Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и радиоэлектроники

Факультет непрерывного и дистанционного обучения

Кафедра информатики

Электронный учебно-методический комплекс  
по дисциплине

Операционные системы и среды

Часть 1

Для студентов специальности

1-31 03 04 Информатика

Минск 2011

# Общие сведения

## Сведения об ЭУМК

Электронный учебно-методический комплекс по дисциплине «Операционные системы и среды» предназначен для студентов специальности «Информатика» вузов, а также может быть использован магистрантами, аспирантами и специалистами соответствующего профиля.

Электронный учебно-методический комплекс составлен на основе рабочей учебной программы по курсу «Операционные системы и среды», утверждённой деканом факультета непрерывного и дистанционного обучения <дата утверждения>, регистрационный № УД 11‑XX‑YY/Р и рабочего учебного плана специальности 1-31 03 04 «Информатика».

**Составители:**

**С. И. Сиротко,** доцент кафедры информатики Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», кандидат физико-математических наук.

**В. В. Шендер**, ассистент кафедры информатики Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», магистр технических наук.

Рассмотрен и рекомендован к изданию на заседании кафедры информатики, протокол № 3 от 26.09.2011.

Одобрен и рекомендован к изданию методической комиссией факультета компьютерных систем и сетей, протокол № \_\_ от \_\_.\_\_.2011.

## Методические рекомендации по изучению дисциплины

В соответствии с учебным планом студенты дистанционной формы обучения специальности «Информатика» изучают курс «Операционные системы и среды» в течение 8-го и 9-го семестров.

Учебным планом по данному курсу предусмотрены 2 части, включающие изучение теоретических вопросов и решение практических задач. Первая часть изучается в течение 8‑го семестра, содержит 2 индивидуальных практические и 2 контрольные работы и заканчивается сдачей зачета, к которому студенты допускаются только при условии выполненных и защищенных индивидуальных практических и контрольных работ.

Рекомендуется изучать курс «Операционные системы и среды» в соответствии с рабочей программой. Сначала необходимо ознакомиться с содержанием курса, затем изучить рекомендуемую литературу, обращая внимание на вопросы, выделенные в рабочей программе, после чего изучить теоретическое изложение курса по приведенным разделам, темам и вопросам, выполнить задачи для решения (выполнения контрольных работ) в соответствии с заданием.

## Рабочая учебная программа

**Учреждение образования**

**«Белорусский государственный университет**

**информатики и радиоэлектроники»**

УТВЕРЖДАЮ

Декан факультета непрерывного и дистанционного обучения

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. М. Бондарик

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2011 г.

Регистрационный № УД-11-23-\_\_\_/р.

**Операционные системы и среды**

Рабочая учебная программа

для специальности 1‑31 03 04

**Информатика**

Факультет **непрерывного и дистанционного обучения**

Кафедра **информатики**

Курс **4, 5**

Части **1, 2**

Контрольные работы **3 работы**

в т.ч.: часть 1 2 работы

часть 2 1 работы

Индивидуальные практические работы: **4 работы**

в т.ч.: часть 1 2 работы

часть 1 2 работы

Курсовой проект **9** **семестр**

Всего часов **268 часов**

в т.ч.: часть 1 122 часа

часть 2 146 часов

Зачет **8 семестр**

Экзамен **9 семестр**

Форма получения

высшего образования **дистанционная**

Минск 2011

Рабочая учебная программа составлена на основе базового учебного плана специальности 1-31 03 04 «Информатика», утвержденного Министерством Образования Республики Беларусь 18.04.2006 г., рег. № G.31-025/баз.

**Составители:**

**С. И. Сиротко,** доцент кафедры информатики Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», кандидат физико-математических наук.

**В. В. Шендер,** ассистент кафедры информатики Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», магистр технических наук.

Рассмотрена и рекомендована к утверждению на заседании кафедры информатики, протокол № 3 от 26.09.2011 г.

Зав. кафедрой информатики \_\_\_\_\_\_\_\_\_\_\_ Л. И. Минченко

Одобрена и рекомендована к утверждению Советом факультета компьютерных систем и сетей Учреждения образования «Белорусский государственный университет информатики и радиоэлектроники», протокол № \_\_ от \_\_.\_\_.2011.

Председатель \_\_\_\_\_\_\_\_\_\_\_ В. А. Прытков

СОГЛАСОВАНО

Начальник ОМОУП \_\_\_\_\_\_\_\_\_\_\_ Ц. С. Шикова

ПРОТОКОЛ СОГЛАСОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ  
ПО ИЗУЧАЕМОЙ УЧЕБНОЙ ДИСЦИПЛИНЕ  
С ДРУГИМИ ДИСЦИПЛИНАМИ СПЕЦИАЛЬНОСТИ

|  |  |  |  |
| --- | --- | --- | --- |
| Название дисцип­лины, с которой требуется согласо­вание | Кафедра, обеспе­чивающая изуче­ние дисциплины | Предложения об изменениях в со­держании учеб­ной программы по изу­чаемой дисциплине | Решение, принятое кафедрой, разработав­шей учебную программу (с указанием даты и номера протокола) |
| Алгоритмы и структуры данных | Информатики | нет | согласовано |
| Компьютерные сети | Информатики | нет | согласовано |
| Архитектура компьютеров | Информатики | нет | согласовано |
| Операционные системы | Информатики | нет | согласовано |
| Системное программирование | Информатики | нет | согласовано |

СОГЛАСОВАНО:

Зав. кафедрой информатики Л.И. Минченко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

**Цель преподавания дисциплины.** Целью изучения дисциплины «Операционные системы и среды» является основ построения и функционирования операционных систем (ОС), преимущественно на примере ОС семейства Unix/Linux, и их компонентов, технологий взаимодействия задач и многозадачного программирования, а также получение теоретических знаний и практических навыков работы в различных ОС, создания программного обеспечения (ПО) и основ администрирования систем.

**Задачи изучения дисциплины.** В результате изучения курса студенты должны освоить теоретически и практически различные аспекты архитектуры и функционирования различных ОС, подходы и методы решения задач параллельного программирования, особенности функционирования, разработки и сопровождения ПО в многозадачной среде.

**Знать:**

– архитектуры, концепции и принципы построения и функционирования ОС различного типа и назначения;

– структуру, особенности, средства пользовательского и программного интерфейсов ОС семейств Unix/Linux и Win 32 (Win NT);

– принципы, модели, технологии и механизмы многозадачного программирования и обеспечения взаимодействия вычислительных процессов (потоков), методы и приемы параллельного программирования;

– основы администрирования и управления ОС:

– особенности и принципы разработки системного ПО;

**Уметь:**

– выполнять настройку подсистем ОС и отдельных приложения, базовое администрирование системы;

– разрабатывать ПО с использованием как универсальных, так и командных (скриптовых) языков для решения широкого круга задач;

– проектировать и создавать ПО системного уровня, а также мультипрограммные комплексы, включая выявление и разрешение конфликтов.

**Иметь представление о:**

– областях и особенностях применения ОС и иного системного ПО;

– особенностях разработки и сопровождения ОС, их компонентов и другого системного ПО.

**Перечень дисциплин, усвоение которых необходимо для изучения данной дисциплины**

|  |  |  |
| --- | --- | --- |
| №  п./п. | Название дисциплины | Раздел, тема |
| 1. | Программирование | Все разделы дисциплины |
| 2. | Алгоритмы и структуры данных | Все разделы дисциплины |
| 3. | Операционные системы | Все разделы дисциплины |
| 4. | Системное программирование | Все разделы дисциплины |
| 5. | Архитектура компьютеров | Все разделы дисциплины |

Кроме того, она тесно связана с дисциплиной «Компьютерные сети».

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № темы | Название и содержание тем | Контрольная работа | Индивидуальная практическая работа | Оснащение индивидуальных практических работ | Литература  (номера) | Рекомендуемый объём для изучения  (в часах) | Форма контроля знаний |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Четвертый курс** | | | | | | | |
| **Часть1, семестр 8** | | | | | | | |
| 1 | **Тема 1. Операционные системы (ОС).** Структура, назначение, виды ОС. Архитектуры ОС и их характеристика. |  |  |  | 1, 4, 6, 7, 8, 9, 13, 14, 18 | 8 |  |
| 2 | **Тема 2. Вычислительные процессы и потоки.** Понятие вычислительного процесса, потока. Атрибуты процесса и потока. Жизненный цикл и состояния процессов (потоков). Реализация процессов и потоков. |  |  |  | 2, 3, 4, 5, 6, 7, 8, 9, 15, 16, 17 | 8 |  |
| 3 | **Тема 3. Основы многозадачности и мультипрограммирования.** Многозадачность и многопоточность. Параллелизм в вычислительных системах. Подсистема управления процессами. |  |  |  | 2, 3, 4, 6, 7, 8, 9, 13, 14, 15, 22, 24 | 8 |  |
| 4 | **Тема 4. Планирование выполнения программ.** Задачи планирования, планировщик процессов. Методы и модели планирования выполнения программ. Приоритеты, управление приоритетами. | 1 |  |  | 2, 3, 4, 6, 7, 8, 9, 13, 14, 15, 22, 24 | 6 | Зачёт по контрольной работе |
| 5 | **Тема 5. Особенности выполнения процессов в многозадачной среде.** Взаимодействие процессов в многозадачной среде. Задачи и проблемы взаимодействия. Модели взаимодействия. Конфликты, методы их разрешения. |  |  | 1-6, 9-11 | 2, 4, 6, 9, 13, 14, 17, 22, 24 | 8 |  |
| 6 | **Тема 6. Задача взаимного исключения.** Критический ресурс, критическая секция. Коллизии. Механизмы реализации защиты критических секций. |  |  |  | 2, 4, 6, 7, 8, 9, 15, 16, 17 | 8 |  |
| 7 | **Тема 7. Задача синхронизации процессов.** Ситуации ожидания. Объекты ожидания. Функции ожидания и синхронизация функциями ожидания. |  | 1 | 1, 3-7, 9-11 | 2, 3, 4, 6, 7, 8, 9, 15, 16, 17 | 8 | Зачёт по индивидуальной практической работе |
| 8 | **Тема 8. Событийное управление.** События и их роль в управлении процессами. Сигналы и сообщения. Обработка прерываний и исключительных ситуаций. |  | 2 | 1, 3, 5-6, 10 | 2, 3, 4, 5, 6, 7, 8, 9, 15, 16, 17 | 4 | Зачёт по индивидуальной практической работе |
| 9 | **Тема 9. Элементы многозадачности в однозадачной среде.** Прерывания, обработчики прерываний и резидентные программы в однозадачной среде. | 2 |  |  | 2, 4, 19, 20, 21, 24, 25 | 4 | Зачёт по контрольной работе |
| 10 | **Тема 10. Средства контроля времени выполнения.** Таймеры, функции времени, системные счетчики, другие средства контроля времени. |  |  |  | 2, 3, 6, 7, 8, 9, 15, 16, 17, 19 | 4 |  |
|  | | | | | | Зачет (20 ч) | |
| **Пятый курс** | | | | | | | |
| **Часть 2, семестр 9** | | | | | | | |
| 11 | **Тема 1. Задача обмена данными.** Обмен данными в многозадачной среде. Механизмы межпроцессного обмена. Сравнительная характеристика, критерии сравнения. |  |  |  | 2, 3, 4, 6, 7, 8, 9, 15, 16, 17 | 4 |  |
| 12 | **Тема 2. Разделяемая память.** Особенности организации и использования разделяемой памяти. |  |  |  | 3, 4, 6, 7, 8, 9, 15, 16, 17 | 4 |  |
| 13 | **Тема 3. Использование разделяемых объектов файловой системы.** Разделяемые файлы, каналы, «почтовые ящики», сокеты. |  | 3 | 1, 3-7, 9-11 | 4, 5, 6, 9, 15, 16, 17 | 4 | Зачёт по индивидуальной практической работе |
| 14 | **Тема 4. Асинхронный ввод-вывод.** Организация асинхронного ввода-вывода и его особенности. Средства контроля и управления асинхронными операциями. Параллельный доступ к файлам и коммуникационным ресурсам. |  |  |  | 2, 4, 6, 9, 17 | 4 |  |
| 15 | **Тема 5. Подсистема памяти.** Задачи и функции управления памятью, распределения памяти. Методы управления памятью. Виртуальное адресное пространство и его использование при распределении памяти. |  |  |  | 2, 3, 4, 6, 7, 8, 9, 17 | 8 |  |
| 16 | **Тема 6. Отображение файлов в память.** Использование файловых отображений: доступ к данным, разделяемые данные. |  |  |  | 4, 6, 7, 9, 17 | 4 |  |
| 17 | **Тема 7. Подсистема управления файлами.** Файловые системы – принципы организации, основные функции. Дополнительные требования и функции. Примеры реализаций. |  |  |  | 4, 6, 9, 13, 14, 26 | 8 |  |
| 18 | **Тема 8. Основы сетевого программирования.** Сокеты для доступа к сетевым интерфейсам. Использование сокетов в сетевых приложениях. |  | 4 | 1, 3-7, 9-11 | 3, 6, 9, 7, 10, 11, 12, 17, 26 | 8 |  |
| 19 | **Тема 9. Управление внешними устройствами.** Логические устройства, драйверы. Виды устройств. Организация доступа к внешним устройствам. |  |  |  | 2, 3, 4, 6, 7, 8, 9, 10, 13, 14, 15, 16, 24 | 4 |  |
| 20 | **Тема 10. Интеграция приложений, разнородные приложения.** Удаленный доступ, удаленное исполнение. Распределенные приложения. Технологии OLE, COM, ActiveX, .NET. «Облачные» технологии. | 3 |  |  | 6, 9, 17 | 8 | Зачёт по контрольной работе |
| 21 | **Тема 11. Средства безопасности и их поддержка в ОС.** Безопасность в вычислительных системах. Угрозы, защищаемые объекты, участники подсистемы безопасности. Задачи подсистемы безопасности. Примеры реализаций. |  |  |  | 4, 6, 7, 8, 9, 10, 12, 17, 26 | 4 |  |
| 22 | **Тема 12. Системы реального времени.** Управляющие и встраиваемые системы. Механизмы системного ПО для режимов реального времени. Примеры реализаций. |  |  |  | 2, 4, 13, 14, 22 | 8 |  |
|  | | | | | | Курсовой проект (24 ч) | |
|  | | | | | | Экзамен (30 ч) | |

1. Наименование тем, их содержание

Тема 1. Операционные системы (ОС).

Структура, назначение, виды ОС. Архитектуры ОС и их характеристика.

**Рекомендуемая литература:** 1, 4, 6, 7, 8, 9, 13, 14, 18.

Тема 2. Вычислительные процессы и потоки.

Понятие вычислительного процесса, потока. Атрибуты процесса и потока. Жизненный цикл и состояния процессов (потоков). Реализация процессов и потоков.

**Рекомендуемая литература:** 2, 3, 4, 5, 6, 7, 8, 9, 15, 16, 17.

Тема 3. Основы многозадачности и мультипрограммирования.

Многозадачность и многопоточность. Параллелизм в вычислительных системах. Подсистема управления процессами.

**Рекомендуемая литература:** 2, 3, 4, 6, 7, 8, 9, 13, 14, 15, 22, 24

Тема 4. Планирование выполнения программ.

Задачи планирования, планировщик процессов. Методы и модели планирования выполнения программ. Приоритеты, управление приоритетами.

**Рекомендуемая литература:** 2, 3, 4, 6, 7, 8, 9, 13, 14, 15, 22, 24

Тема 5. Особенности выполнения процессов в многозадачной среде

Взаимодействие процессов в многозадачной среде. Задачи и проблемы взаимодействия. Модели взаимодействия. Конфликты, методы их разрешения.

**Рекомендуемая литература:** 2, 4, 6, 9, 13, 14, 17, 22, 24

Тема 6. Задача взаимного исключения.

Критический ресурс, критическая секция. Коллизии. Механизмы реализации защиты критических секций.

**Рекомендуемая литература:** 2, 4, 6, 7, 8, 9, 15, 16, 17

Тема 7. Задача синхронизации процессов.

Ситуации ожидания. Объекты ожидания. Функции ожидания и синхронизация функциями ожидания.

**Рекомендуемая литература:** 2, 3, 4, 6, 7, 8, 9, 15, 16, 17

Тема 8. Событийное управление.

События и их роль в управлении процессами. Сигналы и сообщения. Обработка прерываний и исключительных ситуаций

**Рекомендуемая литература:** 2, 3, 4, 5, 6, 7, 8, 9, 15, 16, 17

Тема 9. Элементы многозадачности в однозадачной среде.

Прерывания, обработчики прерываний и резидентные программы в однозадачной среде

**Рекомендуемая литература:** 2, 4, 19, 20, 21, 24, 25

Тема 10. Средства контроля времени выполнения.

Таймеры, функции времени, системные счетчики, другие средства контроля времени.

**Рекомендуемая литература:** 2, 3, 6, 7, 8, 9, 15, 16, 17, 19

2. Индивидуальные практические работы, их характеристика

|  |  |  |  |
| --- | --- | --- | --- |
| № п./п. | Тема | Содержание | Объём в часах |
| 1. | Синхронизация процессов | Разработка многопроцессного приложения (комплекса), предусматривающего синхронизацию, взаимное исключение, передачу управления между процессами | 8 |
| 2. | Управление процессами посредством сигналов | Разработка программы, использующей (обрабатывающей) сигналы для синхронизации и управления (на примере программы-демона) | 8 |
| **Итого** | | | **16** |

3. Контрольные работы, их характеристика

|  |  |  |  |
| --- | --- | --- | --- |
| № п./п. | Тема | Характеристика | Объём в часах |
| 1. | Планирование выполнения процессов, приоритеты | Цель работы – более детальное ознакомление с принципами функционирования планировщика процессов, методами планирования, моделями приоритетов. | 8 |
| 2. | Обработчики прерываний и исключений | Цель работы – более детальное ознакомление с низкоуровневыми механизмами обработки прерываний (исключений), и принципами разработки программ, использующих их | 8 |
| **Итого** | | | **16** |

4. ЛИТЕРАТУРА

ОСНОВНАЯ

1. Бах, М. Дж. Архитектура операционной системы Unix. [Электронный ресурс]. – Режим доступа : readr.ru/moris-bah-arhitektura-operacionnoy-sistemi-unix.html

2. Бек, Л. Введение в системное программирование / Л.Бек ; пер. с англ. – М.: Мир, 1988. – 448 с., ил.

3. Глас, Г. Unix для программистов и пользователей / Г. Глас, К. Эйблс. – 3-е изд. – СПб.: БХВ-Петербург, 2004.

4. Гордеев, А.В. Системное программное обеспечение / А.В. Гордеев, А.Ю. Молча­нов – СПб.: Питер, 2001. – 736 с.: ил.

5. Керниган, Б. В. UNIX – универсальная среда программирования / Б.В. Керниган, Р. Пайк; пер. с англ. – М.: Финансы и статистика, 1992.

6. Рихтер, Дж. Windows для профессионалов / Дж. Рихтер. – СПб.: Питер, 2000. – 752 с.

7. Робачевский, А. М. Операционная система UNIX / А.М. Робачевский, С. Немнюгин, О. Стесик. – 2-е изд. – БХВ-Петербург, 2007.

8. Рочкинд, М. Дж. Программирование для UNIX / М.Дж. Рочкинд. – 2-е изд. – СПб.: БХВ-Петербург, 2005.

9. Руссинович, М. Внутреннее устройство Microsoft Windows. / М. Рус­си­нович, Д. Соломон. – 4-е изд. – СПб.: Питер, Русская Редакция, 2005. – 992 с.

10. Сорокина, С.И. Программирование драйверов и систем безопасности: Учеб. пособие. / С.И. Сорокина, А.Ю. Тихонов, А.Ю. Щербаков. – СПб.: БХВ-Петербург, М.: издатель Молчанов С.В. – 2002. – 256 с.: ил.

11. Стивенс, У.Р. Протоколы TCP/IP. Практическое руководство. – BHV, 2003. – 672с.

12. Стивенс, У.Р., Феннер Б., Рудофф Э.М. UNIX. Разработка сетевых приложений / У.Р. Стивенс, Б. Феннер, Э.М. Рудофф – 3-е изд. – СПб.: Питер, 2007. – 1040 с.

13. Танненбаум, Э. Современные операционные системы / Э. Таннен­ба­ум. – 2-е изд. – СПб.: Питер, 2002.

14. Таненбаум, Э., Вудхалл, А. Операционные системы. Разработка и реализация. 3-е изд. – СПб.: Питер, 2007. – 704 с.

ДОПОЛНИТЕЛЬНАЯ

15. Беляков, М. И. Мобильная операционная система / М. И. Беляков, Ю.И. Рабовер, А. А. Фридман. – М.: Радио и связь, 1991.

16. Богатырев, А. Хрестоматия по программированию на Си в UNIX. [Электронный ресурс]. – Режим доступа : opennet.ru/docs/RUS/bogatyrev; lib.ru/CTOTOR/book.txt.

17. Вильямс, А. Системное программирование в Windows 2000 для профессионалов / А. Вильямс ; пер. с англ. – СПб.: Питер, 2001. – 624 с., ил.

18. Готье, Р. Руководство по ОС UNIX / Р. Готье. – М.: Финансы и статистика, 1985.

19. Джордейн, Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT / Р. Джордейн, ; пер. с англ. – М.: Финансы и статистика, 1992. – 544 с.

20. Зубков С.В. Assembler для DOS, Windows и UNIX. / С.В. Зубков.  – 3-е изд. – М.: ДМК пресс, СПб.: Питер, 2006. – 608 с.

21. Касаткин, А.И. Профессиональное программирование на языке Си. Управление ресурсами: Справ. пособие / А.И. Касаткин. – Минск: Выш. шк., 1992. – 432 с.

22. Программирование на параллельных вычислительных системах: Пер. с англ. / Р. Бэбб, Дж. Мак-Гроу, Т. Акселрод и др.; под ред. Р.Бэбба II. – М.:Мир, 1991. – 376 с.: ил.

23. Сейдж, Р. Приемы профессиональной работы в UNIX / Рассел С. Сэйдж // Онлайн библиотека rulibrary.com [Электронный ресурс] – Режим доступа : www.rulibrary.com/book-99.html.

24. Скляров, В.А. Программное и лингвистическое обеспечение персональных ЭВМ. Системы общего назначения: Справ. пособие / В.А. Скляров. – Минск: Выш. шк., 1992. – 462 с.: ил.

25. Скляров, В.А. Программное и лингвистическое обеспечение персональных ЭВМ. Новые системы: Справ. пособие / В.А. Скляров. – Минск: Выш. шк., 1992. – 334 с.: ил.

26. UNIX – руководство системного администратора / Э. Немеет [и др.] ; пер. с англ. – Киев : BHV, 2000.

5. ПЕРЕЧЕНЬ КОМПЬЮТЕРНЫХ ПРОГРАММ, НАГЛЯДНЫХ И ДРУГИХ   
ПОСОБИЙ, МЕТОДИЧЕСКИХ УКАЗАНИЙ И МАТЕРИАЛОВ   
И ТЕХНИЧЕСКИХ СРЕДСТВ ОБУЧЕНИЯ

1. Операционные системы и среды : электронный учеб.-метод. комплекс [Электронный ресурс]. – Минск : БГУИР, 2007. – Режим доступа : www.bsuir.by.

2. Система тестирования "Инфотест" кафедры информатики: тесты "GNU/Linux", "GNU/Linux Advanced", "GNU/Linux Kernel"

3. ПЭВМ с установленной ОС семейства Unix/Linux, либо средства для удаленного доступа к ЭВМ с указанными ОС.

4. ПЭВМ с установленной ОС семейства Win 32.

5. Стандартный пакет утилит Unix/Linux [Электронный ресурс]. – Режим доступа : стандартные компоненты операционных систем семейства Unix

6. Компилятор gcc и другие компоненты среды программирования Unix/Linux [Электронный ресурс]. – Режим доступа : стандартные компоненты операционных систем семейства Unix

7. Среда разработки Microsoft Visual Studio [Электронный ресурс]. – Режим доступа : Компонент ПО учебных классов

8. Среда разработки Borland IDE 3.1: C/C++ compiler, assembler [Электронный ресурс]. – Режим доступа : Компонент ПО учебных классов

9. Среда разработки Microsoft Driver Developer Kit (DDK) [Электронный ресурс]. – Режим доступа : http://www.microsoft.com/whdc/devtools/ddk/.

10. Среда разработки Microsoft Software Developer Kit (SDK) [Электронный ресурс]. – Режим доступа : http://www.microsoft.com/downloads/.

10. Справочная система man Unix [Электронный ресурс]. – Режим доступа : стандартный компонент операционных систем семейства Unix

11. Информационно-справочная система Microsoft Developer Network (MSDN) [Электронный ресурс]. – Режим доступа : http://www.microsoft.com/downloads.

12. Справочная система TeachHelp [Электронный ресурс]. – Режим доступа : Компонент ПО учебных классов.

# Теоретический раздел

## Лекции

### Введение в дисциплину

Дисциплина «Операционные системы и среды», преподаваемая для специальности «Информатика», является продолжением дисциплин «Системное программирование» и «Операционные системы», и служит дальнейшим развитием, углублением и обобщением начатых в них тем. Так как специфический для платформ Windows и Unix платформ и ориентированный на практические аспекты материал в основном можно считать рассмотренным, в настоящем курсе есть возможность сконцентрироваться на задачах многозадачного программирования, взаимодействия параллельных процессов, возникающих при этом проблем и используемых для их разрешения методов и механизмов. Рассматриваются также отдельные вопросы низкоуровневого программирования, специализированные ОС и некоторые другие.

### Тема 1. Операционные системы

#### 1.1. Основные определения

Операционная система – центральная часть системного программного обеспечения (ПО), набор программ для обеспечения функционирования аппаратного и программного обеспечения вычислительной системы (устройства) как единого комплекса для решения конкретных задач. Помимо ОС, к системному ПО относят также драйверы и системные утилиты. Эти компоненты отвечают за решение широкого круга задач, но занимают подчиненное по отношению к ОС положение.

В ОС, в свою очередь, принято выделять составные части:

– ядро (kernel) – реализует основные функции ОС;

– программный интерфейс, системные вызовы (Application Program Interface – API) – точки доступа к функциям ядра, предоставляемые (экспортируемые) для использования прикладными программами;

– подключаемые к ядру, но не входящие в него модули, например сервисы; заметим, что драйверы также можно считать разновидностью таких модулей, но ввиду специфики функций их традиционно рассматривают как отдельную группу программ;

– уровень абстрагирования оборудования (Hardware Abstraction Layer – HAL).

Модульное построение ОС улучшает ее мобильность, гибкость и модифицируемость. В рамках модульной концепции и HAL, и драйверы служат цели изоляции аппаратно-зависимого кода в небольшом количестве модулей, в то время как остальная часть системы может быть сделана аппаратно-независимой. Традиционно к HAL относят средства абстрагирования низкоуровневого доступа к аппаратным ресурсам, а драйверы отвечают за более высокоуровневые функции логического управления устройствами.

#### 1.2. Архитектуры операционных систем

Одним из важнейших классификационных признаков ОС является отнесение ее к той или иной архитектуре.

Под архитектурой ОС (или ядра ОС, если она достаточно сложна, чтобы иметь выделенное ядро) принято понимать совокупность ее структурных и функциональных (связи между элементами структуры) признаков, которыми обладает данная ОС и которые отличают ее от других. В более широком смысле это подход к построению и функционированию ОС, составу ее компонентов, выбору интерфейсов и так далее. Выбор архитектуры ОС существенным образом влияет на весь комплекс ее технических и потребительских характеристик.

На протяжении существования вычислительной техники существенно менялись требования с ПО в целом и к системному ПО в частности. Эволюция ОС дала ряд основных типов архитектур, характеризующихся своими достоинствами и недостатками.

Наиболее ранние и наиболее простые ОС могли быть организованы, например, в виде расширения (библиотеки) к языку или среде программирования, и их средства активизировались исключительно по инициативе прикладной программы, написанной на этом языке – т.н. пассивные ОС. В ряде случаев (обычно это разного рода встраиваемые системы) и сейчас возможно распределить функции ОС между исполняемыми прикладными программами. Однако для более сложных вычислительных систем, решающих более разнообразные задачи, особенно если актуальна унификация ПО, – необходимо выделять как самостоятельный элемент ОС, имеющую соответствующую архитектуру.

Более сложным и функциональным вариантом ОС является т.н. монитор – набор завершенных и готовых к исполнению системных программ (подпрограмм) и данных, разделяемых и выполняемых другими (прикладными) программами. Технологически это можно представить как оверлеи или динамически подключаемые библиотеки (DLL) с документированными точками входа, доступные всем программам в системе. При этом системный код уже имеет компактный вид и отделен от прикладного, однако все равно остается пассивным – его исполнение инициируется только по запросам приложений.

Современные многозадачные ОС должны быть активными: их функции реализуются в виде системных процессов, выполняющихся параллельно с прикладными. Это требует обязательного решения проблем управления, изоляции и взаимодействия процессов.

1.2.1. Системы с монолитным ядром (monolithic kernel)

Простейший, наиболее естественный и поэтому традиционный способ организации ядра ОС – представить его как единую программу, состоящую из набора подпрограмм, служащих обработчиками запросов через API и аппаратных событий (см. рис. 1.2). Ядро целиком функционирует на едином уровне привилегий, и каждая его подпрограмма может (потенциально) вызвать любую другую, и ОС можно считать целиком состоящей из одного ядра. Если ядро достаточно сложное и объемное, оно может быть структурировано – выделяются группы подпрограмм, отличающиеся назначением и интерфейсом: диспетчер системных функций, обслуживание API, сервисные процедуры и т.д.

01_Архитектура_монолитная

Рис. 1.1 – Система с монолитным ядром

Обращение прикладной программы к ядру осуществляется через API ядра – системные вызовы (system call). Для программы это выглядит как обычный вызов подпрограммы, но сопровождается переключением привилегий на уровень ядра (kernel mode) для выполнения кода ядра. Обратное переключение происходит при возврате из системного вызова. Поэтому системные вызовы оформляются особым образом – например, как ловушки (trap) или шлюзы защищенного режима (protected mode)[[1]](#footnote-1). Для взаимодействия подпрограмм внутри ядра, не требующего переключения привилегий, достаточно обычных процедурных вызовов, что требует намного мéньших затрат и повышает общую производительность.

Вместе с тем, монолитное ядро более критично к качеству проектирования и программирования, при взаимодействии его компонентов велика вероятность конфликтов, например из-за повторных вызовов подпрограмм. Изменение конфигурации ядра, как правило, связано с перекомпиляцией кода (повторной «сборкой»).

Монолитное ядро характерно для ранних ОС, в настоящее время применяется в основном в сравнительно простых системах. В то же время, этой архитектуре с различными модификациями следует большинство Unix-систем.

После появления микроядерной архитектуры (см. ниже) монолитное ядро стали называть также макроядром.

1.2.2. Многоуровневые (многослойные) системы (layered system)

Логичным развитием идеи структурирования системы является построение ее в виде иерархии модулей – "слоёв", каждый из которых отвечает за определенный набор взаимосвязанных функций, причем каждый слой взаимодействует лишь с соседними с ним. Программные интерфейсы для взаимодействия выделяются на каждом из уровней (см. рис. 1.2).

Иерархический подход применяется вообще очень широко в различных сложных системах; получаемую структуру часто называют «стеком» уровней (модулей, протоколов и т.п.). Характерные примеры – стековая модель взаимодействия открытых систем OSI и стек сетевых протоколов TCP/IP и т.п. В то же время, в собственно ОС, ввиду разнообразия выполняемых функций, проблему представляет выделение четкой иерархии уровней.

Первой слоёной системой считают ОС THE (Technishe Hogeschool Eindhoven), реализованную под руководством Дейкстры в 1968 г.

01_Архитектура_многоуровневая

Рис. 1.2 – Фрагмент структуры многоуровневой системы

Общие количество и функции уровней зависят от особенностей конкретной системы. Например, упомянутая ОС THE имела 6 уровней, модель OSI – 7, стек TCP/IP – 4, и т.д. В случае ОС нижний слой обычно отвечает за взаимодействие с аппаратурой, верхний – с программами пользователя. Выбор режима исполнения, формат вызовов, переключение привилегий и т.п. могут решаться для каждого уровня по-разному, в зависимости от назначения и особенностей системы. Общее правило – уровни должны быть максимально изолированы друг от друга, но на практике возможны различные упрощения для уменьшения служебных затрат и повышения производительности системы. Например, с точки зрения независимости уровней блоки данных (массивы) при передаче между ними должны копироваться и оформляться как сообщения, но часто используется передача по указателю (ссылке) – более экономная, но требующая пересечения адресных пространств процессов разных уровней. Компоненты внутри одного уровня взаимодействуют напрямую, с минимальными затратами.

Важным преимуществом слоёной архитектуры является её структурированность и модульность – возможность заменять и модифицировать любой из уровней, не затрагивая соседние. Как следствие, многоуровневые системы удобны для реализации и отладки. Однако служебные затраты могут быть достаточно велики из-за многократной передачи запросов сквозь стек слоёв.

В целом, многоуровневое иерархическое построение применяется в том или ином виде в подавляющем большинстве сложных систем.

1.2.3. Микроядерная архитектура (microkernel architecture)

Сложность связей внутри ядра, особенности выполняемых им функций (особенно в части доступа к аппаратным ресурсам) и множественность обращений со стороны прикладных программ (в многозадачной системе) приводят к конфликтам между подпрограммами ядра. В частности, это проблема нереентерабельности (невозможности повторного вхождения в подпрограмму) и критической зависимости от времени реагирования на события и длительности выполнения некоторых операций. Эти проблемы характерны для работы с аппаратными ресурсами, особенности которой и не могут быть изменены или игнорированы. Решение в виде использование передаваемых через очередь запросов (сообщений) к критическим не-реентерабельным частям кода ядра вместо прямых (процедурных) вызовов было применено в таких системах как RT-11, OS-9, RSX-11 и VMS. Дальнейшее развитие этой идеи привело к выделению исполнительных блоков в специальный компактный элемент архитектуры. Обычно считается, что сам термин «микроядро» (microkernel) был введен Ирой Голдштейн и Полом Дейлом в 1989 году.

Микроядерная архитектура представляет собой вариант иерархической организации с двумя уровнями: микроядро и обслуживающие системные процессы — серверы (см. рис. 1.3).

01_Архитектура_микроядро

Рис. 1.3 – Система на основе микроядра

Микроядро (microkernel) – специальный модуль ядра, реализующий наиболее важные, аппаратно-зависимые функции ОС: управление процессами и их взаимодействие, управление ресурсами, включая память, ввод-вывод, первичная обработка прерываний. Такой модуль обычно невелик, а его код выполняется только в привилегированном режиме.

Серверы или сервисы (service) – системные модули (во время работы системы – процессы), выполняющие все остальные функции ОС: работа с файловой системой, логическими устройствами, другими системными объектами. Они выполняются на пользовательском уровне привилегий и в адресном пространстве прикладных задач, т.е. являются «обычными» процессами.

Обращения к микроядру оформляются как передача внутренних запросов, которые принято называть сообщениями. Сообщения помещаются в очередь, откуда планировщик (диспетчер) микроядра извлекает их и передает соответствующим исполнительным блокам. Создание запроса и постановка его в очередь требует очень небольшого времени, а в течение его выполнения процесс-иницииатор может находиться в состоянии ожидания или продолжать выполняться (но не имея результатов запроса). Процессы самого микроядра всегда имеют преимущество перед прикладными, поэтому выполняемые им критические по времени операции, например реагирование на события, не зависят от состояния прикладных процессов. Таким образом, микроядро обеспечивает корректность выборки и выполнения запросов, соблюдение приоритетов операций, асинхронность и независимость процессов.

Микроядерная архитектура способствует минимизации высокопривилегированного, а также и аппаратно-зависимого кода, повышению надежности и устойчивости системы, она более пригодна для распараллеливания, масштабирования, работы в реальном времени. Ограниченный размер кода микроядра позволяет оптимизировать его выполнение на аппаратном уровне – в первую очередь, за счет размещения целиком в кэше процессора. Модульное построение системы позволяет управлять большинством системных процессов так же, как и прикладными программами: исполнять, останавливать или заменять другими. Микроядерная архитектура хорошо сочетается с технологиями объектного проектирования. К недостаткам относятся дополнительные затраты при использовании интерфейса сообщений – гораздо бóльшие, чем в случае процедурного интерфейса; тем не менее, упоминавшаяся ранее микроядерная ОС QNX является одной из наиболее быстрых систем. Как и в случае любой «не-монолитной» ОС, особого внимания требует проектирование иерархии модулей, в данном случае включение конкретных функций в микроядро либо отнесение их к «внешним».

С точки зрения общей эффективности, желательно минимизировать число вызовов к микроядру и, следовательно, оптимально распределить функции между ним и «внешними» модулями-серверами. Для микроядер первого поколения число системных вызовов (100-150) и объем кода (сотни Кбайт) примерно такие же, как и у современных им монолитных ядер[[2]](#footnote-2). При минимизации микроядра число интерфейсов системных вызовов и, соответственно, объем кода сокращаются до предела: в «легком» микроядре остаются только немногочисленные «базовыми» действиями, а со всеми более высокоуровневыми объектами работают модули прикладного уровня. В принципе, достаточно четырех интерфейсов: создание/удаление порта взаимодействия и прием/передача блока данных через порт. Некоторые ОС близки к этому пределу: так, например, микроядро ОС L4 содержит 7 интерфейсов и имеет размер 12 Кбайт, а микроядро QNX – всего 10 Кбайт. С другой стороны, есть и противоположная тенденция – «тяжелое» микроядро, в которое включаются даже сложные функции, что повышает эффективность их выполнения, но ослабляет некоторые достоинства архитектуры[[3]](#footnote-3). (заметим, что аналогично разрастаются и монолитные ядра).

Современными примерами микроядерных ОС могут служить QNX, Minix3, Mac OS X, L4, GNU Hurd, WPOS (две последние используют ядро Mach, разработанное в Университете Беркли). Элементы этой архитектуры присутствуют в OS/2, Windows NT и 4.4BSD.

1.2.4. Виртуальные машины (virtual machine)

Доведенная до логического завершения идея изоляции пользовательского уровня от деталей вычислительной системы. Вместо прямого или косвенного обращения к ним все программы взаимодействуют только с виртуальной машиной – функциональным эквивалентом реальной, но реализуемой различными аппаратными и программными средствами, а точнее – выполняются ею. «Глубина» реализации может быть различной: виртуальная машина может выполнять «виртуальные» машинные команды, эмулируя работу «виртуального» процессора, а может – строки алгоритмического языка высокого уровня («встроенный» в систему интерпретатор, например, Forth- или Java-машины). Соответственно, зависимой от аппаратной платформы будет только сама виртуальная машина, прочее же ПО, включая наиболее сложные высокоуровневые алгоритмы самой ОС, может и должно быть написано переносимым, что существенно упрощает унификацию ПО, создание программно-совместимых семейств ЭВМ[[4]](#footnote-4).

Первой ОС, основанной на концепции виртуальной машины, считается CP/CMS (она же VM/370) для семейства машин IBM/370. Впоследствии часто использовались для организации многопользовательского режима (предоставление каждому пользователю «собственной» виртуальной машины, поддерживаемой средствами одной «большой» ЭВМ), эмуляции рабочих мест заданной конфигурации (продукты типа VMWare), выполнения чужеродного кода (эмуляторы процессоров) и т.д., т.е. не обязательно в качестве собственно ОС. Общим недостатком любых «виртуальных машин» остается бóльшие затраты ресурсов и, соответственно, мéньшее быстродействие по сравнению с выполнением непосредственно на «родной» платформе из-за необходимости фактически интерпретировать «чужой» код вместо естественного его исполнения. Однако влияние этого недостатка ослабляется по мере повышения характеристик аппаратного обеспечения и снижения стоимости аппаратных ресурсов.

Эмулировать заданную архитектуру можно и на аппаратном уровне, что представляется оптимальной реализации концепций виртуальных машин, но это выходит за рамки настоящего курса[[5]](#footnote-5).

Элементы виртуальной машины присутствуют в том числе в таких ОС как OS/2 и Windows NT: уровень HAL (Hardware Abstraction Layer) изолирует («абстрагирует») вышестоящие уровни от аппаратуры, что теоретически делает систему переносимой между аппаратными платформами.

2.2.5. Модифицированные и смешанные архитектура

Представляет собой попытку совместить достоинства различных архитектур и/или компенсировать их недостатки. Встречается часто, т.к. «чистые» архитектуры на практике обычно приходится приспосабливать к конкретным реальным условиям и требованиям. Например, архитектура Windows NT определяется как «модифицированная микроядерная».

В частности, в последнее время получили развитие «web-ориенти–ро–ванные» ОС, например проект Chromium OS. Архитектуру Chromium (см. рис. 1.4) можно рассматривать как распространение концепции виртуальной машины на среду Web: роль среды выполнения приложений играет браузер Chromium с необходимыми модулями «ядра», который является фактически единственным приложением для «настоящей» ОС. Таким образом, приложения прикладного уровня оказываются ограничены поддерживаемыми браузером функциями и полностью изолированы от низкоуровневых ресурсов системы. Результатом является высокая степень надежности и переносимости, а главное – унифицированный и прозрачный доступ к программам и ресурсам посредством web-технологий независимо от их реального размещения – локального или удаленного, что особенно ценно для мобильных устройств. С другой стороны, существенно возрастают вычислительные затраты и требования к аппаратной платформе, особенно для реализации графических интерфейсов.

01_Архитектура_Chromium

Рис. 1.4 – Архитектура ОС проекта Chromium OS

В целом как актуальную тенденцию можно отметить рост роли систем, поддерживающих удаленный доступ, параллельную и распределенную обработку данных, а также рассчитанных на использование на мобильных устройствах. Прогресс в этом направлении связан в том числе и с адаптацией их архитектур к специфике предъявляемых требований.

### Тема 2. Вычислительные процессы и потоки.

Понятие вычислительного процесса, потока. Атрибуты процесса и потока. Жизненный цикл и состояния процессов (потоков). Реализация процессов и потоков.

### Тема 3. Основы многозадачности и мультипрограммирования.

Многозадачность и многопоточность.

Параллелизм в вычислительных системах.

Подсистема управления процессами.

### Тема 4. Планирование выполнения программ.

Задачи планирования выполнения программ.

Планировщик процессов.

Методы и модели планирования выполнения программ.

Приоритеты, управление приоритетами.

### Тема 5. Особенности выполнения процессов в многозадачной среде

Взаимодействие процессов в многозадачной среде.

Задачи и проблемы взаимодействия.

Модели взаимодействия.

Конфликты, методы их разрешения.

### Тема 6. Задача взаимного исключения.

Определение взаимного исключения.

Критический ресурс, критическая секция.

Коллизии.

Механизмы реализации защиты критических секций.

### Тема 7. Задача синхронизации процессов.

Ситуации ожидания.

Объекты ожидания.

Функции ожидания и синхронизация функциями ожидания.

### Тема 8. Событийное управление.

События и их роль в управлении процессами.

Сигналы и сообщения.

Обработка прерываний и исключительных ситуаций

### Тема 9. Элементы многозадачности в однозадачной среде.

Прерывания, обработчики прерываний.

Резидентные программы.

Проблемы реентерабельности, глобальные данные, коллизии.

Взаимодействие с резидентными программами.

### Тема 10. Средства контроля времени выполнения.

Таймеры.

Функции времени

Системные счетчики

Другие средства контроля времени.

# Практический раздел

## Контрольные работы

### Контрольная работа №1

Планирование выполнения процессов, приоритеты.

#### Указания по выбору варианта

Цель работы – более детальное ознакомление с принципами функционирования планировщика процессов, методами планирования, моделями приоритетов.

Контрольная работа должна быть оформлена в соответствии с общеустановленными нормами и правилами, предъявляемыми к выполнению контрольных работ. При выполнении задания необходимо пользоваться несколькими источниками.

Контрольная работа не предполагает строгого разделения на теоретическую и практическую части. Материал, соответствующий заданию, должен содержать как описания методов, принципов, механизмов, объектов, функций и т.д., так и примеры, необходимые для иллюстрации излагаемого.

Выбор варианта задания контрольной работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

Таблица 1 – Варианты заданий контрольной работы № 1

|  |  |
| --- | --- |
| № п/п | Задание |
| 1. | Многозадачность. Виды многозадачности. |
| 2. | Многозадачность с разделением времени, вытесняющая многозадачность. Квантование времени выполнения. |
| 3. | Параллельное выполнение процессов и многопоточность (различные платформы) |
| 4. | Состояния процессов (потоков) |
| 5. | Планирование времени выполнения задач. Планировщик задач. |
| 6. | Дисциплины планирования и приоритеты |
| 7. | Требования к дисциплине планирования в системах различного назначения. Привилегированные и непривилегированные уровни приоритета |
| 8. | Средства управления приоритетами в ОС Windows |
| 9. | Средства управления приоритетами в ОС Unix (Linux) |
| 10. | Основные правила назначения приоритетов в многопроцессных (многопоточных) приложениях |

### Контрольная работа №2

Обработчики прерываний и исключений

#### Указания по выбору варианта

Цель работы – более детальное ознакомление с низкоуровневыми механизмами обработки прерываний (исключений), и принципами разработки программ, использующих их

Контрольная работа должна быть оформлена в соответствии с общеустановленными нормами и правилами, предъявляемыми к выполнению контрольных работ. При выполнении задания необходимо пользоваться несколькими источниками.

Контрольная работа не предполагает строгого разделения на теоретическую и практическую части. Материал, соответствующий заданию, должен содержать как описания методов, принципов, механизмов, объектов, функций и т.д., так и примеры, необходимые для иллюстрации излагаемого.

Выбор варианта задания контрольной работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

Таблица 2 – Варианты заданий контрольной работы № 2

|  |  |
| --- | --- |
| № п/п | Задание |
| 1. | Способы организации взаимодействия в вычислительной системе. Взаимодействия по прерываниям. |
| 2. | Прерывания. Виды и источники прерываний |
| 3. | Средства обслуживания прерываний (аппаратный и программный уровни) |
| 4. | Обработчики прерываний в составе ПО вычислительных систем |
| 5. | Программные надстройки прикладного уровня для обслуживания и использования прерываний (исключений) |
| 6. | События, событийное управление. Архитектура событийно-управляемых приложений. |
| 7. | Исключительные ситуации – поддержка и обработка в языках программирования (примеры) |
| 8. | Исключительные ситуации – поддержка и обработка средствами операционной системы (примеры) |
| 9. | Особенности ПО с использованием обработчиков прерываний (исключений). |
| 10. | Обеспечение взаимодействия с обработчиками прерываний (исключений) и управления ими |

## Индивидуальные практические работы

### Индивидуальная практическая работа №1

Синхронизация процессов

#### Указания по выбору варианта

Работа предполагает разработку многопроцессного приложения (комплекса), предусматривающего синхронизацию, взаимное исключение, передачу управления между процессами.

Индивидуальная практическая работа предполагает защиту выполненного задания, а также ответы на теоретические вопросы, демонстрирующие владение тематикой.

Индивидуальная практическая работа оформляется в соответствии с общеустановленными нормами и правилами.

Выбор варианта задания индивидуальной практической работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

#### Теоретическая часть (вопросы)

1. …

2.

#### Практическая часть (варианты заданий)

##### Вариант задания №1.

##### Вариант задания №2.

### Индивидуальная практическая работа №2

Управление процессами посредством сигналов

#### Указания по выбору варианта

Разработка программы, использующей (обрабатывающей) сигналы для синхронизации и управления (на примере программы-демона.

Индивидуальная практическая работа предполагает защиту выполненного задания, а также ответы на теоретические вопросы, демонстрирующие владение тематикой.

Индивидуальная практическая работа оформляется в соответствии с общеустановленными нормами и правилами.

Выбор варианта задания индивидуальной практической работы осуществляется студентом самостоятельно на основании номера зачетной книжки (последние цифры, после номера группы). Если порядковый номер превышает количество вариантов, для выбора номера варианта используется остаток от деления порядкового номера на количество вариантов. В отдельных случаях задание может быть сформулировано преподавателем индивидуально.

#### Теоретическая часть (вопросы)

1. …

#### Практическая часть (варианты заданий)

##### Вариант задания №1.

##### Вариант задания №2.

1. Подробное рассмотрение этой и других возможностей защищенного режима (*protected mode*) в архитектуре *IA32* (процессорs семейств i386+) относятся к дисциплине «Архитектура компьютеров». [↑](#footnote-ref-1)
2. В этом смысле, можно сопоставить микроядро с монолитным ядром, а внешние серверы — с библиотеками. [↑](#footnote-ref-2)
3. Аналогичные тенденции свойственны также и монолитным ядрам. [↑](#footnote-ref-3)
4. Альтернативный подход к унификации ПО предполагает проектирование аппаратных средств таким образом, чтобы обеспечить выполнение некоторого заранее оговоренного набора функций, поверх которых реализуется программная надстройка, обеспечивающая доступ к аппаратуре – т.н. *абстрактная машина*. Абстрактную машину можно рассматривать как программную модель реальной ЭВМ, на которую ориентируется все остальное ПО. Примером может служить ОС MS-DOS с обработчиками программных прерываний-функций для IBM PC-совместимых ЭВМ. [↑](#footnote-ref-4)
5. Имевшие распространение ранее микропрограммируемые секционируемые процессоры позволяли эмулировать заданные архитектуру, программную модель и систему команд, причем частично модифицировать систему можно было заменой микропрограмм в ее ПЗУ. [↑](#footnote-ref-5)