ЭМУЛЯТОР ОПЕРАЦИОННОЙ СИСТЕМЫ MS-DOS И ПРОЦЕССОРА АРХИТЕКТУРЫ X86

Головин Е.С. Жук Я.С.

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Оношко Д.Е. – старший преподаватель

В связи с тем, что современные операционные системы зачастую не поддерживают перевод процессоров архитектуры x86-64 в режим Virtual-8086 Mode, выполнение программ, написанных для MS-DOS, на них становится невозможным без сторонних приложений, воссоздающих среду, в которой такие программы могли бы выполняться в защищенном режиме. В проекте была совершена попытка создать для программ среду выполнения, в которой они будут запускаться и выдавать корректные результаты.

Эмуляция адресного пространства и видеорежимов. Ранние модели процессоров архитектуры x86 могли адресовать 1 МБ, в который помимо самой оперативной памяти входили и несколько областей памяти для различных видеорежимов. Для увеличения производительности было принято решение выделить область в оперативной памяти размером в 1 МБ, в который загружается исполняемая программа, таблица векторов прерываний, а также начиная со смещения 0A0000h относительно начала выделяемой области, в случае перехода в видеорежим 13h, в растровое изображение загружались данные из всей область памяти, которая относилась к этому видеорежиму.

Эмуляция регистров. Эмуляция была осуществлена с помощью создания массивов эмулируемых регистров общего назначения и сегментных регистров, в которых находятся значения, хранимые программой. Работа с эмулируемыми регистрами как с массивами вместо традиционного представления контекста программы как записи (структуры) обусловлена тем, что машинный код инструкций, использующих регистры, содержит битовые поля, значения которых можно использовать в качестве индексов в эти массивы.

Декодирование инструкций. Первый байт, прочитанный на данном такте процессором, интерпретировался как числовое обозначение инструкции. Для упрощения логики, был создан массив меток размерностью в 256 элементов, в котором хранились адреса обработчиков данных инструкций. Далее в обработчике в зависимости от типа инструкции могла вызываться функция анализа ModR/M byte [1], при его наличии, либо же при детерминированности операндов сразу же выполняться без вызова дополнительных функций.

Анализ ModR/M byte. Анализ сводился к отдельному рассмотрению каждого из полей. Значение поля mod, если оно лежало в диапазоне [0-2], обозначало размер константного значения, которое в сумме с комбинацией регистров, закодированной в поле R/M, задавало адрес операнда в памяти. Поле reg, в случае двух операндов у инструкции, задавало один из регистров общего назначения. Размер операндов определялся на этапе декодирования инструкции. В случае, если в инструкции предусмотрен лишь 1 операнд, как, например, в инструкции inc ah, в поле reg кодировалась инструкция (например, reg=0, тип инструкции inc, reg=1, тип инструкции dec). Первый байт, интерпретируемый как opcode, выступает в качестве префикса для двухбайтовой инструкции. Размер инструкции подсчитывался динамически, для этого была выделена отдельная переменная, в которую по мере интерпретации этого байта добавлялись размеры константных операндов, участвующих в инструкции.

Эмуляция BIOS. Одной из важных задач было сделать эмуляцию работы BIOS, а конкретно обработку клавиатурного ввода и прерываний системного таймера. Кроме того, вследствие работы с данными видами прерываний происходит заполнение / изменение такой структуры (клавиатурного буфера, параметров системного таймера и т.п.), как BIOS Data Area (BDA) [2], её составление также необходимо. Так как взаимодействие пользователя с программным средством организовывается посредством Windows API, то суть обработки клавиатурного ввода заключается в том, чтобы:

– создать корректный цикл обработки приходящих сообщений;

– выявлять нажатие на такие клавиши, как SHIFT, ALT, CTRL и т.д., для выставления флагов в BDA;

– различать нажатие клавиш символьного ввода и Extended ASCII;

– заносить в клавиатурный буфер скан-код и ASCII-код нажатой обычной клавиши;

– изменять значения указателей клавиатурного буфера («головы» и «хвоста»);

– отслеживать переполнение клавиатурного буфера и реагировать на него (издавать звук частотой в 783 Гц длительностью 270 мс, не отображать введенный символ).

Эмуляция программных прерываний MS-DOS. Было решено так же, как и в MS-DOS, использовать таблицу прерываний. Однако, так как реализация обработчиков всех прерываний не имела необходимости, но «оригинальные» смещения для прямой связи вызываемых программ и памяти были необходимы, была реализована «заглушка» – значение, на которое указывали ссылки нереализованных прерываний и которое является неподдерживаемой инструкцией, которая, следовательно, никогда не встретится в работающей программе, используемое как знак того, что

необходимо передать управление на обработку эмулируемой ОС, а не начинать выполнять записанный начиная с этого места машинный код как пользовательский обработчик прерывания. В качестве такой «заглушки» был выбран префикс lock, который не поддерживается по той причине, что эмулятор не предполагает распараллеливания вычислений.

Эмуляция работы сервисов. В связи с тем, что целью было создание среды для запуска программ, возникла необходимость реализации основной функциональности таких сервисов, как 10h и 16h, которые бы работали с «оперативной памятью» как в реальном режиме – смещение напрямую использовалось в работе эмулятора, то есть не пересчитывалось, а лишь прибавлялось к адресу этой самой памяти, которую, с целью снижения размера исполняемого файла, было решено выделить в неинициализированной области. Однако работа сервисов предполагает, что в «оперативной памяти» с оригинальными смещениями существует область для видеорежимов, а также область клавиатурного буфера.

Видеорежимы. В эмуляторе реализовано 2 наиболее используемых в лабораторных работах в курсе «Конструирование программного обеспечения» (часть 1) видеорежима: 03h и 13h. Для графического режима была выделена память прямо в эмулируемом адресном пространстве, так как корреляция отображаемых пикселей с байтами в памяти прямая, в отличие от текстового режима, где одно знакоместо представлено двумя байтами, задающими символ и его атрибуты. Кроме того, растровый шрифт был скопирован с физического устройства под управлением MS-DOS с помощью специально разработанной для этих целей программы. Однако символа для каретки ввода в шрифте нет, так что отображение каретки осуществлялось её отрисовкой отдельно от текстового содержимого. Размер каждого символа в полученном ранее растровом шрифте составляет 16 байт, а также особенности вывода растрового изображения, которые было решено использовать в качестве инструмента отображения видеопамяти за его преимущества в виде масштабируемости и корректируемости, в кратности памяти, привели к выделению области памяти для текстового режима в 35000 байт; тогда как область памяти для графического режима имеет размер 64000 байт. Каждое растровое изображение требует выбор режима работы с цветом и саму палитру, тем самым возникла необходимость получить палитру, экспорт которой также был осуществлён с физического устройства под управлением MS-DOS. И палитра, и шрифт загружаются при запуске исполняемого файла эмулятора.

Эмуляция PSP и загрузки программ в RAM. Так как в данной реализации эмулировалось взаимодействие операционной системы с COM-программами, необходимо было загружать содержимое исполняемого файла по смещению 100h, до которого заполнить такую структуру, как PSP (Program Segment Prefix) [3], а также настроить эмулируемые регистры, эмулируемый стек и эмулируемые CS:IP, которые будет указывать на место в эмулируемой памяти, с которым эмулируемому процессору необходимо будет производить «чтение» и «выполнение».

Разработка основного цикла обработки сообщений. Работа с Windows API подразумевает взаимодействие с множеством приходящих окну сообщений, например, WM_PAINT, WM_KEYDOWN. Конкретно на этом взаимодействии построена связь эмулируемых ОС и процессора при наличии / отсутствии загруженного исполняемого файла; нажатия и работы пользователя с окном и работы программного средства. Кроме того, требовалось эмулировать работу видеоадаптера по отображению содержимого видеопамяти. В физических устройствах это происходит параллельно работе процессора. Для эмуляции этой особенности работы аппаратуры и в то же время минимизации вычислительной нагрузки, создаваемой обновлением окна эмулятора, было принято решение инициировать перерисовку при каждой второй передаче управления эмулируемым процессором операционной системе.

Суть данного цикла обработки сообщений строится на использовании самих функций Windows API для проверки наличия сообщений в очереди и их получения, а также на анализе значений некоторых флагов:

– isWaitingInput – если какое-либо прерывание в эмулируемой среде дожидается ввода пользователя;

– isProgram – загружен ли исполняемый файл и происходит ли его исполнение.

Список использованных источников:

1. Intel® 64 and IA-32 Architectures Software Developer's Manual. – Intel Corporation, 2023. – 5066 p.

2. System BIOS for IBM PC/XT/AT computers and compatibles. – 5th printing. – USA : Phoenix Technologies, 1990. – 554 p.

3. TechHelp 6.0 [Электронный ресурс] : программное средство / Flambeaux Software. – 1994.