Проектирование модулей загрузки и работы с файловой системой в составе XoS как гибрида операционных систем семейства Windows NT

Бабков В.С., Пехотин Е.В.
Донецкий национальный технический университет
victor@babkov.name

Abstract

Babkov V., Pehotin E. Development loading and file system modules in XoS as Windows NT family hybrid. *Base kernel function (file system and system loader) in Windows NT family are reviewed. New integrated file system for XoS is proposed. New system loader architecture is discussed. Development an executable file for system loader and XoS file system module are described.*

Введение

Операционную систему можно понимать, как комплекс программных средств, выполняющих две основные функции: расширение возможностей машины и управление ее ресурсами [1].

Операционные системы (ОС) семейства Microsoft Windows – наиболее распространенные в мире. Существует несколько платформ этих ОС, однако в настоящий момент можно говорить только об одной – WinNT (остальные уже устарели и не поддерживаются Microsoft). Истоки WinNT восходят к октябрю 1988 года, когда MS решила создать ОС, совместимую с OS/2, обладающую высокой защищенностью и надежностью и поддерживающую многопроцессорную обработку и POSIX [2]. В 2001 году на свет появилось Windows XP – ОС платформы WinNT версии 5.1, а в 2007 вышла последняя на данный момент ОС Microsoft – Windows Vista.

Два известных факта – ОС Microsoft широко распространены и дороги – должны были привести к созданию нескольких клонов WinNT, однако это не происходит. Единственная известная авторам freeware ОС класса WinNT – ReactOS [3], авторы которой стараются сделать ее похожей на WinXP, но с учетом новых тенденций, внесенных ОС Windows Vista.

Несмотря на то, что структуре и реализации базовых функций ОС семейства Windows посвящено много работ [4-5], остается много «белых пятен» и недокументированных особенностей, которые важны для понимания работы ОС, повышения эффективности ее работы, однако подробно в литературе и технической документации не освещены.

В данной работе ставится задача исследовать реализацию базовых функций ОС в семействе Windows NT и предложить новую интегрированную и оптимизированную схему построения ОС на примере нескольких базовых модулей (файловая система и загрузчик ОС). Работа выполняется в рамках проекта по созданию альтернативной ОС класса WinNT – (x) Operation System (XoS). Данный проект разрабатывается авторами с 2007 г, как универсальная платформа, интегрирующая в себе лучшие стороны всех ОС семейства Windows NT в усовершенствованном варианте.

Вопросы разработки эффективных файловых систем для различных ОС освещаются во многих источниках [6-8]. Поскольку в современных условиях файловым системам присущи такие характеристики как: объемность, распределенность, безопасность, надежность, то проектирование и модификация уже существующих систем с целью увеличить вышеуказанные показатели является несомненно актуальной задачей.

Интеграция файловых систем FAT и NTFS в составе проектируемой ОС

Семейство файловых систем (ФС) XFS ((x) File System) разрабатывалось на основе результатов анализа структур ФС FAT (FAT12-FAT32) и NTFS for NT 5.1 с учетом современных объемов сохраняемой информации и размеров носителей. Класс XFS представлен следующими тремя файловыми системами:

- файловая система XFS RMFAT (Removable Media FAT), специально разработанная для переносных носителей, она практически полностью основана на FAT-системах с небольшими включениями из NTFS;
- совмещенная файловая система XFS FAT 32/64 для жестких дисков различных интерфейсов с поддержкой объемов, недоступных ни одной из имеющихся FAT-систем, представляющая собой, наверное, предельный гибрид FAT и NTFS;
- файловая система XFS IFS (Indexed File System) проектируется на основе NTFS, но с удаленными из него устаревшими и ненужными атрибутами и с добавлением некоторых элементов из FAT.

Все файловые системы имеют некоторую самую общую часть, выделенную из родительских при их анализе. Известно [9], что в Boot-секторе имеется структура BIOS Parameter Block (BPB), описывающая основные физические и логические характеристики раздела диска.

Общая схема раздела с ФС XFS представлена на рисунке 1.

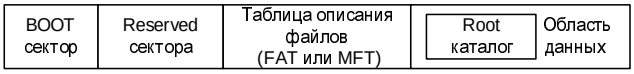

Рисунок 1 – Общая схема раздела с ФС класса XFS

Первой основной особенность реализации ФС класса XFS является их специализация под конкретные типы носителей: RMFAT – под переносные носители; FAT 32/64 – для «среднего надежного» носителя, проектировалась в основном для жестких дисков больших объемов, однако может быть использована и на объемных переносных носителях; IFS – для носителей просто огромных объемов и является наиболее надежной ФС из всего семейства. Второй особенностью является полная поддержка Unicode всеми ФС класса. Также, благодаря заявленной специализации каждой ФС возможно написание оптимизированных под конкретные типы носителей драйверов.

Файловая система XFS RMFAT предназначена для переносных медиаустройств, однако может использоваться для разделов HDD, при этом максимальный стандартный объем раздела равен ~ 2^32 секторов * 512 байт/сектор (стандартный) = 2 ТБ, при этом у самого стандартного диска объем должен быть не более чем (2^32+2^32) секторов * 512 байт/сектор (стандартный) = 4 ТБ. Общая структура ФС показана на рисунке 2.

Рисунок 2 – Общее содержание диска/раздела диска с ФС XFS RMFAT

Основной особенностью данной ФС является то, что в ней может использоваться одна из трех FAT-таблиц: FAT12 с 12 байтами на номер кластера, FAT16 с 16 байтами (словом) на номер кластера и FAT32 с 32 байтами (двойным словом) на номер кластера.

Инфосектор – специальный сектор ФС, предназначенный для хранения некритических констант и значений раздела, которые, могут быть рассчитаны по информации из boot-сектора и FAT-таблицы, однако добавляют гибкости, скорости и надежности ФС. Отличия его от такого же в MS FAT32 – наличие поля количества кластеров lTotalClustersCount и обязательность всех полей при наличии инфосектора.

Boot Mirror-сектора (эта область известна в MS FAT как Backup) хранят копию всех секторов от boot-сектора до инфосектора включительно, что позволяет восстановить ФС после «сбоя нулевой дорожки».

Журнальный сектор содержит лог последних совершаемых транзакций. Также в этой системе возможно создание загрузочного диска. Т.к. корневой каталог может размещаться где угодно, то в первых кластерах размещается загрузчик, после которого идет корневой каталог, в котором описывается файл этого загрузчика. Каталоги RMFAT, как и MS FAT, содержат записи о файлах, однако записи имеют другую структуру, связано это, прежде всего с тем, что имена всех файлов записаны в Unicode и имеют до 255 символов в имени с поддержкой ДОС-имен.

Структура элемента каталога: состоит из двух и более 32-байтных блоков записей, каждый блок имеет тип, поэтому в начале каждой записи имеется обязательное для всех поле типа записи, обозначающее, какие данные следуют далее.

У элементов каталога, которые описывают файл, имеется одна запись стандартной информации, одна или несколько записей имени в Unicode и может быть запись имени в DOS-кодировки. Следует отметить, что DOS-имя подчиняется тем же правилам, что и в MS FAT, т.е. из 11 символов первые 8 – имя файла, если оно короче, то дополняется пробелами справа до 8 символов, следующие 3 – расширение, точка подразумевается. Также, все структуры имени файла имеют поле порядкового номера записи имени, что увеличивает надежность и восстанавливаемость файловой системы.

Файловая система XFS FAT 32/64 специально проектировалась, как гибрид FAT и NTFS в попытке соединить скорость работы FAT и надежность хранения данных NTFS. В результате от FAT была оставлена самая главная ее часть – FAT, работающая быстрее локальных кластерных списков и битовой карты NTFS, все остальное так или иначе было взято из NTFS. В структуру каталогов, в общем, и в корневой каталог в частности было внесено подобие элементов MFT, хотя, как таковой, MFT нет. На рисунке 3 показано фиксированное содержимое корневого каталога. Из рисунков видно, что boot–сектор остался, однако имеет отличную от RMFAT структуру. Следует также отметить, что были оставлены потоки и резидентные/нерезидентные атрибуты, описания которых теперь введены в структуру элементов каталогов.

$BOOT	Boot-сектор
$BOOTMIRROR	Копия boot-сектора
$FAT	Содержит описание FAT-таблиц
$JOURNAL	Файл журнала
$QUOTA	Информация о дисковых квотах
$SECURE	Информация о защите
$VOLDEFATTR	Стандартные атрибуты тома
$VOLUME	Содержит файл тома (с описанием тома)
.	Описание самого себя
..	Родительский каталог – ссылается на самого себя
[...]	Остальные записи

Рисунок 3 – Фиксированное содержание корневого каталога FAT 32/64

Файловая система XFS IFS спроектирована полностью на основе NTFS, однако, из нее убраны недостатки и устаревшие элементы последней, связанные в основное с API ФС, например, заполнение файлами раздела происходит по другому алгоритму, не благоприятствующему фрагментации, также убраны ограничения дефрагментации [10].

Для анализа эффективности разработанных ФС составлена сравнительная таблица 1.

Модификация загрузчика операционной системы

При написании загрузчика основной целью было обеспечить возможность загрузки им ОС с любой файловой системы без необходимости переписывание его под каждую ФС. Поэтому порядок начальной загрузки был изменен по сравнению с загрузчиками Windows и ReactOS кардинальным образом. По результатам анализа дизассемблерных листингов загрузчика Windows (osloader.exe) и исходных текстов загрузчика ReactOS (freeldr.sys) установлено, что они несут в себе код начальной работы с файловой системой. Для достижения поставленной задачи загрузчик XoS был разделен на 2 части – низкоуровневый (low-level) файл-драйвер ФС и, собственно, весь остальной загрузчик. В таблице 2 показаны этапы загрузки всех трех ОС. Из таблицы видна особенность последнего – для обмена информацией между двумя его частями используется механизм программных прерываний и 255-ый вектор прерываний. На рисунке 4 дано описание функции этого прерывания, используемых для взаимодействия частей.

```
;                  255 вектор прерывания (int 0FFh)
; Этот вектор используется XoS как вектор для установки всех, подгружаемых
; процедур РЕАЛЬНОГО РЕЖИМА. Он актуален ТОЛЬКО ДО перехода в PM.
; После перехода в PM основной загрузчик перехватывает вектор на свою
; PM-процедуру и перенастраивает RM-функции на свои PM
; --------------------------- Функции int 255 ----------------------------
;  Общие функции (10 векторов функций):
;   • int 0FFh 000h - вывод строки на экран со 2-й позиции
;   • int 0FFh 001h - чтение секторов (для данной ФС)
;   • int 0FFh 002h - свободно (может быть определено loader'ом файловой
;       системы для функции чтения секторов в режиме LBA (2й вариант))
;   • int 0FFh 003h - запись секторов (для данной ФС)
;   • int 0FFh 004h - свободно (может быть определено loader'ом файловой
;       системы для функции записи секторов в режиме LBA (2й вариант))
;   • int 0FFh 005h - функция htoa - перевод DWord'a из числовой формы в
;       символьную строку
;   • int 0FFh 006h - свободно
;   • int 0FFh 007h - подсчет длины строки (в символах)
;   • int 0FFh 008h - сравнение строк
;   • int 0FFh 009h - конкатенация строк
;   • int 0FFh 00900h - конкатенация MultiByte (1-байтовых) строк
;   • int 0FFh 00901h - конкатенация Unicode (2-байтовых) строк
;   • int 0FFh 00Ah - выделить блок памяти в базовой памяти
;       Выделяется блок памяти в 1-м Мб памяти, начало и размер блока
;       выровнены на страницу [PAGE] (256 байт)
;  [• int 0FFh 00Bh - изменить размер блока памяти в базовой памяти    ]
;  [   Если возможно, будет изменен размер блока памяти в 1-м Мб памяти]
;   • int 0FFh 00Ch - освободить блок памяти в базовой памяти (1-й Мбайт)
;  [• int 0FFh 00Dh - выделить блок памяти в расширенной памяти (сверх]
;  [   первого мегабайта)                                              ]
;  [   Будет выделен блок памяти в расширенной памяти (сверх 1-го Мб).]
;  [   Начало и размер блока кратны 4Кб [MEMPAGE]                      ]
;  [• int 0FFh 00Eh - изменение размера блока памяти в расширенной памяти]
;  [   Если возможно, будет изменен размер памяти в расширенной памяти   ]
;  [• int 0FFh 00Fh - освободить блок памяти в расширенной памяти]
; Функции работы с файлами:
;  [ ставятся лоадером файловой системы ]
;   • int 0FFh 010h - установка текущего каталога
;  [• int 0FFh 011h - создание каталога]
;  [• int 0FFh 012h - удаление каталога]
;   • int 0FFh 013h - [создание/]открытие файла
;   • int 0FFh 014h - закрытие файла
;   • int 0FFh 015h - установка/получение позиции чтения/записи
;   • int 0FFh 016h - чтение файла
;  [• int 0FFh 017h - запись файла]
;  [• int 0FFh 018h - удаление файла]
;  [• int 0FFh 019h - работа с атрибутами]
;   • int 0FFh 01Ah - получение размера файла
;  [• int 0FFh 01Ch - поиск первого файла]
;  [• int 0FFh 01Dh - поиск следующего файла]
;   • int 0FFh 01Eh - функция инициализации; возвращает адрес начала и
;     размер, занимаемый файлом лоадера ФС в памяти
;   • int 0FFh 01Fh - функция размонтирования ФС; освобождает свою память
```

Рисунок 4 - Функции 255 прерывания этапа загрузки XoS

Таблица 1. Сравнение файловых систем

Наименование	Microsoft		XFS		
	FAT	NTFS	RMFAT	FAT 32/64	IFS
Поддержка Unicode	Только FAT32, полная	Полная, родная	Полная, родная	Полная, родная	Полная, родная
Максимальный размер раздела	FAT12– 32 Мб FAT16– 2 Гб FAT32– 2 Тб	16 ЭБ	2^32 кластеров	FAT32 – 2^32 кл. FAT64 – 2^64 кл.	2^64 кл.
Метод описания положения файлов	FAT	Цепочки кластеров в атрибуте $DATA	FAT	FAT	Цепочки кластеров в атрибуте $DATA
Возможные атрибуты файла	Базовый набор	Любые, включая базовые	Базовый набор	Любые, включая базовые	Любые, включая базовые
Устойчивость к сбоям	FAT12/16 – средняя FAT32 – Плохая	Полная	Хорошая	Отличная	Полная
Экономичность	FAT12/16 – минимальна FAT32 – улучшена	Макс.	средняя, возможен подбор параметров	Макс.	Макс.
Быстродействие	$\frac{1}{N_{files}}$	Идеальна для больших массивов данных	Идеальна для переносных носителей, $\frac{1}{N_{files}}$	Идеальна для больших массивов данных	Идеальна для больших массивов данных
Максимальная длина имени файла	255 символов с поддержкой LFN	255	255	255	255
Максимальная длина пути файла	Не огр.	32 767 символов Unicode	Не огр.	Не опред.	32 767 символов Unicode
Максимальный размер файла	FAT12– 32 Мб FAT16– 2 Гб FAT32– 4 Гб	16 Эб	4 Гб	Не опред.	16 Эб
Создание временных меток	Да	Да	Да	Да	Да
Временные метки изменения метаданных	Нет	Да	Нет	Да	Да
Расширенные атрибуты	Нет	Да	Нет	Да	Да
Жесткие ссылки	Нет	Да	Нет	Да	Да
Журналирование	Нет	Да	RMFAT12 – нет, RMFAT16 – возможно, RMFAT32 – да	Да	Да

Таблица 2 – Этапы загрузки ОС класса Windows ядра WinNT

Windows XP (NTFS)	ReactOS (FAT32)	XoS (RMFAT)
BIOS загружает Boot–сектор		
Boot-сектор загружает вторичный загрузчик, расположенный в первых 16 секторах раздела, используя для определения начала раздела поля из BPB	Boot-сектор загружает вторичный загрузчик, расположенный в 14 секторе раздела, используя для определения начала раздела поля из BPB	Boot-сектор инициализирует таблицу 255 прерывания и устанавливает в нее адреса внутренних функций (одна из которых – функция чтения секторов), загружает low-level файл-драйвер xosldr текущей ФС, являющийся по совместительству полным загрузчиком, расположенный в первых кластерах ФС и имеющий известный boot-сектору размер, устанавливает свой обработчик int 255 и передает управление драйверу
Вторичный загрузчик умеет работать с логической структурой NTFS и загружает ntldr, расположенный с произвольного кластера, но описанного в MFT\.\ntldr	Вторичный загрузчик умеет работать с логической структурой FAT32 и загружает freeldr.sys, расположенный с произвольного кластера, но описанного в \freeldr.sys	Low-level драйвер ФС (образ ФС) в реальном режиме устанавливает в таблице 255 прерывания адреса на свои функции работы с файловой системой, загружает 16-в-32 битное ядро (по существу, это ядро является эквивалентом загрузчика osloader и freeldr за исключением того, что в нем отсутствует код работы с ФС) и передает управление последнему
Ntldr несет в себе в конце osloader.exe (сам по себе ntldr занимает до 4 Кб). Это очень удобно, т.к. известно физическое местоположение последнего и оно фиксировано, также отпадает необходимость работать с диском. ntldr переводит систему в защищенный страничный режим, проецирует osloader.exe в виртуальное пространство и передает ему управление; GDT, IDT, таблицы и каталоги страниц имеют фиксированный размер и местоположение и спроецированы на небольшой участок памяти	freeldr является монолитным загрузчиком, он переводит систему в защищенный режим, производит инициализацию внутренних менеджеров памяти (1,5 в данной версии), загружает и интерпретирует boot.ini, загружает куст реестра system, строит текущую конфигурацию оборудования, инициализирует системные структуры, загружает ядро, за ним – драйверы типа Boot, создает каталог и таблицы страниц нулевого потока, GDT и IDT ОС, переходит в страничный режим и отдает управление ядру	16-в-32-битное ядро проверяет соответствие аппаратного обеспечения необходимым требованиям, переходит в защищенный режим, инициализирует менеджеры памяти (2,5; при инициализации менеджера первого мегабайта будет вызвана функция образа ФС для определения занимаемой им памяти), загружает и интерпретирует boot.ini, загружает ядро, инициализирует системные структуры, создает каталог и таблицы страниц нулевого потока, GDT и IDT ОС, переходит в страничный режим и отдает управление ядру
Osloader.exe производит инициализацию внутренних менеджеров памяти, загружает и интерпретирует boot.ini, загружает куст реестра system, строит текущую конфигурацию оборудования, создает полные каталог и таблицы страниц 0 потока, GDT и IDT ОС, загружает ядро, за ним – драйверы типа Boot, инициализирует системные структуры, перенастраивает все глобальные таблицы данного этапа и отдает управление основному ядру ОС		

Из рисунка 4 видно, что функции 10h-1Fh ставятся образом ФС и обеспечивают полную функциональность для работы с ФС, а функции 0Ah-0Fh ставятся 16-в-32-битным ядром и обеспечивают образ ФС необходимыми функциональностью для

работы с памятью. В результате имеется универсальный интерфейс взаимодействия, позволяющий ОС загрузится с любой ФС с малейшими изменениями кода загрузчика, при этом образ ФС ничего не знает об устройстве менеджера памяти ядра, а последнее, в свою очередь, ничего не знает о внутреннем устройстве ФС, с которой оно грузится, т.к. далее этим делом будет заведовать полноценный драйвер ФС. Еще одно преимущество заключается в том, что все 16-битные функции могут быть заменены 16-в-32-битным ядром на 32-битные без переписывания образа ФС. И последнее, эти функции можно использовать для дампа памяти при аварийной ситуации в ОС.

Low-level драйвер ФС (образ ФС) состоит из двух частей – 16-битной части загрузчика 16-в-32-битного ядра и 32-битной части вызываемых функций 255 прерывания – и трех блоков данных после образа, каждый блок имеет размер в сектор диска. Первые два блока используются как буфер для двух секторов FAT-таблицы (текущий обрабатываемый и следующий за ним), последний – буфер для данных. 16-битная часть производит проверку ФС, инициализацию внутренней структуры, установку своих функций работы с ФС в таблицу функций 255-го прерывания (которая имеет фиксированный адрес – она начинается сразу за Boot-сектором), загрузку и старт 16-в-32-битного ядра.

Особенности построения 16-в-32 битного ядра XoS

16-в-32-битное ядро xoskrnrm (являющееся, по существу, основным загрузчиком XoS) состоит из нескольких модулей: код инициализации, менеджеры памяти, менеджер файловой системы, менеджер файлов инициализации, загрузчик PE-файлов, интерфейса пользователя и высокоуровневого видеоинтерфейса, общего кода загрузки системы, уровня абстракции от архитектуры и прерываний, а также библиотечных функций.

Код инициализации предназначен для тестирования железа на наличие необходимой функциональности, настройки первичных структур 16-в-32-битного ядра и перехода в защищенный нестраничный режим. Стоит особо отметить, что до определения необходимого процессора идет код, поддерживаемый всеми процессорами Intel 8086. Алгоритм работы:

1. Запрет прерываний, начальная инициализация регистров и установка функций 05h-0Ah;
2. Определение процессора путем его прямого тестирование (не через BIOS), необходимый процессор – Pentium+; также определяется частота процессора;
3. Открытие шину A20 и определение размера памяти через BIOS (на текущий момент – int 015h 0E801h), минимальный объем – 8 Мб;
4. Определение видеоадаптера, минимум VGA;
5. Переход в PM и вызов функции PM_Start;
6. Функция PM_Start производит вызов функций инициализации менеджеров памяти и функции запуска 32-битного кода RunLoader.
7. Если вернулись из PM_Start, то подготовка и переход в 16-битный режим и вывод сообщения о необходимости перезагрузки.

Функционально имеется 2 менеджера памяти, это связано с тем, что первый МБ памяти (базовая память) имеет особое назначение и карту для системы, а также используется на 255-ом прерывании в функциях выделения базовой памяти, и его память распределяет менеджер цепочечных битовых карт. Всю остальную память распределяет менеджер табличного постраничного описания с нефиксированным началом таблицы.

Имеются функции для захвата и освобождения памяти:

- для менеджера первого МБ эти функции являются функциями 255-го прерывания + функция инициализации;
- для менеджера расширенной памяти это несколько внешних функций выделения и освобождения памяти и внутренние функции работы с таблицей распределения памяти.

Следует особо отметить, что менеджер расширенной памяти, во-первых, запирает первый МБ страниц, устанавливая их в режим KernelFirmwareTemporary, а во-вторых, для выделения памяти меньше страницы использует одну и ту же страницу, пока она не закончится, потом захватывает следующую для этих целей, «забывая» предыдущую, т.е. освобождение малых объемов памяти на самом деле не происходит.

Менеджер файловой системы представляет собой удобный набор функций-переходников типа stdcall на функции образа ФС 255-го прерывания. Это функции открытия файла по его имени, закрытия файла по его дескриптору, чтения файла, получение его размера и установки позиции чтения/записи.

Менеджер файлов инициализации содержит необходимые функции для работы с файлами инициализации .ini, включающие внешние функцию начального парсинга файла (разбиение его на секции и параметры и преобразование этой информации в структуры списков секций и списков параметров каждой секции) и работы с парсированным файлом (функция открытия секции и чтения параметра из секции по имени). На рисунке 5 показаны все используемые структуры менеджера и их взаимосвязь.

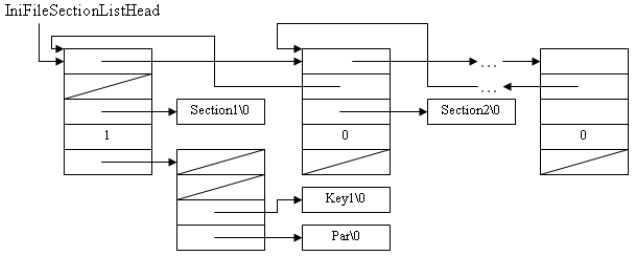

Рисунок 5 – Структуры менеджера файлов инициализации и их взаимосвязь

Загрузчик PE-файлов состоит из 2^x частей: это код загрузки PE 16-в-32-битного ядра и общий Run-Time Library (Rtl) код работы с образом PE. Rtl-функции работы с образом PE – это в основном функции разбора структур заголовка PE. Код загрузки PE включает в себя функции загрузки PE-файла и его разбора, загрузки импортируемых библиотек, заполнение таблиц импорта, поиска образа в списке загрузки и разбор блоков размещения [11]. Следует отметить, что функция загрузки PE-файла грузит его в защищенном режиме при выключенной страничной адресации, но файл оказывается готов для использования после включения страничной адресации.

Высокоуровневый видеоинтерфейс представлен функциями для работы с двойным буфером в памяти: создание буфера, копирование в экранную память, вывод символов и строк в буфер, а также функции работы с видеорежимами.

Интерфейс пользователя включает в себя общий интерфейс пользователя и является интерфейсом между всеми остальными частями 16-в-32-битного ядра и текстовый интерфейс пользователя (те же функции, но в текстовом режиме и несколько внутренних функций типа заливки фона символами).

Оба этих интерфейса предоставляют все необходимые функции для отображения хода загрузки XoS на экране.

Общий код загрузки системы включает в себя бут-менеджер (функция RunLoader), загрузчик ядра и код подготовки основных структур ядра. Алгоритм работы:

1. Устанавливается свой перехватчик 255-го прерывания для функций, для которых необходим переход в 16-битный режим;
2. Производится анализ файла \boot.ini и из секции [boot loader] считывается значение ключа default, содержащего системный корневой каталог XoS;
3. Вызывается функция загрузки XoS LoadAndBootXoS;
4. Инициализируется интерфейс пользователя и на экране отображается процесс загрузки;
5. Создается структура LOADER_PARAMETER_BLOCK и производится ее начальная инициализация;
6. Из каталога SystemRoot\system32\ загружаются xoskrnl.exe и hal.dll и для них создаются структуры LDR_DATA_TABLE_ENTRY, причем для файла ядра xoskrnl.exe создаются 2 структуры – одна из них описывает xoskrnl.exe, а вторая является ее копией, но описывает ntoskrnl.exe (в результате в системе существует как бы два имени файла ядра, что необходимо для полной низкоуровневой совместимости с WinNT) и разбирается импорт с подгрузкой всех импортируемых модулей;
7. Выполняется первая фаза инициализации, состоящая в создании и заполнении описательных структур, связанных со структурой LOADER_PARAMETER_BLOCK и ее самой, и переводом всех имеющихся в ней списков из физической адресации в виртуальную в пространстве ядра (нижние 2 Гб);
8. Выполняется подготовка системных таблиц ядра к заполнению (PCR, GDT, IDT, TSS) и инициализация оставшихся полей LOADER_PARAMETER_BLOCK;
9. Вызывается функция включения страничного режима XoSLdrTurnOnPaging;
10. Выполняется создание и начальная инициализация PDE, таблиц PTE и HAL-региона (PDE заполняется указателями на собственную область памяти и область памяти таблицы HAL-региона);
11. Выполняется цикл преобразования таблицы описания памяти табличного менеджера памяти 16-в-32-битного ядра в структуры описания памяти структурного менеджера памяти 32-битного ядра с одновременным слиянием одинаковых соседних регионов памяти, заполнением таблиц PTE и их занесением в PDE (схема преобразования приведена на рисунке 6);
12. В таблицу HAL-региона заносится описание областей памяти структур KUSER_SHARED_DATA (KI_USER_SHARED_DATA) и KIPCR (KIP0PCRADDRESS);
13. Все оставшиеся адреса в LOADER_PARAMETER_BLOCK'е преобразуются в виртуальные;
14. Происходит инициализация контекста процессора (заполнение структур GDT, IDT и сегментных регистров);
15. Происходит переход в 32-битное ядро xoskrnl.exe.

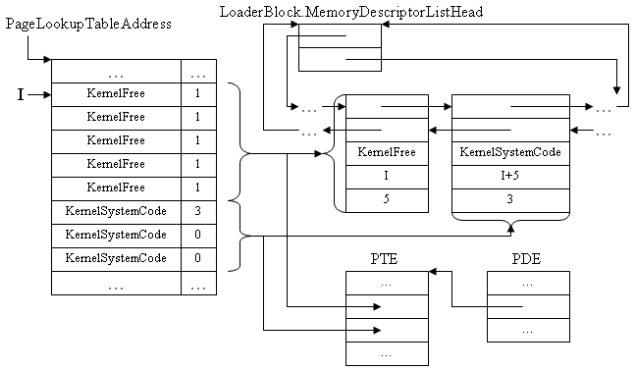

Рисунок 6 – Схема преобразования таблицы описания памяти

Уровень абстракции от архитектуры (ALA, Architecture Abstraction Level) – это набор групп функций, также как и HAL, скрывающих детали функционирования платформы, на которой

запускается система, от всех остальных частей, но в пределах 16-в-32-битного ядра и инкапсуляции работы с прерываниями. Это группы функций общей работы с железом, с консолью, низкоуровневой работы с видео и портами. Следует отметить, что в отличие от загрузчика osloader в WinNT и freeloader в ReactOS, где вызов этих функций осуществляется через заполняемую на этапе выполнения таблицу адресов, в xoskrnrm происходит связывание на этапе компиляции с использованием информации из заголовочных файлов, что несколько ускоряет исполнение, уменьшает размер кода и упрощает сам код.

Библиотечные функции включают в себя Run-Time Library (Rtl) функции (работа со списками и образами PE-файлов) и Software Development Kit (sdk) функции (работа со строками и участками памяти), специально разработанные для fasm'а.

Самый первый и самый большой плюс предложенного модуля загрузки состоит в том, что весь код загрузчика полностью написан на Ассемблере с применением макросистемы flatassembler, что позволяет разрешать компилятору самому генерировать код (используя макросы). Следующим несомненным плюсом является разделение загрузчика на 2 части – low-level драйвера (образа) ФС и 16-в-32-битного ядра и использования универсального интерфейса взаимодействия между ними – 255 прерывания, что дает возможность запускать XoS с любой файловой системы, не переписывая код всего загрузчика, а только изменяя код образа ФС; это же делает код 16-в-32-битного ядра для платформы IBM PC/AT. Также уменьшилось количество этапов запуска ОС, по сравнению с WinNT. Следует отметить стойкость функции прохода по дереву папок образа ФС RMFAT к нарушениям последовательности записей описания объекта в каталоге – такие нарушения будут просто пропущены с переходом на следующую запись. Также, до определения типа процессора идет код, совместимый со всеми i80x86.

Выводы

Данная работа является первым этапом в процессе исследования ОС класса WinNT и проектирования ОС XoS, полностью совместимой с WinNT-семейством вплоть до используемых структур. Предполагается, что процесс состоит из нескольких этапов.

1. Разработка собственного класса ФС XFS;
2. Разработка и написание загрузчика ядра, состоящего из двух частей – низкоуровневого драйвера ФС (RMFAT) и 16-в-32-битного ядра;
3. Написание 32-битного ядра xoskrnl.exe и hal.dll и необходимой для их работы библиотеки bootvid.dll;
4. Написание основных внешних менеджеров, драйверов и сервисов ядра.
5. Перенос остальных сервисов и драйверов из Windows XP и ReactOS и их замена на собственные;
6. Введение в ядро собственных механизмов работы с памятью, управления драйверами и аппаратурой;
7. Введение зачатков ИИ в ядро;

Направление дальнейших исследований – это построение 32-битной части ядра и соответствующих библиотек ОС.

Литература

1. Таненбаум Э., Вудхалл А. Операционные системы: разработка и реализация (+CD). Классика CS. – СПб.: Питер, 2006. – 576 с.
2. Руссинович М., Соломон Д. Внутреннее устройсво Microsoft Windows: Windows Server 2003, Windows XP и Windows 2000. Мастер-класс. / Пер. с англ. – 4-е изд. – М.: Издательско-торговый дом «Русская Редакция»; СПб.: Питер; 2005. – 992 с.
3. Операционная система ReactOS // www.reactos.org
4. Бозуэлл У. Внутренний мир Windows Server 2003, SP1 и R2, М.: Вильямс, 2006, 1264 с
5. Шапиро Д., Бойс Д. Windows 2000 Server. Библия пользователя, М.: Диалектика, 2001, 912 с.
6. Niranjan T., File System Support For Multimedia Applications, Ph.D. Dissertation, ECSL TR-32, Computer Science Department, SUNY at Stony Brook, December 1996.
7. Rashid R., Baron R., Forin A., Golub D., Jones M., Julin D. P., Orr D. and Sanzi R., A Foundation for Open Systems. Proceedings of the Second IEEE Workshop on Workstation Operating Systems. IEEE Computer Society Press, Computer Society Order Number 2003
8. Hutchinson N. C., Peterson L. L. and Rao H. The x-Kernel: An Open Operating System Design, Proceedings of the Second IEEE Worshop on Workstation Operating Systems, IEEE Computer Society Press, Computer Society Order Number 2003
9. Структура BPB // msdn.microsoft.com/archive/en-us/win9x/fat32_6blc.asp
10. Михайлов Д. Файловая система NTFS // www.iXBT.ru
11. Hard Wisdom. Формат исполняемых файлов Portable Executables // www.wasm.ru