Мне была поставлена задача написать гипервизор 1-го типа, который загружался с флешки (на неё должен был быть выставлен приоритет загрузки в биосе) и виртуализировал ОС, находящуюся на основном жестком диске. Сам гипервизор должен был оставаться в оперативной памяти. Результатами первого этапа работы стал гипервизор, за основу которого был взят проект с гитхаба (https://github.com/staticbear/simple_hypervisor), который отличался от всех других проектов тем, что он собирался без сторонних библиотек, используя только компилятор gcc, линковщик ld, утилиту make и ассемблерный компилятор языка fasm; также он запускался до ОС, а не из-под неё, что немного усложняет начальную настройку параметров гостевой ОС.

В данном проекте был переделан способ загрузки гипервизора в оперативную память (раньше это делалось через механизмы ACHI, но мне не удалось его заставить работать из исходников, поэтому пришлось его заменить), а также пришлось модифицировать первый сектор гипервизора, чтобы в нем сохранилась mbr от операционной системы. В результате этих изменений я смог запустить свою сборку на физическом компьютере. На рисунке 1 можно увидеть, как гипервизор изменил вендора процессора на ОС windows7 x32 (таким образом я и далее буду демонстрировать тот факт, что ОС работает под гипервизором).

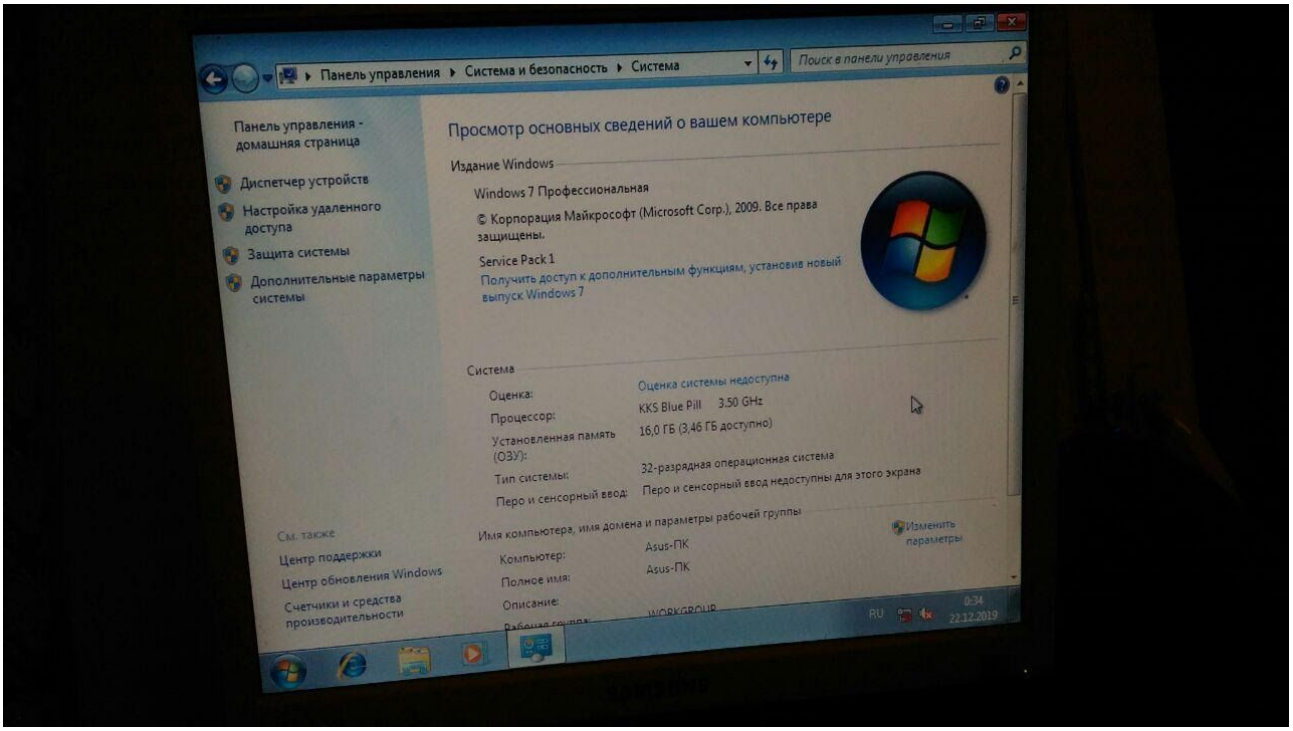

Рисунок 1

Далее я решил переделать гипервизор под работу на моей основной машине с использованием qemu, а также захотел заставить гипервизор запускать не windows, а линукс (чтобы его можно было отлаживать, исследуя исходные коды ОС). И здесь я начал сталкиваться с проблемами. Для начала я обнаружил, что гипервизор virtualbox неправильно реализует вложенную виртуализацию — я очень долго не мог выяснить, что проблема именно в нем, а не в моём гипервизоре, так как ошибки происходили стихийно на различных этапах работы гипервизора. Данную проблему помог решить переход на гипервизор VMware.

Также я решил на первый взгляд сложную проблему, с которой я столкнулся ранее: изначально гипервизор запускался с того же диска, на котором находилась операционная система (данному диску биос присвоил номер 0x80). А нам нужно было, чтобы гипервизор жил на флешке с номером 0x80 (у диска был номер 0x81 - порядок загрузки с носителей в биосе должен быть выставлен таким образом), а после запуска гипервизора порядок загрузки должен был меняться (у флешки номер становился 0x81, а у жесткого диска —

0x80, и после этого должна была бы происходить загрузка ОС). Решилась эта проблема достаточно просто — оказывается, существует правило, по которому при передаче управления первому сектору жесткого диска (который уже должен лежать по адресу 0x7c00 в оперативной памяти) в регистре dx должен содержаться номер диска, с которого будет идти загрузка — поэтому при старте гостевой ОС мы просто присваиваем регистру dx значение 0x81, и загрузка происходит с нужного нам диска. Данный подход позволил нам не вносить никаких изменений на диск с гостевой ОС, что позволит нам запускать гипервизор на виртуальных машинах qemu.

Далее я решил сделать гипервизор более гибким — изначально он всегда выделял операционной системе 4 Гб оперативной памяти и забирал оставшееся место себе. Теперь же он не зависит от цифры 4 Гб и может быть перемещен куда угодно в памяти — этого удалось добиться изменением ld-скрипта и добавлением флага -fPIE при компиляции (position-independent executable code - данный флаг используется для того, чтобы не только адреса функций были относительными (как при использовании -fPIC), но и адреса в секции данных тоже). То есть теперь для работы гипервизора можно использовать компьютеры с меньшим количеством оперативной памяти, чем 4 Гб. Также в связи с данным изменением пришлось внести очень много правок в саму структуру гипервизора (связывать файлы не через константные значения, а через систему глобальных переменных, при этом изменяя работу некоторых функций).

Также на данный момент удалось выяснить причину странного поведения одного и того же кода на разных рабочих машинах, которое я наблюдал — все дело оказалось в различиях компонент процессоров. Это отражается на MSR-регистрах (model specific registers) – существуют регистры, которые открыты только на чтение, и которые отображают те комплектующие, которые обязательно должны быть включены при виртуализации, иначе гипервизор не будет запускаться (такой механизм самоконтроля). Также в процессе отладки я немного неправильно обрабатывал обращения из гостя в гипервизор: из-за этого у меня неправильно отрабатывала инструкция CPUID - поэтому в

какой-то момент я смог запустить на нем 32-х битную ОС Ubuntu 9.04 с выключенной PAE.

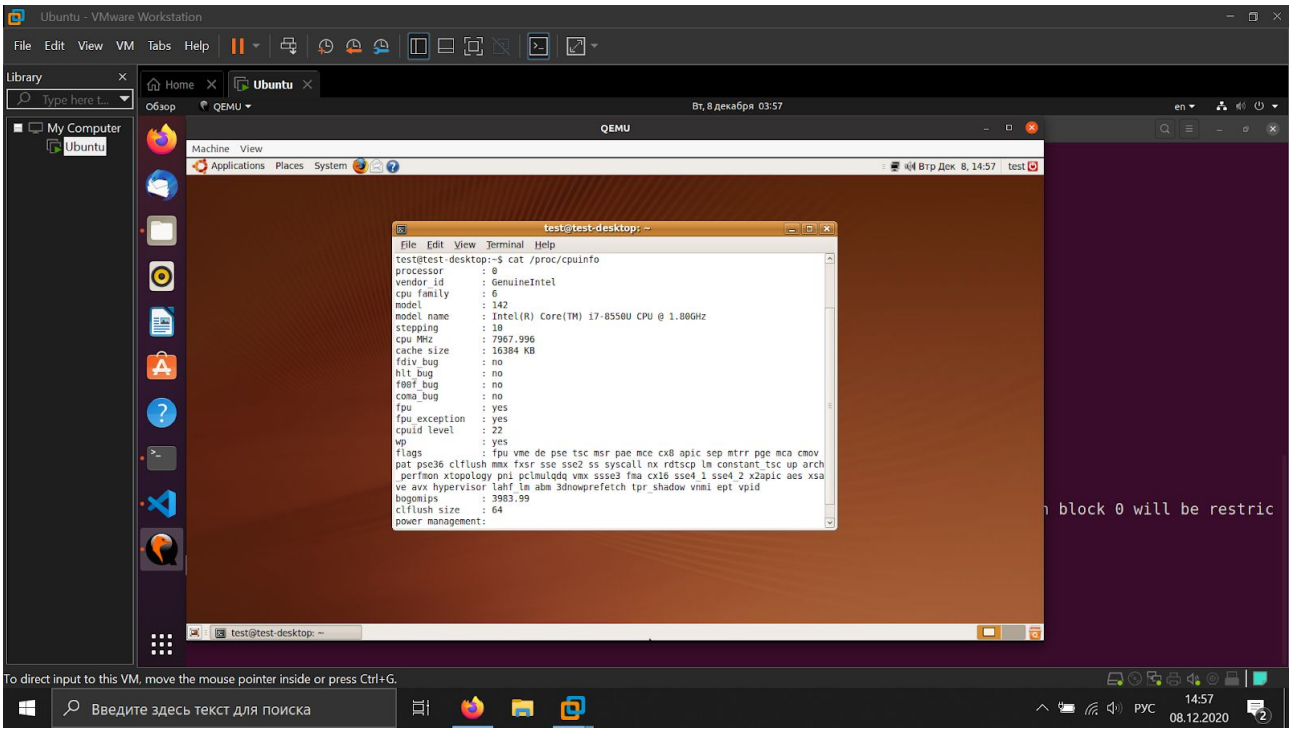

Рисунок 2 — запуск ОС напрямую в qemu

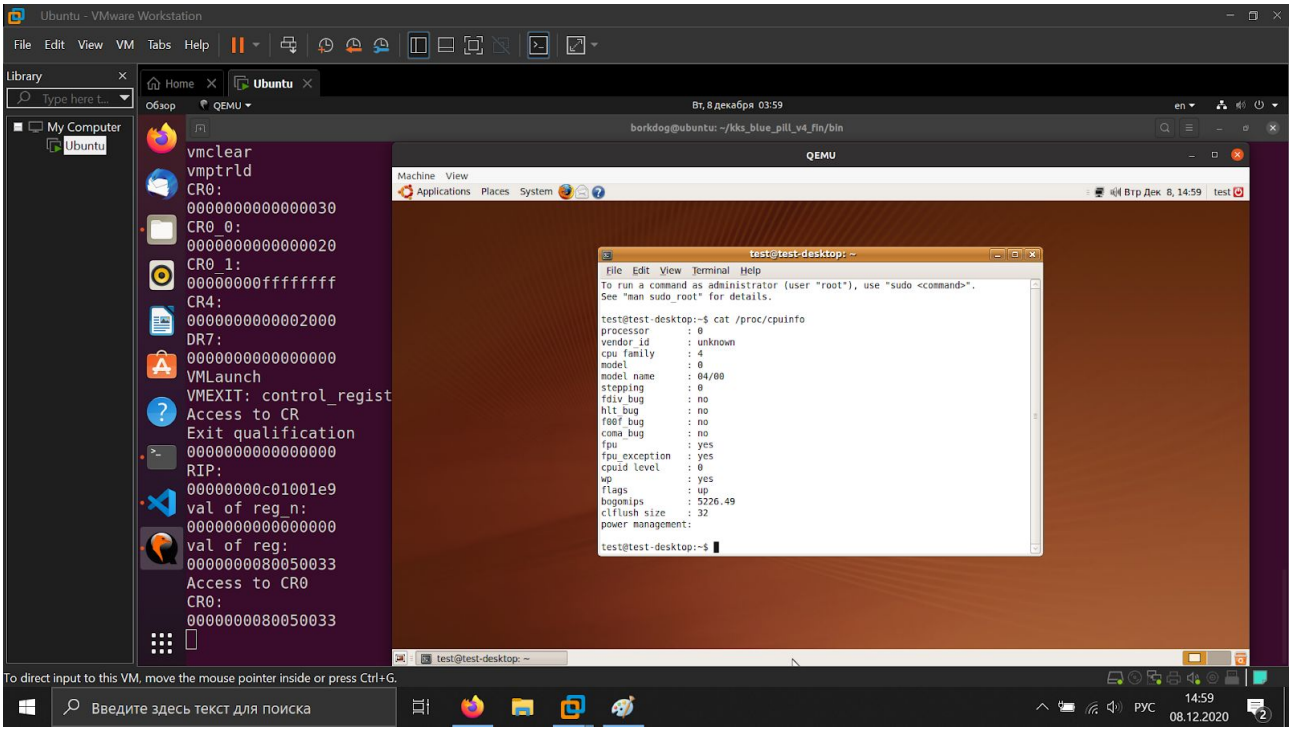

Рисунок 3 — запуск на моем гипервизоре

На рисунке 3 видно, что ОС не может распознать вендора процессора и многие его характеристики — все из-за неправильной обработки команды CPUID.

После исправления данной неисправности любая ОС перестала запускаться корректно, потому что процессор стал поддерживать много новых функций, на которые мой гипервизор просто не рассчитан. Поэтому мне пришлось написать маленькое ядро, которое загружает себя в память, заходит в protected mode и считывает строку с вендором процессора с помощью функции CPUID, после чего автоматически завершает работу. Весь вывод ведется через последовательный порт, который в qemu подключается к stdout. Ядро запускается bash-скриптом в папке ./test_OS/run.sh, а гипервизор, который виртуализирует данное ядро, запускается из папки ./hypervisor/bin/run.sh Он использует 2Гб оперативной памяти вместо 4-х для демонстрации своей мобильности. (Перемещать в памяти его всё равно стоит аккуратно, так как в памяти есть области, недоступные для записи - их можно узнать из структуры memory RAM Map, вызывающейся с помощью функции int 0x15 при eax = 0x0e820). Гипервизор занимает на данный момент 16 Мб в самом конце оперативной памяти, хотя может и меньше (но сейчас это минимальный параметр для его стабильной работы, подобранный опытным путём). На рисунке 4 изображена работа ядра без гипервизора, а на рисунке 5 — работа под гипервизором (на нем видно, что был произведен перехват инструкции CPUID и были изменены результаты её работы).

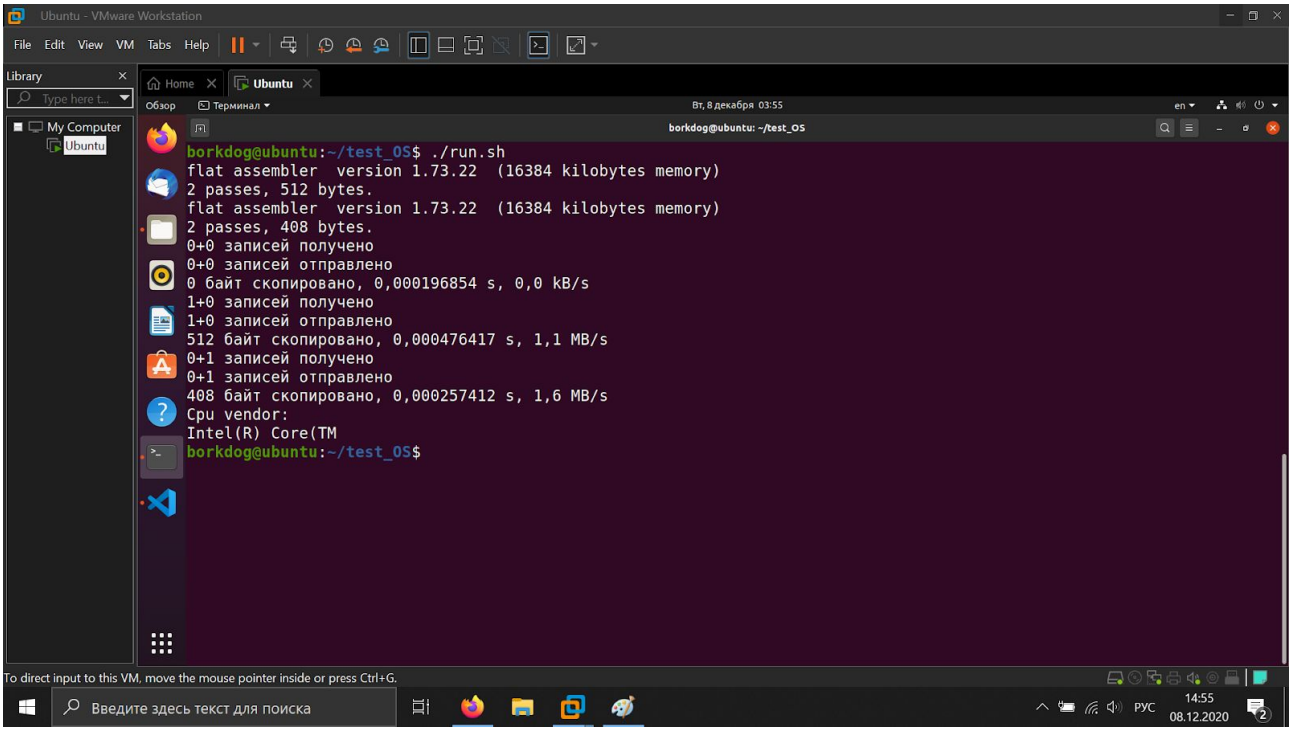

Рисунок 4

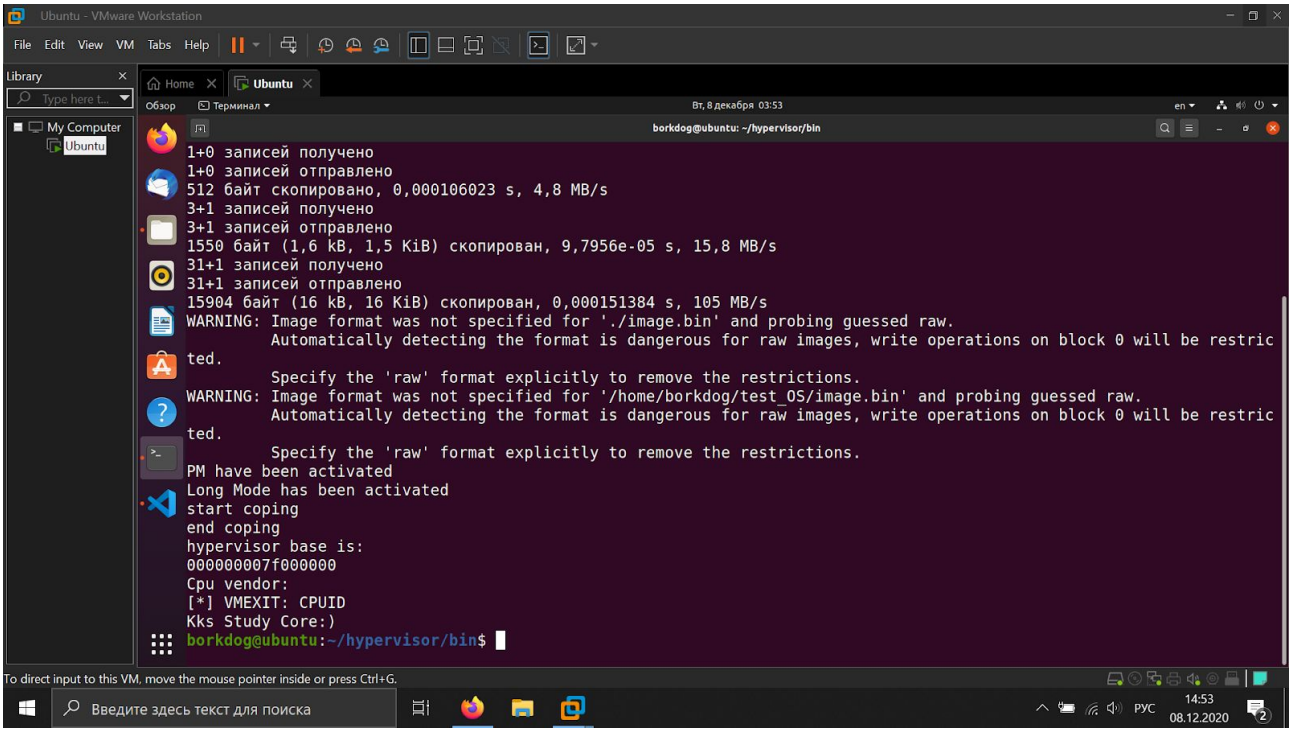

Рисунок 5

В дальнейшем я планирую переделать проект таким образом, чтобы он смог запустить полноценную linux-систему.