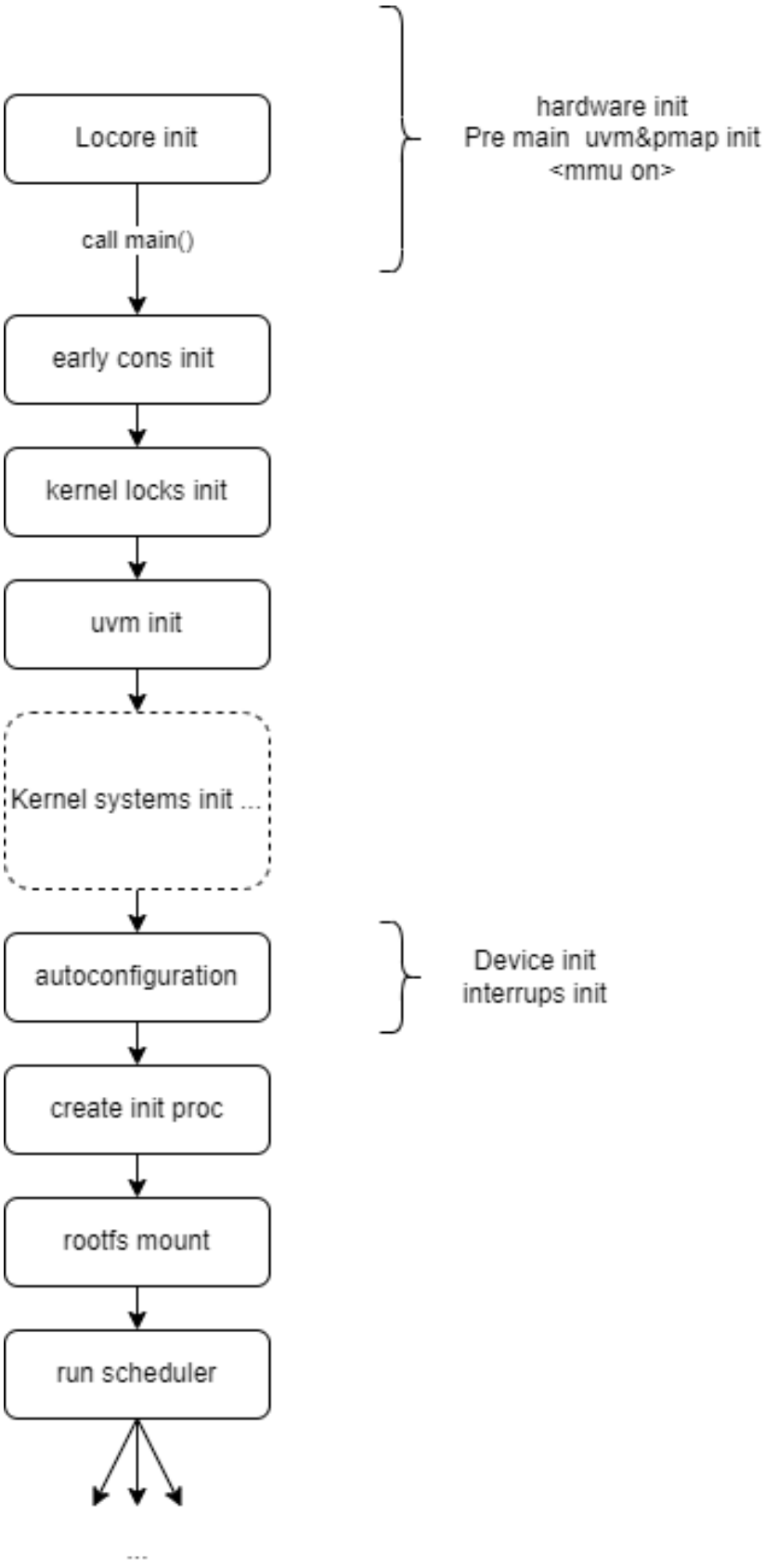

Рис. 16. Процесс запуска операционной системы NetBSD.

3. Исполнение и отладка NetBSD на симуляторе системы.

При разработке таких масштабных программных продуктов как гетерогенная вычислительная карта, невозможно сразу отлаживать программное обеспечение на аппаратуре. Это связано в первую очередь с тем, что аппаратура разрабатывается одновременно с программным обеспечением и точная подгонака и отладка происходит до того момента как микросхемы будут поставлены на производство. Это снижает вероятность критических ошибок в микросхеме и удешевляет и ускоряет производство.

Очевидно, для такого подхода нужно симулировать функционал аппаратуры программно для того чтобы запускать программное обеспечение и отлаживать его.

13.1. Интегральная система функционального моделирования

Поскольку вычислительная система у нас гетерогенная, то и симулятор ее должен быть гетерогенный.

На момент написания работы, интегральный симулятор гетерогенной системы создан еще не был, однако, его строение уже было сформулировано и утверждено, так что рассмотреть его считаю необходимым.

Интегральный симулятор состоит из следующих основных частей:

- Симулятор массива GPU. Intel имеет проприетарный симулятор GPU.
- Симулятор CPU. Также проприетарный симулятор компании Intel, как и сама архитектура.
- Модуль который воссоединит эти два симулятора воедино. И обеспечит эмуляцию общей памяти
- Модули отладки для ПО GPU и ПО CPU.
- Модуль эмулятор ввода-вывода.

Логическая схема интегрального симулятора выглядит следующим образом:

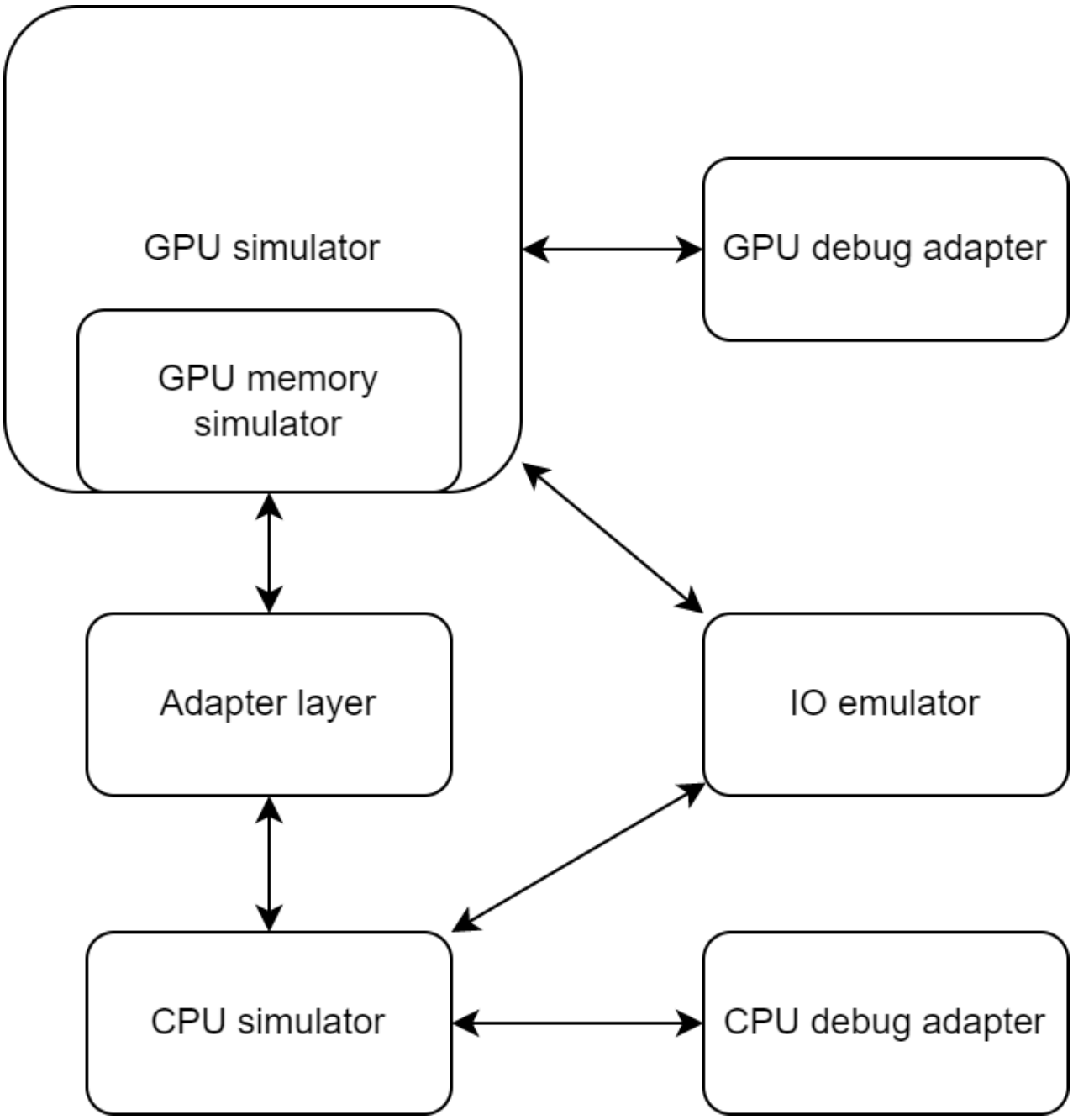

Рис 17. Структура интегрального симулятора гетерогенной системы

13.2. Отладочные средства

Для отладки в симуляторе CPU имеется возможность запускать его совместно с GDB и достаточно комфортно отлаживать выполняющийся на симуляторе код.

Также для отладки NetBSD на CPU был разработан небольшой модуль для функционального симулятора CPU, предназначенного для осуществления символьного ввода-вывода.

Реализован он был добавлением двух виртуальных системных регистров. Один из которых содержал данные для ввода-вывода, а второй предназначался для статуса и команд.

Работа этого модуля изображена на рисунке:

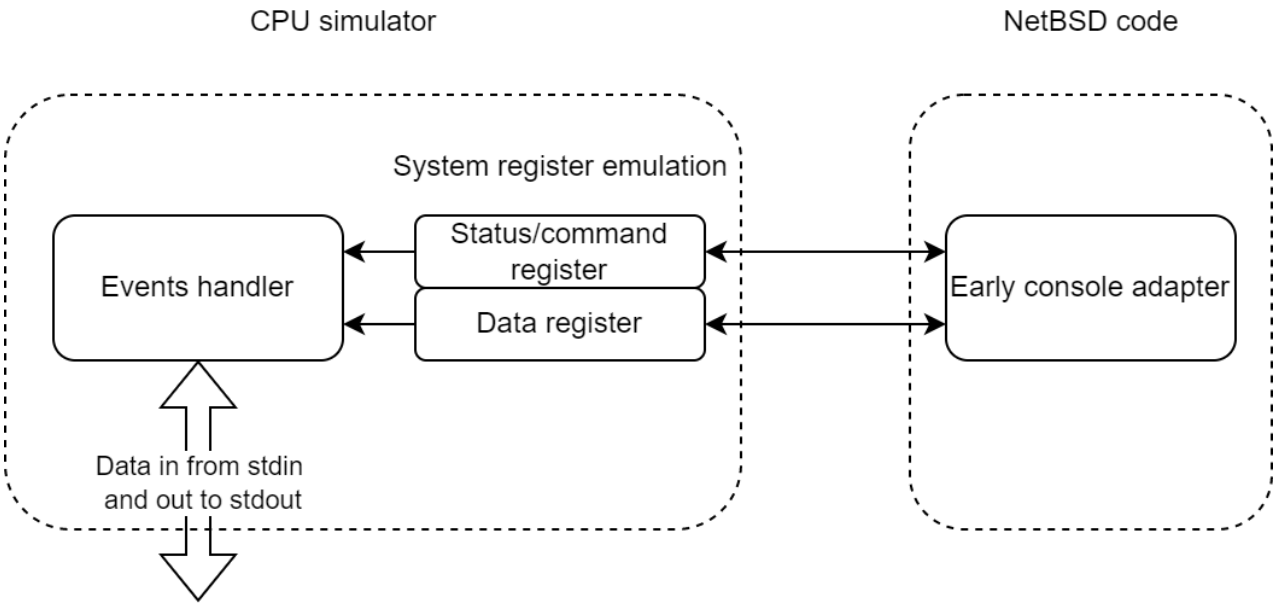

Рис 18. Система символьного ввода вывода симулятора.

4. Заключение

Изучена архитектура гетерогеннной вычислительной системы. Сформулированы требования к операционной системе среды исполнения гетерогенных вычислительных задач. В соответствие с выдвинутыми требованиями выбрана операционная система NetBSD. Изучено внутреннее устройство системы NetBSD. Создана заготовка порта операционной системы. Портированы некоторые модули операционной системы. Некоторые модули портированы частично. Достигнут некоторый прогресс в запуске операционной системе на симуляторе.

Дальнейшие планы:

- Окончание портирования ОС
- Отладка на симуляторе
- Тесты производительноси на симуляторе

Выводы:

- Новые подходы к проектированию вычислительных систем могут дать значительный прирост производительности вычислений.
- Разработка программно-аппаратных комплексов в современном мире требует одновременной работы над программным и аппаратным обеспечением. Симуляция аппаратуры в такой ситуации позволяет избежать значительных задержек в разработке, а также позволяет избегать некоторых ошибок проектирования.
- Современные ОС общего назначения подходят для управления гетерогенными вычислительными системами лишь с некоторыми условностями. Возможно необходим новый подход к проектированию операционных систем.

5. Использованная литература

- Agarwal. A. 1989. Performance tradeoffs in multithreaded processors. MIT VLSI Memo No. 89-566.
- Nickolls, J., and Dally, W. J. 2010. The GPU computing era. *IEEE Micro 30*, 2, 56–69.
- Vasily Volkov Understanding Latency Hiding on GPUs Electrical Engineering and Computer Sciences University of California at Berkeley Technical Report No. UCB/EECS-2016-143.
- McKusick M.K. Neville-Neil G.V. Watson R.N.M 2016. The Design and Implementation of the FreeBSD Operating System 2nd Edition
- NetBSD manual pages https://man.netbsd.org/
- NVIDIA. 2012c. *Dynamic Parallelism in CUDA*. Technical Brief. NVIDIA.
- NVIDIA. 2013. *CUDA C Programming Guide v5.5*. NVIDIA. July 2013.