

Формализованная структурная модель операционной системы Windows 2000

77-30569/292997

12, декабрь 2011

Черненко В. М., Шульмин А. С.

УДК 004. 451

МГТУ им. Н.Э. Баумана

chernen@bmstu.ru

Цель статьи - построение формализованной структурной модели операционной системы Windows 2000 - актуальной операционной системы, предназначенной для работы на современной аппаратной базе, поддерживающей новые процессорные архитектуры. Windows 2000 – серверная операционная система, следовательно, особое внимание ее разработчики уделяли обеспечению надежности и безопасности ОС. Однако эта ОС построена таким образом, что расширения ее ядра не изолированы и выполняются как отдельные процессы в режиме ядра, что потенциально создает многие проблемы в области безопасности [3]. Кроме того, Windows 2000, в отличие от более поздних систем от Microsoft, не обладает дополнительными структурными элементами, усложняющими процесс формализации.

Вся информация, изложенная в статье, получена из открытых источников (Microsoft Press [1]), или же добыта самостоятельно в результате реверс-исследования компонентов системы.

В ходе формализации структуры ОС будем пользоваться аппаратом теории множеств. В модели (где возможно) будем использовать англоязычные сокращения или аббревиатуры, совпадающие с названиями смысловых аналогов, используемых Microsoft.

Как было отмечено ранее, при описании модели ОС возьмем за основу ее структурный состав, отложив построение функциональной модели на будущее.

Итак, определим структуру Windows 2000 в целом следующим образом:

$$OSWX = \langle SBS, NTD, EXC, KRN, HAL, DRV, SPR \rangle \quad (1.1)$$

где **SBS** – блок подсистемы окружения, **NTD** – библиотека системной поддержки ntdll.dll, **EXC** – исполнительная система, **KRN** – ядро, **HAL** – уровень абстракций от оборудования, **DRV** – драйверы устройств, **SPR** – системные процессы.

Структурная модель блока подсистем окружения

Прежде чем переходить к непосредственному построению модели блока подсистем окружения отметим, что в современных ОС Windows, сделанных на базе технологий NT, реализована только одна подсистема окружения – Windows [1], однако для точности модели мы рассмотрим все возможные подсистемы. Формализованная модель блока подсистем окружения представляет собой следующее:

$$SBS = \langle SS, RQ, OP \rangle, \quad (1.2)$$

где **SS** – множество подсистем окружения, **RQ** – подсистемы, загружаемые при старте системы (список), **OP** – подсистемы, запускаемые по требованию (список).

$$SS = \{subs_i\}, \quad (1.3)$$

где **subs_i** – подсистема окружения. Подсистема окружения по сути – совокупность некоторых программных компонентов. Опишем их.

$$subs_i = \langle subs_{NM}, subs_{PC}, subs_{DLL}, subs_{DRV}, subs_T, subs_{DRV_G} \rangle, \quad (1.4)$$

где **subs_{NM}** – наименование подсистемы окружения;

subs_{PC} – процесс подсистемы окружения (исполняемый образ);

subs_{DLL} – динамически подключаемые библиотеки конкретной подсистемы окружения;

$subs_{DRV}$ – драйвер подсистемы окружения;

$subs_T$ – тип подсистемы окружения;

$subs_{DRV_G}$ – драйверы графических устройств.

Для обеспечения выполнения программных модулей, написанных для конкретной подсистемы, подсистема окружения использует набор динамически подключаемых библиотек.

$$subs_{DLL} = \{subs_dll_i\}, \quad (1.5)$$

где **$subs_dll_i$** – DLL подсистемы окружения.

Мы выделяем два типа подсистем окружения – основную и дополнительные. Основная подсистема обрабатывает все, что связано с клавиатурой, мышью, экраном, а также содержит реализацию некоторых API-функций, которых не реализованы в других подсистемах из соображений исключения дублирования кода. Дополнительные подсистемы окружения запускаются по требованию и, в случае необходимости, их процессы обращаются к основной подсистеме окружения.

$$subs_T \in SUBT, \quad SUBT = \{subt_m, subt_a\}, \quad (1.6)$$

где **$SUBT$** – множество типов подсистем окружения;

$subt_m$ – основная подсистема (Windows);

$subt_a$ – дополнительная подсистема (OS/2, POSIX).

Основная подсистема окружения может содержать в своем составе драйверы графических устройств, а именно: драйверы графического дисплея, драйверы принтеров и минипорт-драйверы видеокарт.

$$subs_{DRV_G} = \langle subs_{GD_DRV}, subs_{GD_T} \rangle, \quad subs_{GD_T} \in SUBGDT, \quad (1.7)$$

$$SUBGDT = \{subgd_d, subgd_p, subgd_v\}$$

где **$subs_{GD_DRV}$** – драйвер графического устройства;

$subs_T$ – тип графического устройства;

$SUBGDT$ – множество типов графических устройств;

$subgd_d$ – драйвер графического дисплея;

subgd_p – драйвер принтера;

subgd_v – минипорт-драйвер видеоадаптера.

Структурная модель библиотеки системной поддержки

Ntdll.dll – специальная библиотека системной поддержки Windows. Она содержит в себе следующие компоненты: интерфейсы диспетчера системных сервисов к сервисам исполнительной системы, внутренние функции поддержки, диспетчер куч, функции для взаимодействия с процессом подсистемы Windows, универсальные функции библиотеки времени выполнения, диспетчер исключений и диспетчер асинхронных вызовов процедур [1]. Определим библиотеку системной поддержки следующим образом:

$$NTD = \langle DSS, ISF, HDF, CRS, RTL, DEX, APC \rangle, \quad (1.8)$$

где **DSS** – функции интерфейса диспетчера системных сервисов;

ISF – внутренние функции поддержки;

HDF – функции диспетчера куч;

CRS – функции для взаимодействия с процессом основной подсистемы;

RTL – универсальные функции библиотеки времени выполнения;

DEX – функции диспетчера исключений;

APC – функции диспетчера прерываний APC (Asynchronous procedure call).

Структурная модель исполнительной системы

Исполнительная система находится на верхнем уровне ntoskrnl.exe и обеспечивает выполнение основных функций ОС.

Мы считаем целесообразным построить две модели исполнительной системы (ИС). В первой модели отразим функциональный состав ИС, во второй – структурную организацию ИС.

Итак, с позиций функционального набора, исполнительная система может быть описана следующим образом:

$$EXC = \langle SSF, DDF, KSF, GNE, ING \rangle, \quad (1.9)$$

где **SSF** – экспортируемые исполнительной системой функции, доступные для вызова из пользовательского режима;

DDF – функции драйверов устройств, вызываемые через DeviceIOControl;

KSF – экспортируемые функции, доступные для вызова только из режима ядра;

GNE – функции, определенные как глобальные, но не являющиеся экспортируемыми;

ING – внутренние функции в каком-либо модуле, не определенные как глобальные.

Для описания структуры исполнительной системы будем использовать следующую модель:

$$EXC = \langle DCONF, DPT, MNP, DIO, DPP, DEP, PWMI, DC, DMEM, TLP, DOBJ, TLPC, SF, SEXC \rangle, \quad (1.10)$$

где **DCONF** – диспетчер конфигурации, отвечающий за реализацию и управление системным реестром;

DPT – диспетчер процессов и потоков (управление потоками и процессами реализовано в ядре, в исполнительной системе — только некоторые дополнительные функции);

MNP– монитор состояния защиты, реализующий политики безопасности на локальном компьютере;

DIO – диспетчер ввода-вывода, реализующий аппаратный ввод-вывод и отвечающий за перенаправление ввода-вывода нужным драйверам устройств;

DPP – диспетчер Plug and Play, определяющий, какие драйверы нужны для конкретного устройства, и загружающий их. Также распределяет ресурсы для каждого устройства и оповещает систему об изменениях в аппаратном обеспечении системы.

DEP – диспетчер электропитания. Контролирует изменение энергопотребления отдельных устройств и посылает соответствующие уведомления драйверам устройств.

PWMI – подпрограммы WDM Windows Management Instrumentation, позволяющие драйверам публиковать информацию о своих характеристиках и о своей конфигурации, а также получать команды от службы WMI пользовательского режима;

DC – диспетчер кэша, повышающий производительность файлового ввода-вывода, используя поддержку проецируемых файлов со стороны диспетчера памяти;

DMEM – диспетчер памяти, реализующий виртуальную память, обрабатывающий исключения PageFault и обеспечивающий низкоуровневую поддержку диспетчера кэша;

TLP – средство логической предвыборки, ускоряющее старт системы и процессов за счет оптимизации загрузки данных, к которым происходит обращение при запуске системы или процессов.

Кроме этих компонентов в состав исполнительной системы также входят четыре основных блока функций поддержки, которые используются названными выше компонентами.

DOBJ – диспетчер объектов. Создает, управляет и удаляет объекты и абстрактные типы данных исполнительной системы, используемые для представления процессов, потоков и синхронизирующих объектов.

TLPC – механизм LPC (Local Procedure Call), передающий сообщения между клиентскими и серверными процессами на одном компьютере; оптимизированный аналог RPC;

SF – блок стандартных библиотечных функций (обработка строк, арифметические операции, преобразование типов данных, обработка структур безопасности);

SEXC – подпрограммы поддержки исполнительной системы. Предназначены для выделения памяти, доступа к памяти со взаимоблокировкой; также эти подпрограммы содержат два специальных типа синхронизирующих объектов: ресурс и быстрый мьютекс.

Структурная модель ядра операционной системы

Ядро ОС Windows 2000 состоит из набора функций в `ntoskrnl.exe`, представляющих фундаментальные механизмы (в т.ч. планирования потоков и синхронизация), которые используются исполнительной системой и низкоуровневыми средствами абстрагирования от оборудования. Ядро не участвует в принятии решений, связанных с системной политикой, оно лишь реализует системные механизмы для других компонентов системы.

Ядро ОС состоит из т.н. «объектов ядра», которые представляют собой набор простых объектов для контроля обработки данных процессором. По сути, объекты исполнительной системы, как правило, инкапсулируют в себе один или несколько объектов ядра.

Формально определим ядро операционной системы следующим образом:

$$KRN = \langle OBJs, INTR, SINT \rangle, \quad (1.11)$$

где ***OBJs*** – объекты ядра (см. выше);

INTR – семантически идентичные и переносимые между архитектурами интерфейсы (для обеспечения абстрагирования от оборудования);

SINT – x86-специфичные интерфейсы для поддержки старых программ MS-DOS.

$$OBJs = \{objs, objs_T\}, \quad objs_T \in OBJ_T, \quad (1.12)$$

где ***objs*** – собственно объекты ядра (данные и код);

objs_T – класс объекта ядра;

OBJ_T – множество классов объектов ядра.

$$OBJ_T = \{c_obj, d_obj\}, \quad (1.13)$$

где

c_obj – управляющий объект (control object);

d_obj – объект диспетчера (dispatcher object).

«Управляющий объект» и «объект диспетчера» – это два больших класса объектов, включающих в себя заранее известное множество типов объектов.

Опишем каждый класс.

$$c_obj \in C_OBJ_T, C_OBJ_T = \{apco, dpco, dioo\} \quad (1.14)$$

где

apco – APC-объекты;

dpco –DPC-объекты (DPC – Deferred Procedure Call);

dioo – объекты, используемые диспетчером ввода-вывода.

$$d_obj \in D_OBJ_T, D_OBJ_T = \{m_o, e_o, s_o, t_o, wt_o\} \quad (1.15)$$

где

m_o – мьютекс;

e_o – событие;

s_o – семафор;

t_o – таймер;

wt_o – ожидаемый таймер.

Структурная модель уровня абстрагирования от оборудования

Одна из важнейших особенностей ОС Windows 2000 – переносимость между платформами. Ключевой компонент, обеспечивающий такую переносимость, – уровень абстрагирования от оборудования – HAL [1, 2, 4]. HAL – загружаемый модуль режима ядра (hal.dll), предоставляющий низкоуровневый интерфейс с аппаратной платформой. Все остальные компоненты системы обращаются к оборудованию только через HAL. Переносимость достигается путем использования различных слоев абстрагирования от оборудования. В HAL содержатся системные функции и экспортируемые символы, используемые драйверами. Построим модель HAL следующим образом:

$$HAL = \langle HF, HS \rangle, HF = \{hf_i\} \quad (1.16)$$

где

HF – набор функций HAL;

HS – экспортируемые символы HAL;

hf_i – функции HAL.

$$hf_i = \{hf, hf_T\}, \quad (1.17)$$

где

hf – собственно функция HAL;

hf_T – тип функции.

$$hf_T \in HFT, \quad HFT = \{fx, fh, fk, fio, fs\} \quad (1.18)$$

где

HFT – множество типов функций HAL;

fx – функция исполнительной системы, реализованная в HAL;

fh – функция HAL, предназначенная для сокрытия аппаратных различий;

fk – функция ядра ОС, перенесенная в HAL;

fio – функция диспетчера ввода-вывода, перенесенная в HAL;

fs – специальная функция HAL (например, макрокоманды для записи/чтения в/из портов).

Структурная модель драйверов устройств

Драйверы устройств в Windows – это бинарные программы, выполняющиеся в нулевом кольце привилегий. Работой драйвера управляет подсистема ввода-вывода ОС. Модель драйвера устройства представим следующим образом:

$$DRV = \{drv_i\}, \quad drv_i = \{drv, drv_T, drv_{TR}, drv_{WDM}, drv_{ML}\}, \quad (1.19)$$

$$drv_T \in DRV_T, \quad drv_{WDM} \in DRV_WDM = \{0,1\}, \quad drv_{ML} \in DRV_ML = \{0,1\},$$

где

drv_i – драйвер;

drv – файл драйвера (бинарный код);

drv_T – тип драйвера;

drv_{TR} – режим работы драйвера (рассмотрено далее);

drv_{WDM} – соответствие драйвера модели WDM (да или нет);

drv_{ML} – признак того, что драйвер является многоуровневым (аналогично);

DRV_T – множество типов драйверов.

В ОС Windows драйверы могут выполняться в контексте пользовательского процесса, инициировавшего процедуру ввода-вывода, в контексте системного процесса режима ядра и как результат прерывания (а значит, не в контексте какого-либо процесса или потока). Следовательно:

$$drv_{TR} \in DRV_TR = \{ring0, ring3, int\}, \quad (1.20)$$

где

DRV_{TR} – множество режимов работы драйвера;

ring0 – драйвер предназначен для работы в контексте системного процесса;

ring3 – драйвер предназначен для работы в контексте пользовательского процесса;

int – драйвер предназначен для работы в качестве обработчика прерывания.

ОС Windows работает с драйверами следующих типов [1]:

$$DRV_T = \{vd_d, p_d, fs_d, pp_d, up_d, b_d, fn_d, fl_d, dc_d, p_d, mp_d\}, \quad (1.21)$$

где

vd_d – драйверы виртуального устройства (virtual device driver, VDD), используются для эмуляции 16-разрядных программ MS-DOS. Эти драйверы позволяют программам MS-DOS обращаться к системным структурам и портам ввода-вывода. Драйверы этого типа выполняются в пользовательском режиме.

p_d – драйверы принтера. Транслируют аппаратно-независимые запросы к графическому устройству в команды для конкретного принтера. Затем эти команды направляются непосредственно драйверу принтера, работающему в режиме ядра, или USB-шине. Драйверы этого типа выполняются в пользовательском режиме.

fs_d – драйверы файловой системы. Принимают запросы на ввод-вывод и формируют более специфические запросы драйверам устройств.

pp_d – Plug and Play драйверы. В их число входят драйверы для устройств массовой памяти, видеоадаптеров, устройств ввода и сетевых адаптеров.

up_d – драйверы, не отвечающие спецификации Plug and Play – расширения ядра. Расширяют функциональность системы, предоставляя доступ из пользовательского режима к сервисам и драйверам режима ядра.

b_d – Драйверы шин. Управляют логическими или физическими шинами. Отвечают за распознавание новых устройств и за управление электропитанием шины.

fn_d – Функциональные драйверы. Управляют конкретным типом устройств. Экспортируют рабочий интерфейс операционной системе.

fl_d – Драйверы-фильтры. Дополняют или изменяют функциональность устройства или другого драйвера.

dc_d – Драйверы класса устройств. Реализуют обработку ввода-вывода для целого класса устройств. Используются для классов устройств со стандартизированными аппаратными интерфейсами.

p_d – порт-драйверы. Обработывают запросы на ввод-вывод для определенного типа порта ввода-вывода.

mp_d – минипорт-драйверы. Преобразуют универсальные запросы ввода-вывода к порту конкретного типа в запросы, специфичные для адаптера конкретного типа.

Процедурный состав программного модуля драйвера следующий:

$$drv = \langle ad_p, \{d_p_i\}, io_p, hi_p, dpc_p, i_p \rangle \quad (1.22)$$

где

ad_p – процедура добавления устройства (реализуются только в PnP-драйверах; через эту процедуру диспетчер PnP посылает уведомление драйверу при обнаружении в системе устройства соответствующего типа);

{d_p_i} – процедуры диспетчеризации (основные функции, предоставляемые драйвером устройства);

io_p – процедура инициации ввода-вывода (с помощью этой процедуры драйвер может инициировать передачу данных как на устройство, так и считывание данных с него);

hi_p – процедура обслуживания прерываний (когда устройство генерирует прерывание, диспетчер ввода-вывода передает управление этой процедуре);

dpc_p – DPC-процедура (выполняется при обработке прерываний; иницирует завершение текущей операции ввода-вывода и запускает новую из очереди этого устройства);

i_p – инициализирующая процедура (диспетчер ввода-вывода выполняет ее при загрузке драйвера, регистрируя в себе остальные процедуры драйвера и выполняя базовую инициализацию структур драйвера).

Структурная модель системных процессов

В каждой системе Windows 2000 выполняются некоторые служебные процессы. Представим структуру блока системных процессов:

$$SPR = \langle Idle, Sysm, Smss, Csrs, Winl, Serv, Lsas \rangle, \quad (1.23)$$

где

Idle – процесс простоя системы, ОС создает по одному такому процессу для каждого процессора;

Sysm – процесс-носитель системных потоков, выполняющихся в режиме ядра.

Smss – процесс «диспетчер сеансов». Участвует в процессе загрузки системы и запускает процессы подсистем (Csrss.exe и Winlogon.exe).

Csrs – процесс основной подсистемы. Назначение описано выше.

Winl – процесс входа в систему. С помощью библиотеки GINA проводит аутентификацию пользователя, полученные данные отправляет процессу Lsass.exe.

Serv – диспетчер управления сервисами (Services.exe) и создаваемые им дочерние процессы сервисов (например, Svchost.exe).

Lsas – процесс локальной аутентификации. Получает от Winlogon.exe аутентификационные данные пользователя, проводит их проверку. В случае

успеха генерирует объект «маркер доступа», который содержит профиль безопасности пользователя.

В данной статье предпринята попытка построения формализованной структурной модели операционной системы Windows 2000. Здесь описаны основные структурные компоненты системы с указанием их взаимосвязи.

Литература

1. Руссинович, М., Соломон, Д. Внутреннее устройство Microsoft Windows: Windows Server 2003; Windows XP и Windows 2000. Мастер-класс. / Пер. с англ. – 4-е изд. – М.: Издательство-торговый дом «Русская Редакция»; СПб.: Питер, 2005. – 992 с.: ил. ISBN 5-469-01174-7.
2. Таненбаум, Э., Вудхалл, А. Операционные системы: разработка и реализация. Классика CS. – СПб.: Питер, 2006. – 576 с.: ил. ISBN 5-469-00148-2.
3. Bacon J., Harris T. Operating Systems: Concurrent and Distributed Software Design - Addison Wesley, 2003. – 720 p. ISBN 0-321-11789-1
4. Silberschatz, A., Galvin, P., Gagne G. Operating System Concepts, 6th Edition. – John Wiley & Sons Inc., 2002. – 842 p. ISBN 0-471-41743-2.

Formalized structural model of OS Windows 2000

77-30569/292997

12, December 2011

Chernen'kii V.M., Shul'min A.S.

Bauman Moscow State Technical University

chernen@bmstu.ru

The article describes an attempt of creating a formalized structural model of Windows 2000 OS. The authors present structural descriptions which lay no claim to a functional analysis. All the information in the article is obtained from the open sources (Microsoft Press [1]), or was obtained independently as a result of reverse-study of the system components. The authors propose a definition of the OS structure which, in turn, includes descriptions of the structures of its base components - such as the environment subsystem block, system support library, actuator system, operating system kernel, device drivers, system processes. The system basic structural components are given with the indication of their interrelation.

Publications with keywords: [operating system](#), [operating system's structure](#), [formalized description](#), [OS components](#)

Publications with words: [operating system](#), [operating system's structure](#), [formalized description](#), [OS components](#)

Reference

1. Russinovich, M., Solomon, D., Internals Microsoft Windows: Windows Server 2003; Windows XP and Windows 2000. Master class, Moscow, Izdatel'stko-torgovyi dom «Russkaia Redaktsiia», SPb., Piter, 2005, 992 p.
2. Tanenbaum E., Vudkhall A.,
3. Operating Systems: Design and Implementation. Classic CS, SPb., Piter, 2006, 576 p.
4. Bacon J., Harris T. Operating Systems: Concurrent and Distributed Software Design, Addison Wesley, 2003, 720 p.
5. Silberschatz A., Galvin P., Gagne G., Operating System Concepts, 6th Edition, John Wiley & Sons Inc., 2002, 842 p.