

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Локтионова Оксана Геннадьевна
Должность: проректор по учебной работе
Дата подписания: 05.04.2023 14:09:28
Уникальный программный ключ:
0b817ca911e6668abb13a5d426d49e5f1c11eabb73e943df4a4851fda56d089

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования «Юго-Западный
государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ

Проректор по учебной работе

О.Г. Локтионова

« 23 » 03

2023 г.



ИССЛЕДОВАНИЕ СТРУКТУРЫ ФАЙЛА

Методические указания по выполнению лабораторных работ для
студентов укрупненной группы специальностей и направлений
подготовки 10.00.00

Курск 2023

УДК 681.3

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры
«Информационная безопасность» А.Л. Марухленко

Исследование структуры файла: методические указания по выполнению лабораторной работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 19 с. Библиогр.: с. 19.

Излагаются методические указания по выполнению лабораторной работы на персональной ЭВМ. Изучаются методы анализа форматов исполняемых файлов в подсистеме Win32.

Методические указания по выполнению лабораторных работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____ . Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 148. Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

ОГЛАВЛЕНИЕ

1. Цель работы.	4
2. Сведения о структуре исполняемых на платформе Win32 файлов.....	4
3. Исследование формата исполняемого файла.....	8
3.1. Анализ заголовка файла.	8
3.2. Секции.	14
3.3. Таблица импорта функций.....	16
4. Задание на лабораторную работу.....	18
5. Содержание отчёта.....	18
6. Вопросы для самоконтроля.....	18
7. Библиографический список	19

1. ЦЕЛЬ РАБОТЫ.

Получение представления о структуре исполняемых файлов Win32, ознакомление с PE-заголовком, секциями и таблицами импорта приложений, получение начальных сведений о работе загрузчика Windows NT (ntdll.dll), а также получение навыков работы с редакторами исполняемых файлов и дизассемблерами.

2. СВЕДЕНИЯ О СТРУКТУРЕ ИСПОЛНЯЕМЫХ НА ПЛАТФОРМЕ WIN32 ФАЙЛОВ.

PE (*Portable Executable*) – это файловый формат исполняемых файлов Win32, спецификации которого происходят от Unix/COFF (*Common Object File Format*). Загрузчик PE распознает и использует этот файловый формат даже когда приложение запускается на не PC CPU платформе.

Общую структуру PE-файла можно представить следующим образом (рис. 1):

DOS MZ-заголовок
Заглушка DOS (stub)
Заголовок PE
Таблица секций
Секция 1
...
Секция n

Рис. 1 – Структура формата PE.

Как видно из рисунка, все PE-файлы (в том числе 32-битные с расширением dll) должны начинаться с DOS-заголовка. Обычно он служит для того, чтобы в случае запуска программы из операционной системы, не знающей о PE-формате, такой, как DOS, запускалась DOS-stub. Большинство компиляторов используют Int 21h, сервис 9, чтобы напечатать строку “This program cannot run in DOS mode”. Но поскольку DOS-заглушка – это полноценное DOS-приложение, не исключено другое её применение.

После DOS-stub'a идет PE-заголовок. PE-заголовок – это общее название структуры под названием IMAGE_NT_HEADERS. Эта структура содержит много основных полей, используемых PE-загрузчиком. В случае, если программа запускается операционной системой, которая знает о PE-формате, PE-загрузчик может найти смещение PE-заголовка в заголовке DOS-MZ. После этого он может пропустить DOS-stub и перейти напрямую к PE-заголовку, который является настоящим заголовком исполняемого файла.

Настоящее содержимое PE-файла разделено на блоки, называемые секциями. Секция – это блок данных с общими атрибутами, такими как код/данные, чтение/запись и т.д. Группирование данных производится на основе их атрибутов. Не играет роли, как используются код/данные, если данные/код в PE-файле имеют одинаковые атрибуты, они могут быть сгруппированы в секцию, т.е. секции могут содержать и данные и код одновременно, главное, чтобы те имели одинаковые атрибуты. Если у вас есть данные, и вы хотите, чтобы они были доступны только для чтения, вы можете поместить эти данные в секцию, помеченную соответствующим атрибутом. Таблица секций, соответственно, располагает данными о каждой секции PE-файла.

Остановимся подробнее на заголовке PE-файла. Адрес PE-заголовка загрузчик получает из DOS-MZ заголовка, который, в свою очередь тоже является структурой. Сама структура IMAGE_NT_HEADERS, представляющая PE-заголовок, определена следующим образом:

```
IMAGE_NT_HEADERS STRUCT
    Signature dd ?
    FileHeader IMAGE_FILE_HEADER <>
    OptionalHeader IMAGE_OPTIONAL_HEADER32 <>
IMAGE_NT_HEADERS ENDS
```

Поле Signature представляет собой двойное слово 50 45 00 00 (ASCII-символы “PE”, за которыми следуют два нулевых байта).

Далее следуют две структуры, IMAGE_FILE_HEADER и IMAGE_OPTIONAL_HEADER. Приводить их полное описание нецелесообразно, поскольку большинство из их полей не используются, либо используются для отладки. Поэтому ограничимся перечисленными полями в следующих таблицах:

Таблица 1. – Структура файлового заголовка

Имя поля	Описание
Machine	Целевая CPU платформа машины, на котором файл будет исполняться. Для Intel это значение равно константе IMAGE_FILE_MACHINE_I386. Это поле обычно заполняется компилятором, когда он встречается соответствующую директиву (например, .386 в <code>masm</code>).
NumberOfSections	Число секций PE-файла. Оно также определяет количество элементов в таблице секций.
TimeDataStamp	Дата и время создания файла.
SizeOfOptionalHeader	Размер опционального заголовка.
Characteristics	Содержит дополнительные флаги файла, например, является ли он <code>exe</code> или <code>dll</code> .

Таблица 2. – Структура опционального заголовка

Имя поля	Описание
AddressOfEntryPoint	RVA точки входа – первой инструкции, которая будет передана загрузчику после размещения файла в виртуальной памяти.
ImageBase	Предпочитаемый адрес загрузки PE-файла. Файл загружается в указанную область если она не занята никаким другим процессом.
SectionAlignment	Размер выравнивания секций в памяти. Например, если значение в этом поле равно 4096 (1000h), каждая секция должна начинаться по адресу, кратном этому значению. Если первая секция находится в 401000h и его адрес равен 10 байтам, следующая секция должна начинаться в 402000h, даже если адресное пространство между ними останется неиспользованным.

Имя поля	Описание
FileAlignment	Размер выравнивания секций в файле. Например, если значение в этом поле равно 512 (200h), каждая секция должна начинаться на расстоянии от начала файла кратном 512 байтам. Если первая секция в файле находится по смещению 200h и ее размер 10 байт, следующая секция должна быть расположена со смещением 400h: пространство между смещениями 522 и 1024 будет неиспользовано/неопределено.
Minor/Major Subsystem Versions	Версия подсистемы win32. Если PE-файл спроектирован для Win32, версия подсистемы должна быть 4.0, в противном случае диалоговое окно не будет иметь 3D-вида.
SizeOfImage	Общий размер образа PE в памяти. Это сумма всех заголовков и секций, выровненных по SectionAlignment.
SizeOfHeaders	Размер всех заголовков + таблицы секций. То есть это значение равно размеру файла минус комбинированный размер всех секций в файле.
Subsystem	Указывает для какой из подсистем NT предназначен этот PE-файл. Для большинства win32-программ используется только два значения: Windows GUI и Windows CUI (консоль).
DataDirectory	Массив структур IMAGE_DATA_DIRECTORY. Каждая структура дает RVA важной структуры данных в PE-файле, например таблицы адресов импорта (IAT).

Во второй таблице: RVA – *relative virtual address* – относительный виртуальный адрес, расстояние для ссылающейся точки в виртуальном адресном пространстве (т.е. RVA, в отличие от абсолютного VA, не зависит от адреса загрузки образа, который может отличаться от ImageBase).

Примечание: все указанные структуры, имена полей и констант определены в заголовочных файлах Windows.

3. ИССЛЕДОВАНИЕ ФОРМАТА ИСПОЛНЯЕМОГО ФАЙЛА.

3.1. Анализ заголовка файла.

Рассмотрим простейшее приложение helloworld.exe, выводящее строку на экран консоли Win32:



Рис. 2 – Окно простейшего приложения Win32

Несмотря на то, что программа работает в консоли, это полноценное Win32 приложение. Для наглядности ниже приведен его листинг:

```
.386
.model flat,stdcall
option casemap:none

includelib kernel32.lib

SetConsoleTitleA PROTO :DWORD
GetStdHandle PROTO :DWORD
WriteConsoleA PROTO
:DWORD, :DWORD, :DWORD, :DWORD, :DWORD
ExitProcess PROTO :DWORD
Sleep PROTO :DWORD
.const
sConsoleTitle db 'My Nth Console Application',0
sWriteText db 'Hello World!!!!'
```



```
.code
Main PROC
    LOCAL hStdout :DWORD
    push offset sConsoleTitle
    call SetConsoleTitleA
    push -11
    call GetStdHandle
    mov hStdout, EAX

    push 0
    push 0
    push 16d
    push offset sWriteText
    push hStdout
    call WriteConsoleA
    push 2000d
    call Sleep
    push 0
    call ExitProcess
Main ENDP
end Main
```

Как можно увидеть из листинга программы, её основная работа состоит в получении дескриптора окна консоли с помощью API функции `GetStdHandle` и вывода строки в консоль с помощью API функции `WriteConsoleA`. API-функция `SetConsoleTitleA` устанавливает заголовок окна консоли, `Sleep` – обеспечивает задержку для фиксации результата выполнения программы на экране. Все API вызовы, используемые этой программой, импортируются из системной библиотеки `kernel32.dll`.

Работу с файлом будем производить с помощью редактора двоичных файлов `НIEW`. `НIEW` совмещает функции шестнадцатеричного редактора и дизассемблера, имеет возможности анализа исполняемых файлов (рис. 3).

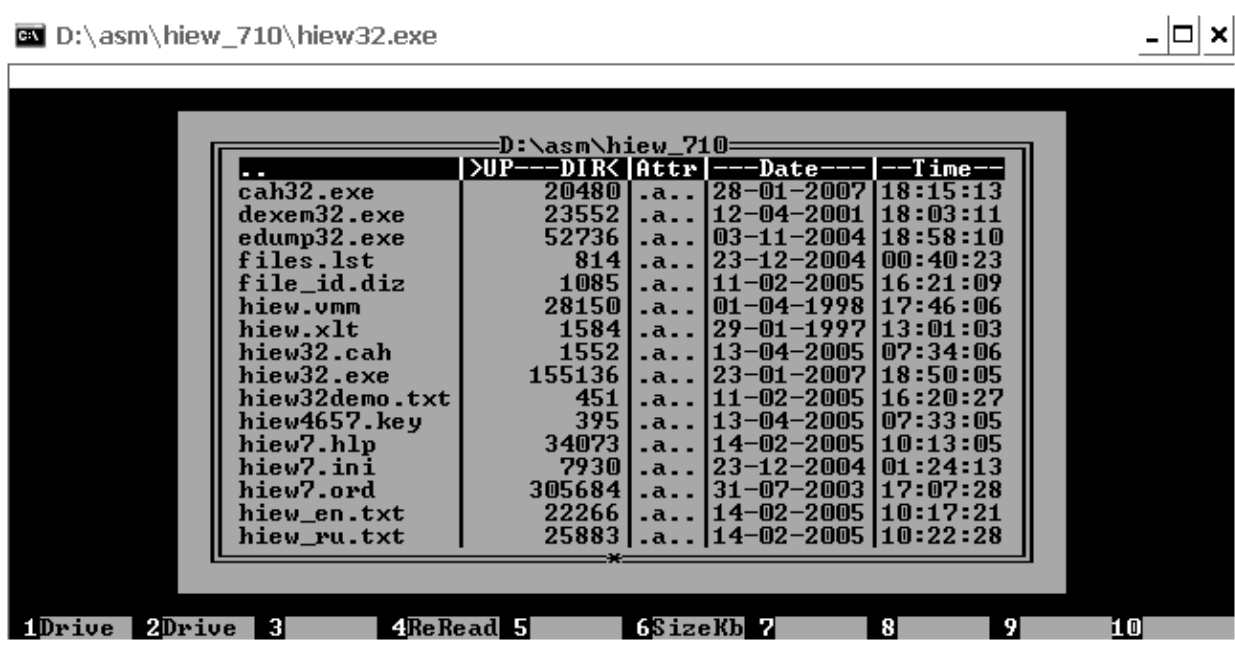


Рис. 3 – Главное окно программы HIEW

С помощью клавиш навигации находим исследуемый файл, нажимаем Enter для его открытия (рис. 5).



Рис. 4 – Текстовое представление файла helloworld.exe

По умолчанию файлы открываются для текстового просмотра/редактирования. Для перехода в шестнадцатеричный режим, необходимо нажать F4 (Mode) и выбрать Hex (рис. 5).

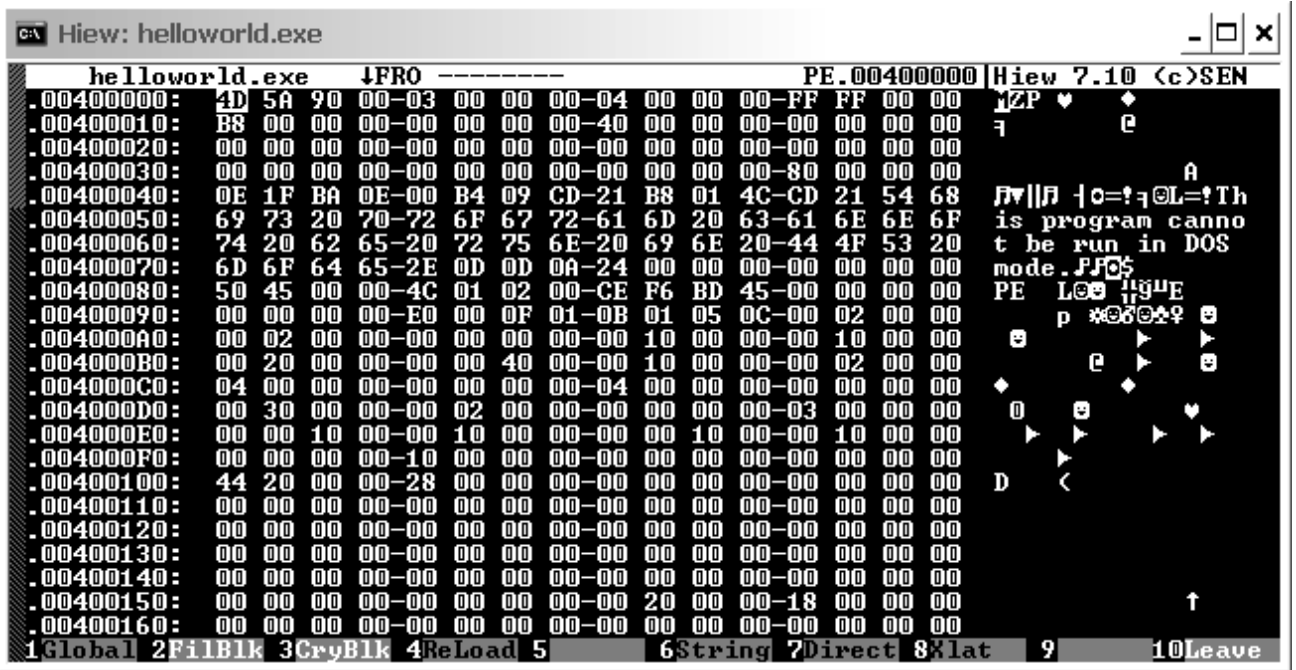


Рис. 5 Шестнадцатеричное представление файла helloworld.exe

HIEW автоматически показывает виртуальные смещения слева для всех исполняемых файлов, как будто образ уже загружен в память (виртуальные адреса отмечены точкой слева от смещения). Нажатие Alt+F1 переключает режим отображения смещений между виртуальными адресами и внутрифайловыми смещениями (рис. 6):

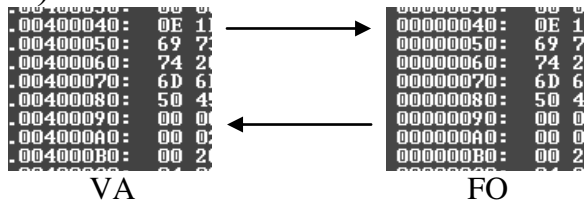


Рис. 6 – Виртуальные и адресные смещения

Имея теоретические сведения о формате PE-файлов и заголовков, можно кратко проанализировать файл (обратите внимание на то, что для указания любых значений используется обратный порядок байтов):

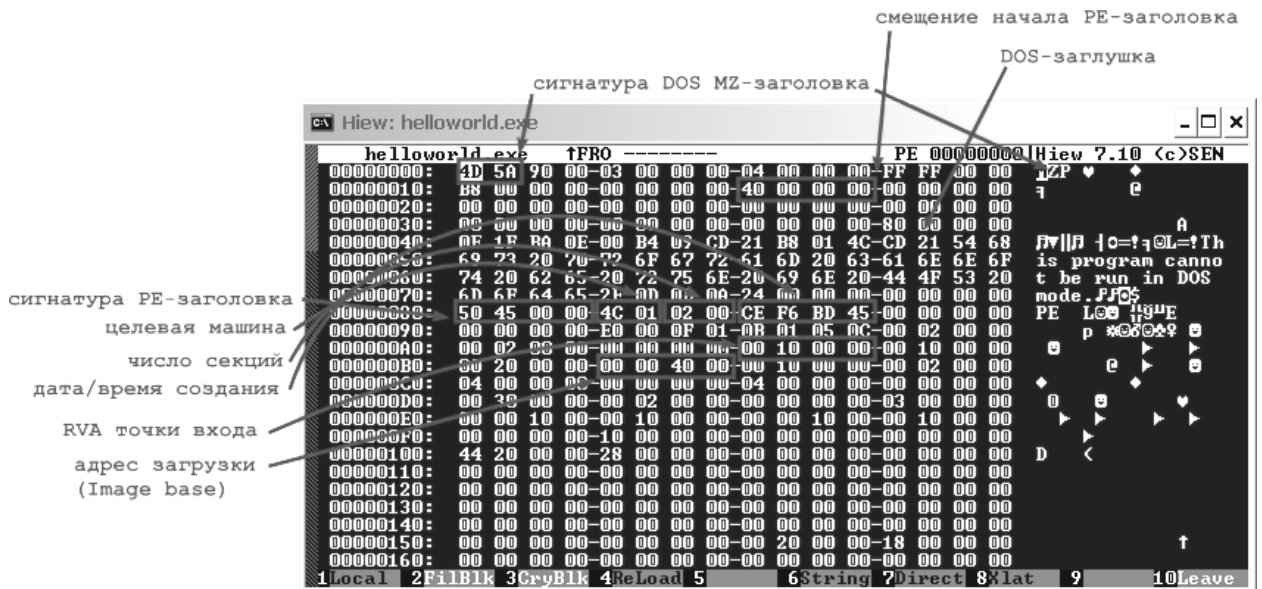


Рис. 7 –Элементы PE-файла в окне дизассемблера HIEW

Встроенный в HIEW анализатор заголовков позволяет проверить правильность найденных вручную значений. Для этого необходимо нажать F8 (Header):

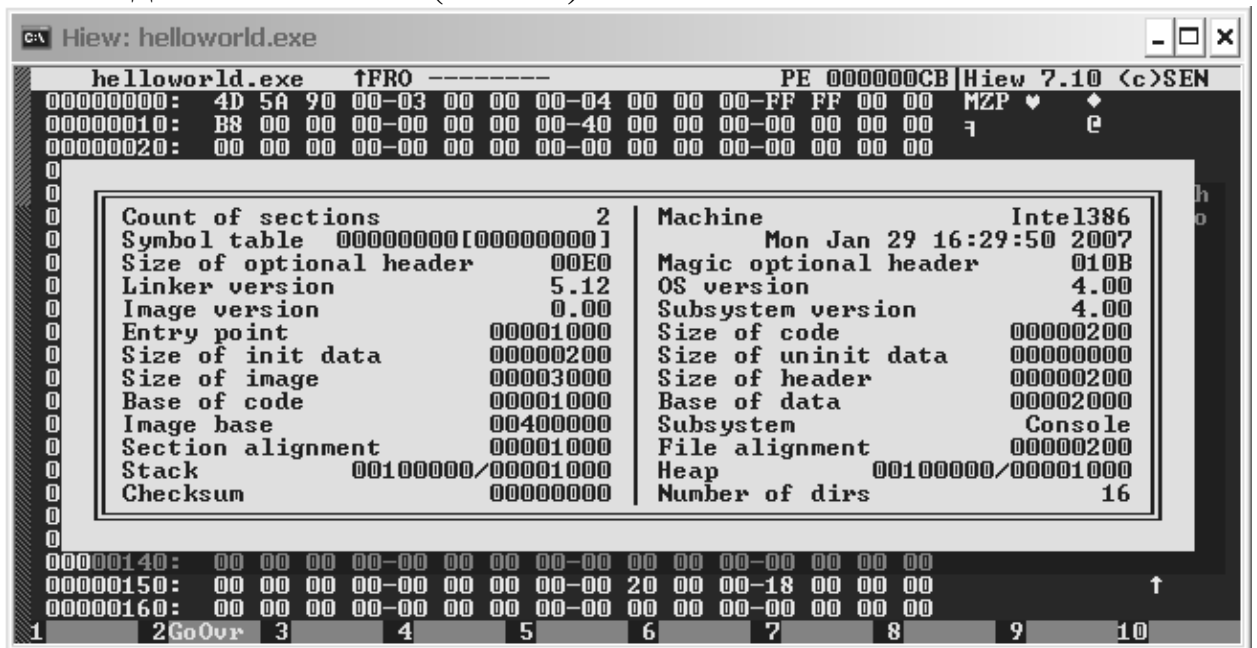


Рис. 8 – Встроенный анализатор

Полученных сведений достаточно, чтобы определить следующие параметры исполняемого файла:

- целевая машина – I386;
- число секций – 2;
- целевая подсистема – Windows CUI (Console);
- виртуальный адрес загрузки образа – 0x00400000;

д) виртуальный адрес и абсолютное внутрифайловое смещение точки входа:

- RVA = 1000h,
- VA = ImageBase + RVA = 0x00400100;

Чтобы найти абсолютное внутрифайловое смещение точки входа, необходимо нажать F5 (Entry), после чего происходит переход на точку входа.

е) внутрифайловое смещение точки входа равно 0x0200.

При нахождении на точке входа, нажатие F4 (Mode) и выбор режима Decode отобразит на экране дизассемблированный код приложения (рис. 9).

```

helloworld.exe  ↑FRO ----- a32 PE 00000200 | Hiew 7.10 <c>SEN
00000200: 55          push      ebp
00000201: 8BEC       mov      ebp,esp
00000203: 83C4FC    add      esp,-004 ;"H"
00000206: 6818204000 push     000402018 ;'My Nth Console Applicatio
0000020B: E82E000000 call     00000023E ----> (2)
00000210: 6AF5      push     0F5
00000212: E82D000000 call     000000244 ----> (3)
00000217: 8945FC    mov     [ebp-04],eax
0000021A: 6A00      push     000
0000021C: 6A00      push     000
0000021E: 6A10      push     010
00000220: 6833204000 push    000402033 ;'Hello, World!!!!'
00000225: FF75FC    push    d, [ebp-04]
00000228: E81D000000 call     00000024A ----> (5)
0000022D: 68B80B0000 push    000000BB8
00000232: E81F000000 call     000000256 ----> (6)
00000237: 6A00      push     000
00000239: E812000000 call     000000250 ----> (7)
0000023E: FF2510204000 jmp     SetConsoleTitleA ;kernel132
00000244: FF2500204000 jmp     GetStdHandle ;kernel132
0000024A: FF2504204000 jmp     WriteConsoleA ;kernel132
00000250: FF2508204000 jmp     ExitProcess ;kernel132
00000256: FF250C204000 jmp     Sleep ;kernel132
1Local 2FillBlk 3 4ReLoad 5OrdLdr 6byte 7Direct 8Xlat 9auto 10Leave

```

Рис. 9 – Дизассемблированный код приложения

Поскольку по смещению 0x0200 находится точка входа нашего приложения, первой инструкцией является операция помещения в стек указателя базы кадра стека:

```
00000200: 55          push      ebp
```

Системе безразлична конечная цель применения тех или иных данных. Это означает, что если точка входа указывает, скажем, на секцию с данными, то процессор будет трактовать, например, строковые данные как код и пытаться его исполнить, что может привести к непредсказуемым последствиям. Так же и дизассемблер будет пытаться дизассемблировать любой код, начиная с указанного байта, и если инструкция имеет, например, 3-хбайтный

опкод, а дизассемблирование начинается со второго байта, то результаты дизассемблирования будут неверными.

```

000001FF: 00558B      add     [ebp][-75],dl
00000202: EC        in     al,dx
00000203: 83C4FC      add     esp,-004 ;"H"
00000206: 6818204000 push   000402018 ;'My Nt

```

Рис. 10 – Результат неверного дизассемблирования

На рисунке показан пример неверного дизассемблирования. Точка входа имеет адрес 0x200, а дизассемблирование начинается с адреса 0x1FF трёхбайтной инструкцией. Для того, чтобы получить верное отображение файла в декодированном виде, следует поставить указатель на правильный байт (в нашем случае, по адресу 0x200) и нажать клавишу «/» (current offset at top). В этом случае дизассемблирование начнётся с правильного места.

3.2. Секции.

Как уже было выяснено из анализа заголовка, исследуемое приложение содержит две секции. Их описание содержится в таблице секций, описываемой массивом структур IMAGE_SECTION_HEADER.

```

IMAGE_SECTION_HEADER STRUCT
    Name1 db IMAGE_SIZEOF_SHORT_NAME dup(?)
    union Misc
        PhysicalAddress dd ?
        VirtualSize dd ?
    ends
    VirtualAddress dd ?
    SizeOfRawData dd ?
    PointerToRawData dd ?
    PointerToRelocations dd ?
    PointerToLinenumbers dd ?
    NumberOfRelocations dw ?
    NumberOfLinenumbers dw ?
    Characteristics dd ?
IMAGE_SECTION_HEADER ENDS

```

Нас интересуют имя секций, адреса их начала, а также их виртуальные и физические размеры. Иногда интерес также представляет поле Characteristics, используемое для назначения

атрибутов секций (например, возможность чтения или записи для секции).

Рассмотрим таблицу секций приложения. Таблица секций начинается по адресу .00400178 именем первой секции (.text). Вторая запись начинается по адресу .04001A0 именем второй секции (.rdata) (рис. 11)

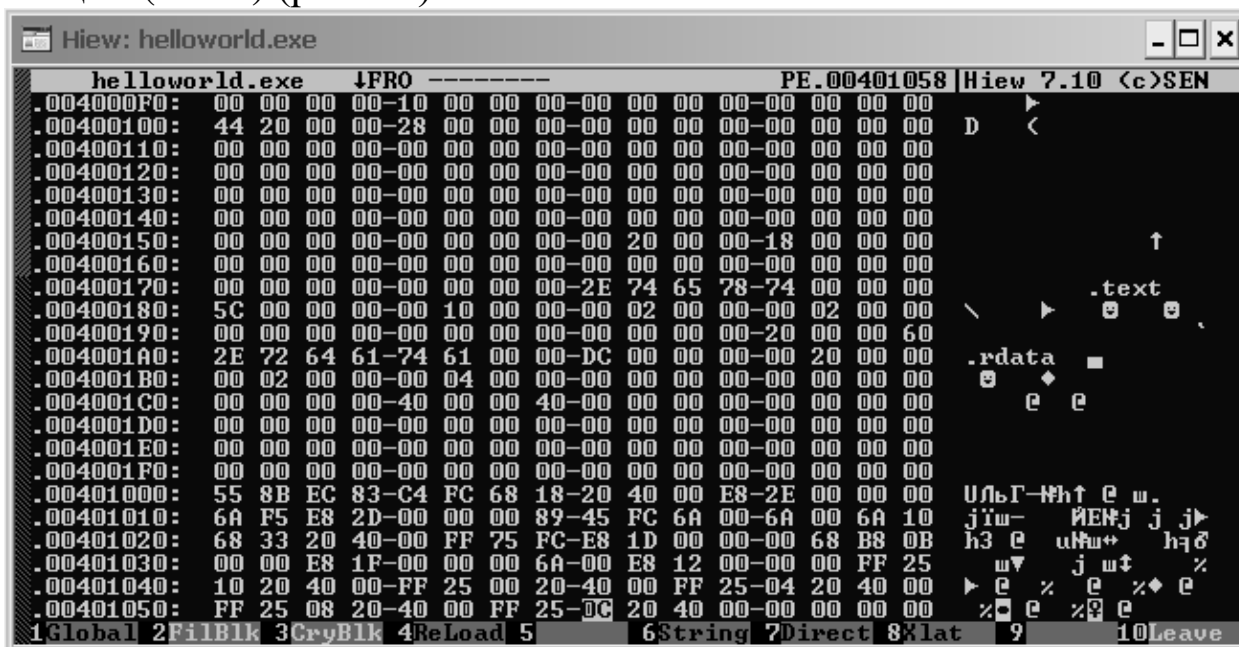


Рис. 11 – Таблицы секций приложения

Рассмотрим первую секцию. Следующие за её именем двойные слова представляют собой её виртуальный размер, относительный виртуальный адрес, физический адрес, внутрифайловое смещение и характеристики (как указано в структуре).

Выписываем необходимые данные:

- Имя секции: .text
- RVA начала: 0x00001000
- Внутрифайловое смещение начала: 0x00000200
- Виртуальный размер секции: 5Ch = 92 байт
- Физический размер секции: 200h = 512 байт

Аналогично записываем в отчет данные о второй секции:

- Имя секции: .rdata
- RVA начала: 0x00002000
- Внутрифайловое смещение начала: 0x00000400
- Виртуальный размер секции: DCh = 220 байт
- Физический размер секции: 200h = 512 байт

Встроенный в HIEW анализатор секций позволяет проверить правильность найденных данных. Для этого необходимо нажать F8 (Header) и F6 (ObjTbl) (рис. 12)

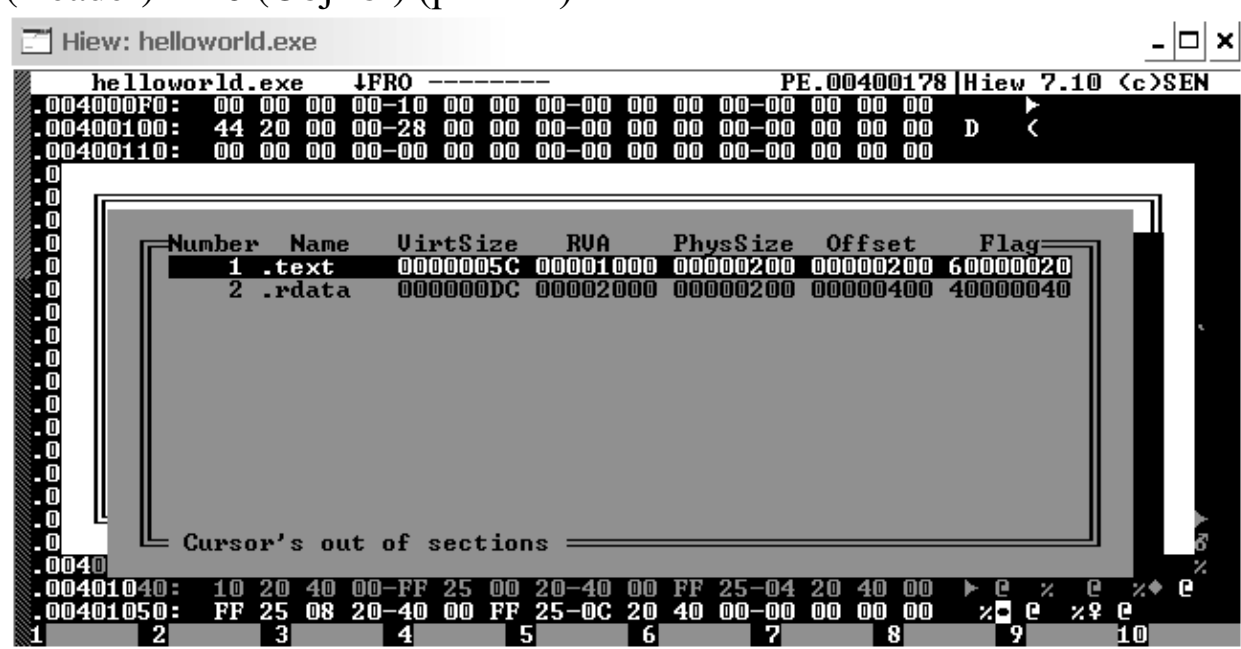


Рис. 12 – Результат работы встроенного в HIEW анализатора секций

3.3. Таблица импорта функций.

Импортируемая функция – это функция, которая находится не в модуле вызывающего приложения, но им вызывается. Функции импорта физически находятся в одной или более DLL. В модуле вызывающего находится только информация о функциях. Эта информация включает имена функций и имена DLL, в которых они находятся. На неё указывает член структуры опционального заголовка DataDirectory (см. описание опционального заголовка). Эти структуры содержат не только символы экспорта, но также символы импорта, ресурсы, исключения, безопасность и т.д. Каждый элемент массива директорий данных – структура IMAGE_DATA_DIRECTORY:

```
IMAGE_DATA_DIRECTORY STRUCT
    VirtualAddress dd ?
    isize dd ?
IMAGE_DATA_DIRECTORY ENDS
```


Это RVA начала структуры и её размер в байтах. Анализировать директории данных полностью нет необходимости. В нашем примере нам нужны только имена импортируемых функций. Их можно найти во второй секции (рис. 13)

```

helloworld.exe ↓FRO ----- PE.00402084 Hiew 7.10 (c)SEN
.004011E0: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.004011F0: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00402000: 98 20 00 00-A8 20 00 00-B8 20 00 00-C6 20 00 00  Ш и э ф
.00402010: 84 20 00 00-00 00 00 00-4D 79 20 4E-74 68 20 43  Д My Nth C
.00402020: 6F 6E 73 6F-6C 65 20 41-70 70 6C 69-63 61 74 69  onsole Applicati
.00402030: 6F 6E 00 48-65 6C 6C 6F-2C 20 57 6F-72 6C 64 21  on Hello, World?
.00402040: 21 21 21 00-6C 20 00 00-00 00 00 00-00 00 00 00  ??? 1
.00402050: CE 20 00 00-00 20 00 00-00 00 00 00-00 00 00 00  ;
.00402060: 00 00 00 00-00 00 00 00-00 00 00 00-98 20 00 00  ;
.00402070: A8 20 00 00-B8 20 00 00-C6 20 00 00-84 20 00 00  и э ф Д
.00402080: 00 00 00 00-25 02 53 65-74 43 6F 6E-73 6F 6C 65  SetConso
.00402090: 54 69 74 6C-65 41 00 00-34 01 47 65-74 53 74 64  TitleA 40GetStd
.004020A0: 48 61 6E 64-6C 65 00 00-95 02 57 72-69 74 65 43  Handle WriteC
.004020B0: 6F 6E 73 6F-6C 65 41 00-80 00 45 78-69 74 50 72  onsoleA A ExitPr
.004020C0: 6F 63 65 73-73 00 60 02-53 6C 65 65-70 00 6B 65  ocess Sleep ke
.004020D0: 72 6E 65 6C-33 32 2E 64-6C 6C 00 00-00 00 00 00  rnel32.dll
.004020E0: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.004020F0: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00402100: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00402110: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00402120: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00402130: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00402140: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
1Global 2FilBlk 3CryBlk 4ReLoad 5 6String 7Direct 8Lat 9 10Leave
  
```

Рис. 13 – Название импортируемых функций в коде программы

Каждая импортируемая функция содержит слово-индекс для быстрого нахождения функции в IAT загрузчиком и её имя. Наше приложение использует 5 импортируемых функций: SetConsoleTitleA, GetStdHandle, WriteConsoleA, ExitProcess, Sleep. Все они импортируются из библиотеки kernel32.dll. Список импортируемых библиотек можно увидеть, нажав F8 (Header) и F7 (Import) в HIEW (рис. 14)

```

helloworld.exe ↓FRO ----- PE.00402084 Hiew 7.10 (c)SEN
.004011E0: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.004011F0: 00 00
.00402000: 98 20
.00402010:
.00402020:
.00402030:
.00402040:
.00402050:
.00402060:
.00402070:
.00402080:
.00402090:
.004020A0:
.004020B0:
.004020C0:
.004020D0:
.004020E0:
.004020F0:
.00402100:
.00402110:
.00402120:
.00402130:
.00402140:
1Global 2FilBlk 3CryBlk 4ReLoad 5 6String 7Direct 8Lat 9 10Leave
  
```

Offset	Function Name	DLL Name
308	GetStdHandle	kernel32.dll
661	WriteConsoleA	kernel32.dll
128	ExitProcess	kernel32.dll
608	Sleep	kernel32.dll
549	SetConsoleTitleA	kernel32.dll

Рис. 14 – Список импортируемых функций.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ.

1. Проанализировать любой исполняемый файл в редакторе двоичных файлов HIEW, найти следующие его характеристики:

- а) целевая машина;
- б) число секций;
- в) целевая подсистема (Subsystem);
- г) виртуальный адрес загрузки образа (Image Base);
- д) виртуальный адрес и абсолютное внутрифайловое смещение точки входа;
- е) первую инструкцию, исполняемую загруженным приложением.

2. Для каждой секции найти её имя, относительный виртуальный адрес и внутрифайловое смещение начала секции, её виртуальный и физический размеры.

3. Найти в файле структуру таблицы адресов импорта. Выписать все API функции, импортируемые приложением, а также соответствующие им библиотеки.

5. СОДЕРЖАНИЕ ОТЧЁТА.

- 1) титульный лист;
- 2) скриншот программы своего варианта;
- 3) скриншоты основных этапов выполнения работы;
- 4) результаты выполнения работы.

6. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ.

- 1) Опишите структуру PE-файла.
- 2) Что такое DOS-заглушка?
- 3) Опишите формат заголовка PE-файла.
- 4) Что такое относительный виртуальный адрес?
- 5) Что такое точка входа? Почему при дизассемблировании необходимо правильно указывать её адрес?
- 6) Как определить количество секций программы и характеристики каждой из них?
- 7) Что такое импортируемая функция? Где они расположены?

7. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

- 1) Гордеев, А.В., Молчанов, А.Ю. Системное программное обеспечение – Спб.: Питер, 2001. – 736с. ил.
- 2) Рихтер, Дж. Windows для профессионалов: создание эффективных WIN32 приложений с учетом специфики 64-х разрядной версии Windows. – СПб.: Издательский дом «Питер», 2001.
- 3) Брой, М. Информатика. Основополагающее введение. Структуры систем и системное программирование.– М.: Диалог-МИФИ, 1996
- 4) Ганеев, Р.М., Проектирование интерфейса пользователя средствами Win32 API. – М.: Горячая линия–Телеком, 2001. – 336 с.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования «Юго-Западный
государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
О.Г. Локтионова
« 23 » 02 2023 г.



НАСТРОЙКА МЕЖСЕТЕВОГО ЭКРАНА В LINUX

Методические указания по выполнению лабораторных работ
для студентов укрупненной группы специальностей и направлений
подготовки 10.00.00

Курск 2023

УДК 004

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры
«Информационная безопасность» А.Л. Марухленко

Настройка межсетевого экрана в Linux: методические указания по выполнению лабораторной работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 33 с. Библиогр.: с. 33.

Излагаются методические указания по настройке межсетевого экрана IPTables на платформе Linux. Указывается порядок выполнения лабораторной работы, правила оформления, содержание отчета.

Методические указания по выполнению лабораторных работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____ . Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 148 . Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

1.	Введение.....	4
2.	Цель работы.....	5
3.	Задание.....	5
4.	Порядок выполнения работы.....	5
5.	Содержание отчета.....	5
6.	Теоретическая часть.....	6
6.1.	Основы архитектуры сетей.....	6
6.2.	IPTables.....	15
6.3.	Написание командных файлов.....	19
7.	Выполнение работы.....	22
8.	Варианты заданий.....	28
9.	Контрольные вопросы.....	32
10.	Библиографический список.....	33

ВВЕДЕНИЕ

В эпоху информатизации всех сфер жизни от умения защищать информацию зависит успех и отдельного мероприятия, и дела в целом. Компания, которая игнорирует этот факт, подвергает себя риску потерять конкурентоспособность из-за получения информационного ущерба в виде утечки информации или поломки информационной системы.

Одним из приоритетных направлений защиты информационной системы является защита от несанкционированного доступа, получить который можно, к примеру, используя открытые порты на компьютере, принадлежащем информационной системе. Чтобы защитить компьютер от вирусного ПО и других вредоносных программ, необходимо использовать межсетевые экраны, иначе называемые фаерволами, которые предназначены для закрытия неиспользуемых вами портов, через которые злоумышленник может получить доступ к компьютеру и информационной системе в целом.

Межсетевые экраны могут быть двух типов: аппаратные и программные. Одним из самых популярных экземпляров программного экрана для платформы Linux является IPTables, настройка и установка которого позволит понять основные принципы функционирования межсетевых экранов.

ЦЕЛЬ РАБОТЫ

Цель лабораторной работы – освоение методики настройки межсетевого экрана с помощью утилиты IPTables на платформе Linux.

ЗАДАНИЕ

Используя утилиту IPTables, встроенную в систему Linux с ядром версии 2.4 и выше, настроить межсетевой экран защищающий от наиболее распространенных атак.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретическую часть;
2. Произвести настройку межсетевого экрана в соответствии с методическими требованиями, подкрепляя каждый из этапов работы скриншотом;
3. Составить отчет;

СОДЕРЖАНИЕ ОТЧЕТА

1. Титульный лист;
2. Теоретическая часть;
3. Текст командного файла и скриншоты;
4. Вывод;

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Основы архитектуры сетей

Прежде чем перейти к подлинному пониманию сетевой безопасности, необходимо понять архитектуру сетей. В данном разделе приведен краткий обзор сетевых концепций и терминов. Знакомство с ними поможет вам понять основы протокола TCP/IP.

Как вы, вероятно, знаете, конструкцию каждой сети можно разделить на семь логических частей, каждая из которых решает определенную часть коммуникационной задачи. Эта семиуровневая конструкция называется Эталонной моделью взаимосвязи открытых систем (ВОС). Она была разработана Международной организацией по стандартизации (ISO) для представления логической модели описания сетевых коммуникаций, и она помогает поставщикам стандартизовать оборудование и программное обеспечение. В табл. 1 проиллюстрирована эталонная модель ВОС и приведены примеры каждого уровня.

Физический уровень

Этот уровень представляет реальную физическую среду передачи данных. Для различных типов среды применяются разные стандарты. Например, коаксиальный кабель, неэкранированная витая пара и волоконно-оптический кабель предназначены для различных целей: коаксиальный кабель используется в более старых ЛВС, а также для подключения к Интернету через сети кабельного ТВ, витая пара - для внутренней кабельной разводки, в

то время как оптоволокну обычно применяют для протяженных соединений с высокой пропускной способностью.

Таблица 1 – Эталонная модель ВОС

Номер уровня модели ВОС	Название уровня	Примеры протоколов
Уровень 7	Прикладной уровень	DNS, FTP, HTTP, SMTP, SNMP, Telnet
Уровень 6	Уровень представления	XDR
Уровень 5	Уровень сеанса	RPC
Уровень 4	Транспортный уровень	NetBIOS, TCP, UDP
Уровень 3	Сетевой уровень	ARP, IP, IPX, OSPF
Уровень 2	Канальный уровень	Arcnet, Ethernet, Token ring
Уровень 1	Физический уровень	Коаксиальный кабель, оптоволокну, витая пара

Канальный уровень

Этот уровень относится к различным частям оборудования сетевых интерфейсов. Он помогает кодировать данные и помещать их в физическую среду передачи. Он также позволяет устройствам идентифицировать друг друга при попытке взаимодействия с другим узлом. Примером адреса канального уровня служит MAC-адрес сетевой платы (Medium Access Control - управление доступом к среде передачи). MAC-адрес является числом, которое уникальным образом идентифицирует плату компьютера в сети. В 1970-80-х годах корпорации использовали много различных типов стандартов канального уровня, определенных по большей части их поставщиками оборудования. В наше время большинство

организаций используют Ethernet, так как он широко распространен и недорог.

Сетевой уровень

Этот уровень является первой частью видимого взаимодействия с сетями TCP/IP. Сетевой уровень дает возможность взаимодействовать через различные физические сети с помощью вторичного уровня идентификации. В сетях TCP/IP для этого используется IP-адрес. IP-адрес на компьютере помогает осуществлять маршрутизацию данных при передаче из одного места в другое в сети и через Интернет. Этот адрес является уникальным числом для идентификации компьютера в IP-сети - ни одна другая машина в Интернете не может иметь такой адрес. Это справедливо для обычных открыто маршрутизируемых IP-адресов.

При возникновении сетевых проблем, для их решения могут использоваться MAC-адреса, а также сетевые анализаторы. С их помощью можно проследить источник проблемного сетевого трафика. MAC-адреса обычно регистрируются серверами DHCP в Windows или межсетевыми экранами, поэтому можно сопоставить MAC-адреса с определенным IP-адресом или именем машины.

Транспортный уровень

Этот уровень обеспечивает доставку пакета данных из точки А в точку В. На этом уровне располагаются протоколы TCP и UDP. TCP (Transmission Control Protocol - протокол управления передачей) по сути обеспечивает согласованность отсылки пакетов и их приема на другом конце. Он позволяет исправлять ошибки на

уровне битов, повторно передавать потерянные сегменты и переупорядочивать фрагментированный трафик и пакеты. UDP (User Datagram Protocol - пользовательский дейтаграммный протокол) является менее тяжеловесной схемой, используемой для потоков мультимедийных данных и кратких взаимодействий с небольшими накладными расходами, такими как запросы DNS. Большинство межсетевых экранов оперируют на транспортном и сетевом уровнях.

Уровень сеанса

Уровень сеанса обслуживает в основном установление соединения и его последующее закрытие. Иногда на этом уровне выполняется аутентификация, для того чтобы установить, кому разрешено участвовать в сеансе.

Уровень представления

Этот уровень обеспечивает определенное кодирование и декодирование, требующееся для представления данных в формате, понятном получателю. Считается, что прикладной уровень и уровень представления по сути совпадают.

Прикладной уровень

Заключительный уровень, на котором прикладные программы (FTP, HTTP, SMTP и т.п.) получают данные. На этом уровне в дело вступает некоторая программа, обрабатывающая реальные данные из пакетов. К сожалению, этот уровень является наиболее уязвимым для угроз.

Сети TCP/IP

Этот протокол был изобретен военным исследовательским агентством, DARPA, для обеспечения бесперебойной работы сетей. Преследовалась цель создания сети, способной выдержать отказ множества линий связи в случае катастрофического события, такого как ядерный удар. Традиционные способы передачи этого обеспечить не могли. В TCP/IP предложен способ "пакетирования" данных, позволяющий им находить собственный путь через сеть. Тем самым была создана первая отказоустойчивая сеть.

Только с появлением Интернета в начале 1990-х TCP/IP превратился в стандарт передачи данных. Это привело к снижению цен на оборудование для IP-сетей, и также значительно облегчило межсетевое взаимодействие.

TCP/IP позволяет взаимодействующим узлам устанавливать соединение и затем проверять, когда передача данных начинается и завершается. В сети TCP/IP передаваемые данные разбиваются на фрагменты, называемые пакетами, и помещаются в последовательность "конвертов", каждый из которых содержит определенную информацию для следующего протокольного уровня. Пакеты помечаются 32-битными порядковыми номерами, чтобы даже в случае прихода в неправильном порядке передаваемые данные можно было собрать заново. Когда пакет пересекает различные части сети, каждый уровень открывается и интерпретируется, а затем оставшиеся данные передаются дальше согласно полученным инструкциям. Когда пакет данных прибывает

в место назначения, реальные данные, или полезная нагрузка, доставляются приложению.

В табл. 2 показано, как некоторые сетевые протоколы инкапсулируют данные.

Таблица 2 – Пример пакета данных TCP/IP

Протокол	Содержимое	Уровень модели ВОС
Ethernet	MAC-адрес	Канальный
IP	IP-адрес	Сетевой
TCP	Заголовок TCP	Транспортный
HTTP	Заголовок HTTP	Прикладной
Прикладные данные	Web-страница	Данные

Можно видеть, что на внешнем "конверте" для наших данных написан адрес Ethernet. Он идентифицирует пакет в сети Ethernet. Внутри этого конверта находится сетевая информация, а именно, IP-адрес; еще глубже находится транспортный уровень, который устанавливает соединение и закрывает его. Затем располагается прикладной уровень с заголовком HTTP, сообщающим web-навигатору, как форматировать страницу. Наконец, мы доходим до реальной полезной нагрузки пакета - содержимого web-страницы. Этот пример иллюстрирует многоуровневую природу сетевых коммуникаций.

При использовании протокола TCP/IP взаимодействие между двумя узлами сети подразделяется на несколько фаз (рис.1). Не вдаваясь в детали, касающиеся сервера доменных имен (DNS), и

предполагая, что используются IP-адреса, а не имена хостов, в качестве первой фазы выделим порождение ARP-запроса (Address Resolution Protocol - протокол разрешения адресов) для поиска адреса Ethernet, соответствующего IP-адресу, с которым пытаются взаимодействовать. ARP преобразует IP-адрес в MAC-адрес сети

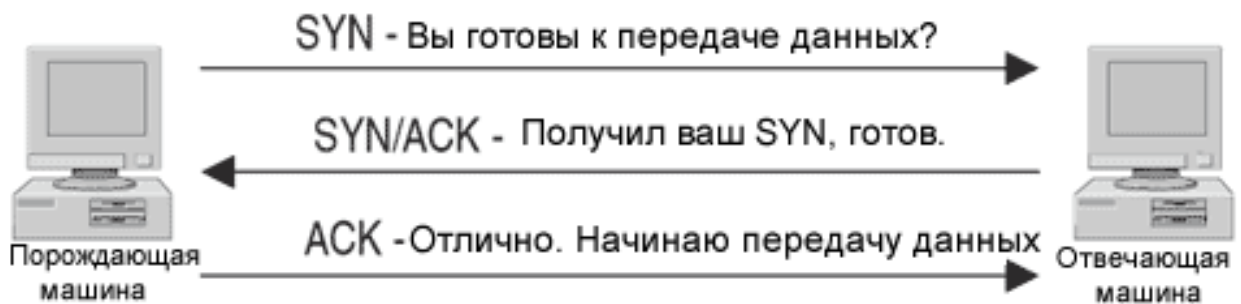


Рисунок 1 – Трехходовое квитирование установления связи

Ethernet. Теперь, когда мы можем общаться с целевой машиной по протоколу IP, для формирования сеанса между машинами с помощью протокола TCP осуществляется трехходовое взаимодействие. Машина, собирающаяся послать данные другой машине, посылает пакет SYN для синхронизации или инициирования передачи. Пакет SYN, по сути, говорит: "Вы готовы к передаче данных?" Если другая машина готова принять соединение от первой, она посылает SYN/ACK, что означает: "Подтверждаю получение вашего пакета SYN, я готова." Наконец, порождающая машина отправляет пакет ACK обратно, говоря тем самым: "Отлично. Начинаю посылать данные." Такая процедура называется трехходовым квитированием установления связи в TCP. Если какой-либо из трех ходов не выполнится, то соединение не

будет установлено. Когда все данные посланы, одна из сторон посылает второй стороне соединения пакет FIN. Та отвечает пакетом FIN/ACK и сама посылает пакет FIN, в ответ на который посылается последний пакет FIN/ACK для закрытия сеанса TCP/IP.

Наиболее важным для межсетевых экранов является блокирование внешнего трафика путем запрета пакетов SYN, направляемых извне на машины внутри сети. В результате внутренние машины могут общаться с внешним миром и инициировать соединения, но внешним машинам не удастся открыть сеанс.

Межсетевые экраны

В Linux существует несколько встроенных экранирующих приложений: Iptables в версиях ядра 2.4x, Ipchains в 2.2x и Ipfwadm в ядре версии 2.0. Большинство межсетевых экранов на платформе Linux делают свое дело, используя одну из этих служебных программ уровня ядра.

Все три упомянутых приложения действуют аналогичным образом. У межсетевых экранов обычно имеется два или больше сетевых интерфейсов, и под Linux это достигается наличием в компьютере двух или большего количества сетевых плат. Один интерфейс обычно соединяется с внутренней ЛВС – доверенный; другой интерфейс предназначен для общедоступной стороны (ГВС). Все пакеты, проходящие через машину, подвергаются обработке определенными фильтрами.

Межсетевые экраны могут фильтровать пакеты на нескольких различных уровнях. Они могут анализировать IP-адреса и блокировать трафик, приходящий от определенных машин или сетей, проверять заголовок TCP и определять его состояние, и на более высоких уровнях анализировать приложение или номер порта TCP/UDP. Межсетевые экраны можно конфигурировать для отбрасывания целых категорий трафика, таких как ICMP. Пакеты типа ICMP, такие как ping, обычно отбрасываются межсетевыми экранами, поскольку они часто используются для исследования сети и атак на доступность.

Существует два метода настройки межсетевых экранов. Можно в качестве исходного принять положение "разрешить все" и затем задавать поведение, которое требуется блокировать, или же начать с положения "запретить все", после чего специфицировать то, что следует разрешить (допустимое поведение пользователей). Безусловно, предпочтительным является исходное положение "запретить все", поскольку при этом автоматически блокируются все потоки данных, если только они не разрешены явным образом. Используя подход "запретить все", вы автоматически блокируете все, что не считается добропорядочной активностью.

Тип конфигурации "разрешить все" может иметь смысл в крайне либеральной среде, где накладные расходы на дополнительные разрешающие строки превышают ценность информации в сети (пример - некоммерческий или чисто информационный сайт). Но для большинства организаций подход

"запретить все" более безопасен, однако его применение не делает вашу сеть полностью безопасной.

IPTables

Механизм отслеживания состояний

Важной особенностью iptables является механизм определения состояний (`connection tracking`, `nf_conntrack`) — специальная подсистема, отслеживающая состояния соединений и позволяющая использовать эту информацию при принятии решений о судьбе отдельных пакетов.

Определение состояния соединения порою бывает довольно сложной задачей. Скажем, в случае TCP все относительно просто — контроль состояний реализован средствами самого протокола, установлены специальные процедуры открытия и закрытия соединения. А вот в протоколе UDP для этого специальных процедур не предусмотрено, поэтому UDP-соединением с точки зрения `conntrack` является поток пакетов, следующих с интервалом, не превышающим заданный тайм-аут (по умолчанию 30 секунд), с одного и того же порта одного хоста, на один и тот же порт другого хоста.

Заметим, что классификация пакетов по отношению к соединениям, реализуемая системой `conntrack`, во многих случаях отличается от официального описания сетевых протоколов. Например, с точки зрения критерия `conntrack`, TCP-пакет SYN/ACK (отвечающий на SYN) — уже часть существующего сеанса, а по

определению TCP такой пакет — всего лишь элемент открытия сеанса.

Существует также понятие «связанных соединений». Например, когда в ответ на UDP-пакет с нашего хоста удаленный хост отвечает ICMP-пакетом `icmp-port-unreachable`, формально этот ответ является отдельным соединением, так как использует совсем другой протокол.

В некоторых случаях целесообразно отключить отслеживание состояния соединений. Например, если ваш сервер находится под (D)Dos-атакой типа флуд, и вам удалось локализовать ее источники, отслеживать соединения с атакующих хостов и тратить для этого ресурсы своей системы явно не имеет смысла.

Критерий состояния соединения

При помощи критерия `conntrack` вы можете классифицировать пакеты на основании их отношения к соединениям. В частности, состояние `NEW` позволяет выделять только пакеты, открывающие новые соединения, состояние `ESTABLISHED` — пакеты, принадлежащие к установленным соединениям, состоянию `RELATED` соответствуют пакеты, открывающие новые соединения, логически связанные с уже установленными (например, соединение данных в пассивном режиме FTP). Состояние `INVALID` означает, что принадлежность пакета к соединению установить не удалось.

Например, одним простым правилом

```
iptables -I INPUT -m conntrack --ctstate ESTABLISHED -j
ACCEPT
```

вы можете обеспечить корректное пропускание всех входящих пакетов, принадлежащих установленным соединениям, и сконцентрироваться только на фильтрации новых соединений.

Заменив в предыдущем правиле ESTABLISHED на ESTABLISHED,RELATED и подгрузив соответствующие модули ядра, вы автоматически обеспечите корректную фильтрацию протоколов, использующих связанные соединения — FTP, SIP, IRC, H.323 и других.

Использование

Идея, лежащая в основе Iptables и Ipchains, состоит в создании каналов входных данных и их обработке в соответствии с набором правил (конфигурацией вашего межсетевого экрана) с последующей передачей в выходные каналы. В Iptables правила располагаются в таблицах, а внутри таблиц - в цепочках. Основными цепочками, используемыми в Iptables, служат:

1. Input.
2. Forward.
3. Prerouting.
4. Postrouting.
5. Output.

Общий формат инструкций Iptables таков:

Iptables команда спецификация_правил расширения,

где команда, спецификация_правил и расширения - это одна или несколько допустимых опций. В табл. 4 перечислены команды Iptables, а табл. 3 содержит спецификации правил Iptables.

Существуют и другие команды, опции, но мы перечислили самые распространенные операции. Весь список команд можно найти в оперативной справке Iptables, набрав `man iptables` в командной строке.

Таблица 3 – Спецификации правил IPTables

Спецификация правила	Описание	
-p протокол	Задаёт протокол, которому соответствует правило. Допустимыми типами протоколов являются icmp, tcp, udp и all	
-s адрес/маска	Задаёт определенный адрес или сеть для соответствия. Используется стандартная нотация с косой чертой для указания диапазона IP-адресов	
-j цель	Указывает, что делать с пакетом, если он соответствует спецификациям. Допустимыми опциями для цели являются:	
	DROP	Отбрасывает пакет без всяких дальнейших действий.
	REJECT	Отбрасывает пакет и посылает в ответ пакет с уведомлением об ошибке.
	LOG	Протоколирует пакет в файле.
	MARK	Помечает пакет для дальнейших действий.
	TOS	Изменяет поле TOS (тип обслуживания).
	MIRROR	Меняет местами исходный и целевой адреса и посылает пакеты обратно, по сути "отражая" их назад отправителю.
	SNAT	Трансляция исходных сетевых адресов. Эта опция применяется при выполнении трансляции сетевых адресов. Исходный адрес преобразуется в другое статическое значение, определенное с помощью ключа - to-source.
	DNAT	Трансляция целевых сетевых адресов. Данная опция аналогична предыдущей, но

Спецификация правила	Описание	
		применяется к целевым адресам.
	MASQ	Маскарад с помощью общедоступного IP-адреса.
	REDIRECT	Перенаправляет пакет.

Написание командных файлов

Часто требуется автоматизировать некоторый процесс или иметь одну команду для запуска нескольких инструкций. Применительно к межсетевому экрану обычно желательно,

Таблица 4 – команды IPTables

Команда	Описание
-A цепочка	Добавляет в конец указанной цепочки одно или несколько правил, заданных в инструкции вслед за командой
-I цепочка номер_правила	Вставляет правила в позицию с заданным номером в указанной цепочке. Это полезно, если вы хотите перекрыть правила, заданные ранее
-D цепочка спецификация_правил	Удаляет из указанной цепочки специфицированные номером или текстом правила
-R цепочка номер_правила	Заменяет правило в позиции с заданным номером в указанной цепочке
-L цепочка	Выдает все правила в цепочке. Если цепочка не задана, выдаются все цепочки
-F цепочка	Сбрасывает все правила в цепочке, по сути ликвидируя конфигурацию вашего межсетевого экрана. Это полезно в начале конфигурирования, чтобы гарантировать отсутствие существующих правил, и избегания конфликтов с новыми
-Z цепочка	Обнуляет все счетчики пакетов и байтов в указанной цепочке
-N цепочка	Создает новую цепочку с заданным именем
-X цепочка	Удаляет указанную цепочку. По умолчанию удаляются все цепочки
-P цепочка политика	Задаёт политику для указанной цепочки

чтобы все его команды выполнялись при загрузке системы. Лучшим способом сделать это является написание командного файла. Это простой текстовый файл, содержащий команду или список команд. Интерпретатор командного языка выполняет команды, когда он вызывается пользователем, набравшим имя файла.

Чтобы создать командный файл, откройте сначала текстовый редактор, такой как vi или EMACS, и введите свои команды.

Введите в самом верху строку, которая выглядит следующим образом:

```
#!/bin/bash
```

Данная строка сообщает, какой интерпретатор использовать для выполнения команд. Вы должны иметь его в своей ОС, а команды, помещенные в файл, должны быть его корректными командами. Приведенный пример задает маршрутное имя интерпретатора bash в Mandrake Linux. Можно использовать другой интерпретатор, например, Tcsh или Csh. Просто задайте в первой строке его маршрутное имя. Затем сохраните файл.

Сделайте файл исполнимым, чтобы интерпретатор мог выполнить его как программу. Это делается с помощью команды chmod. Введите

```
chmod 700 имя_командного_файла
```

Такой режим доступа делает файл читаемым, записываемым и исполнимым.

Чтобы выполнить командный файл, наберите его имя в командной строке. (В `bash` необходимо задать `./` перед именем файла, расположенного в текущем каталоге.) После нажатия клавиши ввода должны выполняться команды из файла.

Вы должны находиться в том каталоге, где размещен командный файл, или задать его маршрутное имя. Чтобы он выполнялся из любого места, можно добавить этот каталог в переменную окружения `PATH` или поместить файл в один из каталогов, фигурирующих в значении `$PATH`.

ВЫПОЛНЕНИЕ РАБОТЫ

Можно вводить команды интерактивно, по одной, чтобы сразу видеть результаты. Можно также поместить их в командный файл и выполнять его при загрузке системы, поднимая тем самым межсетевой экран (см. врезку о написании командных файлов). Набирайте их точно так же, как показано, сохраняя, в частности, регистр букв.

В следующем примере предполагается, что диапазон IP-адресов 192.168.0.1 - 192.168.0.254 принадлежит вашей подсети ЛВС, к которой подключен интерфейс eth1, и что интерфейс eth0 является соединением с Интернетом или ГВС.

1. Начните с удаления всех существующих правил с помощью команды Flush:

```
iptables -F FORWARD
```

Это стирает все правила цепочки FORWARD, являющейся основной "воронкой" для всех пакетов, пытающихся пройти через межсетевой экран.

2. Очистите другие цепочки:

```
iptables -F INPUT
```

```
iptables -F OUTPUT
```

Эти команды стирают все правила на пути пакетов, направленных в локальную машину, и в выходной цепочке.

```

Файл  Правка  Вид  Поиск  Терминал  Справка
root@dhcpc4:~# iptables -F FORWARD
root@dhcpc4:~# iptables -F INPUT
root@dhcpc4:~# iptables -F OUTPUT
root@dhcpc4:~#

```

3. Поместите стандартную инструкцию "запретить все" в самое начало.

```
iptables -P FORWARD DROP
```

```
iptables -A INPUT -i eth0 -j DROP
```

4. Решение о допуске фрагментированных пакетов в Iptables необходимо оформить явным образом:

```
iptables -A FORWARD -f -j ACCEPT
```

```

root@dhcpc4:~# iptables -P FORWARD DROP
root@dhcpc4:~# iptables -A INPUT -i eth0 -j DROP
root@dhcpc4:~# iptables -A FORWARD -f -j ACCEPT

```

5. Существует два типа распространенных атак, которые необходимо сразу заблокировать. Одна из них называется подделкой (подделываются заголовки IP-пакетов, чтобы казалось, будто внешний пакет имеет внутренний адрес). Делая это, злоумышленник может попасть в вашу сеть, даже если вы используете собственные IP-адреса. Другой тип атаки реализуется отправкой потока пакетов на широковещательный адрес сети, чтобы перегрузить ее. Это называется штормовой атакой. Атаки перечисленных типов можно блокировать с помощью двух простых инструкций:

```
iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -j DROP
```

```
iptables -A FORWARD -p icmp -i eth0 -d 192.168.0.255 -j DROP
```

```

root@dhcpc4:~# iptables -A FORWARD -i eth0 -j ACCEPT
root@dhcpc4:~# iptables -A FORWARD -s 192.168.0.0/24 -i eth0 -j DROP
root@dhcpc4:~# iptables -A FORWARD -p icmp -i eth0 -d 192.168.0.255 -j DENY
iptables v1.4.14: Couldn't load target `DENY':No such file or directory

Try `iptables -h' or 'iptables --help' for more information.
root@dhcpc4:~# iptables -A FORWARD -p icmp -i eth0 -d 192.168.0.255 -j DROP
root@dhcpc4:~# █

```

Первая инструкция предписывает отбрасывать все пакеты, приходящие из Интернет-интерфейса eth0 с внутренним адресом 192.168.0.0/24. По определению ни один пакет не должен приходить из недоверенного интерфейса с внутренним, собственным исходным адресом. Вторая инструкция отвергает все приходящие извне на адрес внутренней сети широковещательные пакеты протокола ICMP.

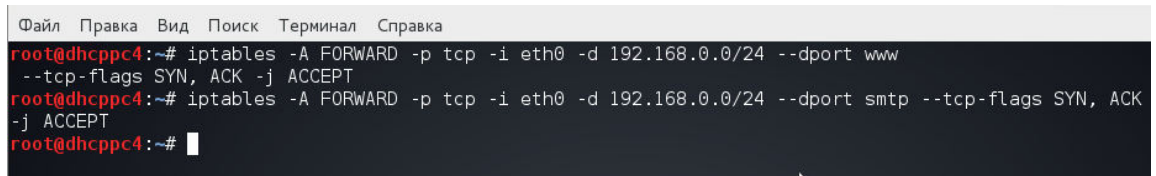
6.Вы, как правило, желаете принимать входящие потоки данных, поступающие по соединениям, инициированным изнутри (например, кто-то просматривает web-страницу). Пока соединение, инициированное изнутри, поддерживается - все, наверное, хорошо. Можно, однако, ограничить тип пропускаемого внутрь трафика. Предположим, вы хотите разрешить сотрудникам только web-доступ и электронную почту. Можно определить типы трафика для прохода внутрь и только для уже инициированного соединения. Следующая инструкция разрешает потоки данных по web-протоколу HTTP и почтовому протоколу SMTP на основе этого критерия.

```

iptables -A FORWARD -p tcp -i eth0 -d 192.168.0.0/24 --dport
www --tcp-flags SYN, ACK -j ACCEPT

```

```
iptables -A FORWARD -p tcp -i eth0 -d 192.168.0.0/24 --dport
smtp --tcp-flags SYN, ACK -j ACCEPT
```

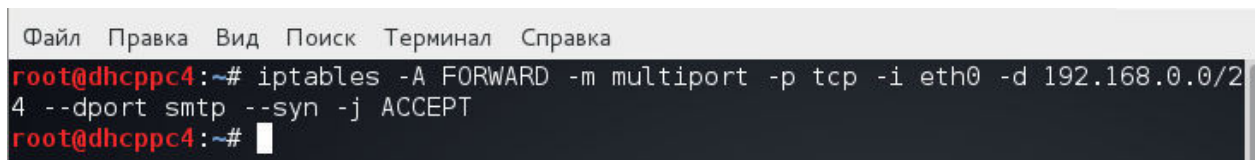


```
Файл Правка Вид Поиск Терминал Справка
root@dhcppc4:~# iptables -A FORWARD -p tcp -i eth0 -d 192.168.0.0/24 --dport www
--tcp-flags SYN, ACK -j ACCEPT
root@dhcppc4:~# iptables -A FORWARD -p tcp -i eth0 -d 192.168.0.0/24 --dport smtp --tcp-flags SYN, ACK
-j ACCEPT
root@dhcppc4:~# █
```

Флаг `-m multiport` извещает Iptables, что вы будете выдавать инструкции сопоставления с портами. Конструкция `- sports` разрешает только трафик электронной почты и web-навигации. Опция `- syn` разрешает пакеты SYN с неустановленным флагом ACK (и RST), то есть инициирование соединений TCP, а предшествующий восклицательный знак инвертирует смысл этого условия. В результате допускаются только пакеты, не иницирующие соединений.

7. Чтобы можно было принять входящие соединения извне только на определенных портах (например, входящие соединения электронной почты с вашим почтовым сервером), и разрешить из этих же портов направлять трафик вовне, используйте следующие инструкции:

```
iptables -A FORWARD -m multiport -p tcp -i eth0 -d
192.168.0.0/24 --dport smtp --syn -j ACCEPT
```



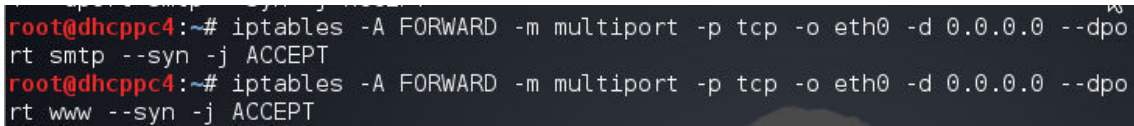
```
Файл Правка Вид Поиск Терминал Справка
root@dhcppc4:~# iptables -A FORWARD -m multiport -p tcp -i eth0 -d 192.168.0.0/2
4 --dport smtp --syn -j ACCEPT
root@dhcppc4:~# █
```

Предполагается, что IP-адресом почтового сервера служит 192.168.0.2. Флаги --dport и --sport разрешают только почтовый трафик SMTP.

8. Можно разрешить пользователям инициировать исходящие соединения и передавать по ним данные, но только для определенных протоколов. Именно здесь вы можете запретить применение FTP и других необязательных программ.

```
iptables -A FORWARD -m multiport -p tcp -o eth0 -d
0.0.0.0 --dport www --syn -j ACCEPT
```

```
iptables -A FORWARD -m multiport -p tcp -o eth0 -d
0.0.0.0 --dport smtp --syn -j ACCEPT
```



```
root@dhcppc4:~# iptables -A FORWARD -m multiport -p tcp -o eth0 -d 0.0.0.0 --dpo
rt smtp --syn -j ACCEPT
root@dhcppc4:~# iptables -A FORWARD -m multiport -p tcp -o eth0 -d 0.0.0.0 --dpo
rt www --syn -j ACCEPT
```

9. Необходимо пропускать некоторые входящие и исходящие пакеты UDP. UDP применяются для DNS, и, если эти пакеты заблокировать, то пользователи не смогут выполнять разрешение адресов. Так как, в отличие от TCP, UDP-пакеты не имеют состояния, нельзя полагаться на проверки флагов SYN или ACK. Вы хотите разрешить UDP только на порт 53, поэтому вы задаете domain (встроенную переменную для порта 53) как единственно допустимый порт. Это делается с помощью следующих инструкций:

```
iptables -A FORWARD -m multiport -p udp -i eth0 -d
192.168.0.0/24 --dports domain -j ACCEPT
```

```
iptables -A FORWARD -m multiport -p udp -i eth0 -s
192.168.0.0/24 --sports domain -j ACCEPT
```

```
iptables -A FORWARD -m multiport -p udp -i eth1 -d
0.0.0.0--dports domain -j ACCEPT
```

```
iptables -A FORWARD -m multiport -p udp -i eth0 -s
0.0.0.0 --sports domain -j ACCEPT
```

```
root@dhcppc4:~# iptables -A FORWARD -m multiport -p udp -i eth0 -d 192.168.0.0/24 --dport domain -j ACCEPT
root@dhcppc4:~# iptables -A FORWARD -m multiport -p udp -i eth0 -s 192.168.0.0/24 --dport domain -j ACCEPT
root@dhcppc4:~# iptables -A FORWARD -m multiport -p udp -i eth1 -d 0.0.0.0 --dport domain -j ACCEPT
root@dhcppc4:~# iptables -A FORWARD -m multiport -p udp -i eth0 -s 0.0.0.0 --dport domain -j ACCEPT
root@dhcppc4:~# █
```

Первая из двух приведенных выше инструкций разрешает входящие дейтаграммы UDP, а вторая - исходящие.

10. Наконец, вы хотите установить протоколирование, чтобы, просматривая журнал, можно было увидеть, какие пакеты были отброшены. Журнал желательно периодически просматривать, даже если проблем нет, просто чтобы иметь представление о видах отброшенного трафика. Если вы видите повторно отброшенные пакеты из одной и той же сети или одного адреса, то вас, возможно, атаковали. Протоколирование всех видов трафика задается одной инструкцией:

```
iptables -A FORWARD -m tcp -p tcp -j LOG
```

```
iptables -A FORWARD -m udp -p udp -j LOG
```

```
Файл  Правка  Вид  Поиск  Терминал  Справка
root@dhcppc4:~# iptables -A FORWARD -m tcp -p tcp -j LOG
root@dhcppc4:~# iptables -A FORWARD -m udp -p udp -j LOG
root@dhcppc4:~# █
```

После выполнения всех перечисленных действий итеративно в командной строке, или заполнения командного файла и его

выполнения, межсетевой экран, защищающий от наиболее распространенных атак, будет настроен. Подробное описание команд можно получить:

`iptables -h`

ВАРИАНТЫ ЗАДАНИЙ

Группа разбивается на подгруппы, каждая из которых берет один из представленных вариантов заданий. В отчет о выполненной работе должен входить скриншот со списком правил, получаемым с помощью `iptables -S`.

Вариант 1	Входящие	Исходящие	Транзитные
Разрешить	<ul style="list-style-type: none"> • http(80) • domain • ftp,ftp-data • icmp 	<ul style="list-style-type: none"> • http(80) • domain • ftp,ftp-data • icmp 	
Запретить	<ul style="list-style-type: none"> • icmp с 192.168.1.2 • tcp с портов 20,21 192.168.1.2 	<ul style="list-style-type: none"> • tcp 365 • udp 334 	
По умолчанию	запретить	разрешить	запретить
Вариант 2	Входящие	Исходящие	Транзитные
Разрешить	<ul style="list-style-type: none"> • http(80) • domain • smpt 	<ul style="list-style-type: none"> • http(80) • domain • smpt 	

	<ul style="list-style-type: none"> • ftp,ftp-data • icmp 	<ul style="list-style-type: none"> • icmp 	
Запретить	<ul style="list-style-type: none"> • icmp с 192.168.1.7 • tcp с портов 17:30 192.168.1.2 	<ul style="list-style-type: none"> • tcp 3682 • udp 16825 	
По умолчанию	запретить	разрешить	запретить
Вариант 3	Входящие	Исходящие	Транзитные
Разрешить	<ul style="list-style-type: none"> • http(80) • domain • pop3 • smtp • icmp • ftp,ftp-data • порты 2048:65535 	<ul style="list-style-type: none"> • http(80) • domain • smtp • pop3 • icmp 	
Запретить	<ul style="list-style-type: none"> • icmp с 192.168.1.7 • с портов 17:30 192.168.1.2 	<ul style="list-style-type: none"> • tcp 3682 • udp 16825 	
По умолчанию	запретить	разрешить	запретить
Вариант 4	Входящие	Исходящие	Транзитные

Разрешить	<ul style="list-style-type: none"> • ftp,ftp-data • domain • pop3 • smtp • icmp • порты 2048:65535 	<ul style="list-style-type: none"> • ftp,ftp-data • smpt • pop3 • icmp • порты 2048:65535 	
Запретить	<ul style="list-style-type: none"> • icmp с 192.168.1.23 • tcp на наш хост к портам 3312,2234 	<ul style="list-style-type: none"> • tcp 368:774 • udp 334:1658 	
По умолчанию	запретить	запретить	разрешить
Вариант 5	Входящие	Исходящие	Транзитные
Разрешить	<ul style="list-style-type: none"> • telnet • domain • ntp • smtp • icmp • ftp,ftp-data • 1024:65535 • 	<ul style="list-style-type: none"> • telnet • domain • ntp • smtp • icmp • 1024:65535 	

Запретить	<ul style="list-style-type: none"> • icmp с 192.168.1.15/30 • udp на наш хост к портам 1456,3333 	<ul style="list-style-type: none"> • tcp 368:774 • udp 334:1658 	
По умолчанию	запретить	запретить	запретить
Вариант 6	Входящие	Исходящие	Транзитные
Разрешить	<ul style="list-style-type: none"> • ssh • domain • ftp,ftp-data • smtp • icmp • 1024:65535 	<ul style="list-style-type: none"> • Ssh • domain • ftp,ftp-data • smtp • icmp • 1024:65535 	
Запретить	<ul style="list-style-type: none"> • icmp с 192.168.1.15/30 • udp на наш хост к портам 1456,3333 	<ul style="list-style-type: none"> • tcp 368:774 • udp 334:1658 	
По умолчанию	запретить	запретить	разрешить

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Сколько существует уровней эталонной модели взаимосвязи открытых систем?
2. Перечислите уровни эталонной модели ВОС, кратко охарактеризуйте их?
3. Объясните принципы функционирования протокола ТСР/РР.
4. Объясните принципы функционирования межсетевых экранов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дэвид В. Чепмен, мл., Энди Фокс. Брандмауэры Cisco Secure PIX = Cisco® Secure PIX® Firewalls. — М.: «Вильямс», 2003. — С. 384. — ISBN 1-58705-035-8.
2. Оглтри Т. "Firewalls. Практическое применение межсетевых экранов", Пресс, 2001 год.
3. Мэйволд, Э. Безопасность сетей. - М: Эком, 2005. – 528 с.
4. Максимов, В. Межсетевые экраны. Способы организации защиты// КомпьютерПресс. – 2003. - № 3

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования «Юго-Западный
государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ

Проректор по учебной работе

О.Г. Локтионова

« 23 » 03

2023 г.



НАСТРОЙКА МЕЖСЕТЕВОГО ЭКРАНА В ОПЕРАЦИОННОЙ СИСТЕМЕ WINDOWS

Методические указания по выполнению лабораторных работ для
студентов укрупненной группы специальностей и направлений
подготовки 10.00.00

Курск 2023

УДК 004

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры
«Информационная безопасность» А.Л. Марухленко

Настройка межсетевого экрана в операционной системе Windows: методические указания по выполнению лабораторной работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 23с. Библиогр.: с. 23.

Излагаются методические указания об администрировании и управлении программно-аппаратными средствами контроля и фильтрации сетевых пакетов способами, а так же защиты от несанкционированного доступа к ресурсам персонального компьютера на операционной системе Windows. Указывается порядок выполнения лабораторной работы, правила оформления и содержание отчета.

Методические указания по выполнению лабораторных работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 149. Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

1	ЦЕЛЬ РАБОТЫ	4
2	ЗАДАНИЕ	4
3	ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	4
4	СОДЕРЖАНИЕ ОТЧЕТА	4
5	ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	5
5.1	Введение	5
5.2	Классификация межсетевых экранов	7
5.2.1	Фильтрующие маршрутизаторы.....	7
5.2.2	Шлюзы сеансового уровня.....	9
5.2.3	Шлюзы уровня приложений	12
6	ВЫПОЛНЕНИЕ РАБОТЫ	14
6.1	Активация встроенного меж сетевого экрана	14
6.2	Настройка параметров брандмауэра.....	15
6.3	Задание для самостоятельной работы	21
7	КОНТРОЛЬНЫЕ ВОПРОСЫ	22
8	БИБЛИОГРАФИЧЕСКИЙ СПИСОК	23

1 ЦЕЛЬ РАБОТЫ

Цель лабораторной работы – ознакомиться с возможностями межсетевого экрана операционной системы Windows XP, изучить последовательность операций по включению и настройке межсетевого экрана и приобрести практические навыки по защите компьютера с помощью механизма межсетевого экранирования.

2 ЗАДАНИЕ

Ознакомиться с теоретическим материалом, активировать встроенный брандмауэр операционной системы Windows XP и настроить его параметры.

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Получить задание;
2. Изучить теоретическую часть;
3. Активировать встроенный межсетевой экран;
4. Настроить параметры брандмауэра;
5. Составить отчет;

4 СОДЕРЖАНИЕ ОТЧЕТА

1. Титульный лист;
2. Краткая теория;
3. Выполненное задание со скриншотами;
4. Ответы на контрольные вопросы;
5. Вывод.

5 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

5.1 Введение

Межсетевое экранирование повышает безопасность объектов внутренней сети за счет игнорирования неавторизованных запросов из внешней среды, тем самым, обеспечивая все составляющие информационной безопасности. Кроме функций разграничения доступа, экранирование обеспечивает регистрацию информационных обменов.

Функции экранирования выполняет межсетевой экран или брандмауэр (firewall), под которым понимают программную или программно-аппаратную систему, которая выполняет контроль информационных потоков, поступающих в информационную систему и/или выходящих из нее и обеспечивает защиту информационной системы посредством фильтрации информации. Фильтрация информации состоит в анализе информации по совокупности критериев и принятии решения о ее приеме и/или передаче.

Брандмауэр в Windows XP — это система защиты подключения к Интернету (Internet Connection Firewall, ICF), представляет собой программу настройки ограничений, регулирующих обмен данными между Интернетом и небольшой сетью или локальным компьютером. Брандмауэр ICF необходимо установить для любого компьютера, имеющего прямое подключение к Интернету.

При включении брандмауэра для локального компьютера, подключенного к Интернету с помощью модема удаленного доступа, брандмауэр ICF обеспечивает защиту этого подключения.

Для брандмауэра подключения к Интернету предусмотрен журнал безопасности для записи событий, связанных с его работой. Журнал безопасности ICF поддерживает следующие возможности.

Записывать пропущенные пакеты. Этот параметр задает запись в журнал сведений о всех потерянных пакетах, исходящих из сети (компьютера) или из Интернета. Если установить флажок «Записывать потерянные пакеты» будут собираться сведения о каждом пакете, который пытался пройти через ICF, но был обнаружен и отвергнут брандмауэром.

Записывать успешные подключения. Этот параметр задает запись в журнал сведений о всех успешных подключениях, инициированных из сети (компьютера) или из Интернета.

Журнал безопасности брандмауэра состоит из двух разделов. В заголовке содержатся сведения о версии журнала и полях, в которые можно записывать данные. Содержимое заголовка имеет вид статического списка.

Содержимое журнала безопасности представляет собой откомпилированные данные, которые вводятся при обнаружении трафика, пытающегося пройти через брандмауэр. Поля журнала заполняются слева направо, как они расположены на странице.

Для того, чтобы в журнал вводились данные, необходимо выбрать хотя бы один параметр ведения журнала или оба параметра.

5.2 Классификация межсетевых экранов

В настоящее время не существует единой и общепризнанной классификации межсетевых экранов. Выделим следующие классы межсетевых экранов:

- Фильтрующие маршрутизаторы.
- Шлюзы сеансового уровня.
- Шлюзы уровня приложений.

Эти категории можно рассматривать как базовые компоненты реальных межсетевых экранов. Лишь немногие межсетевые экраны включают лишь одну из перечисленных категорий. Тем не менее эти компоненты отражают ключевые возможности, отличающие межсетевые экраны друг от друга.

5.2.1 Фильтрующие маршрутизаторы

Фильтрующий маршрутизатор представляет собой маршрутизатор или работающую на сервере программу, сконфигурированные таким образом, чтобы фильтровать входящие и исходящие пакеты. Фильтрация пакетов осуществляется на основе информации, содержащейся в ТСР- и IP-заголовках пакетов.

Фильтрующий маршрутизатор обычно может фильтровать IP-пакеты на основе группы следующих полей заголовка пакета:

- ❖ IP-адрес отправителя;
- ❖ IP-адрес получателя;
- ❖ порт отправителя;
- ❖ порт получателя.

Некоторые маршрутизаторы проверяют, с какого сетевого интерфейса маршрутизатора пришел пакет, и затем используют эту информацию как дополнительный критерий фильтрации.

Фильтрация может быть реализована различными способами для блокирования соединений с определенными компьютерами или портами. Например, можно блокировать соединения, идущие от конкретных адресов тех компьютеров и сетей, которые считаются враждебными или ненадежными.

Правила фильтрации пакетов формулируются сложно, к тому же обычно не существует средств для проверки их корректности, кроме медленного ручного тестирования. При этом в отсутствие фильтрующего маршрутизатора средств протоколирования (если правила фильтрации пакетов все-таки позволят опасным пакетам пройти через маршрутизатор) такие пакеты не смогут быть выявлены до обнаружения последствий проникновения. Даже если администратору сети удастся создать эффективные правила фильтрации, их возможности останутся ограниченными. Например, администратор задает правило, в соответствии с

которым маршрутизатор будет отбраковывать все пакеты с неизвестным адресом отправителя. Однако в данном случае хакер для проникновения внутрь защищенной сети может осуществить атаку, которую называют подменой адреса. В таких условиях фильтрующий маршрутизатор не сумеет отличить поддельный пакет от настоящего и пропустит его.

К положительным качествам фильтрующих маршрутизаторов можно отнести следующие:

- сравнительно невысокая стоимость;
- гибкость в определении правил фильтрации;
- небольшая задержка при прохождении пакетов.

Недостатки фильтрующих маршрутизаторов:

- внутренняя сеть видна (маршрутизируется) из сети Интернет;
- правила фильтрации пакетов трудны в описании и требуют очень хороших знаний технологий TCP и UDP;
- при нарушении работоспособности межсетевого экрана с фильтрацией пакетов все компьютеры за ним становятся полностью незащищенными либо недоступными;
- отсутствует аутентификация на пользовательском уровне.

5.2.2 Шлюзы сеансового уровня

Данный класс маршрутизаторов представляет собой транслятор TCP-соединения. Шлюз принимает запрос авторизованного клиента на конкретные услуги и после проверки

допустимости запрошенного сеанса устанавливает соединение с местом назначения (внешним хостом). После этого шлюз копирует пакеты в обоих направлениях, не осуществляя их фильтрации. Как правило, пункт назначения задается заранее, в то время как источников может быть много. Используя различные порты, можно создавать разнообразные конфигурации соединений. Данный тип шлюза позволяет создать транслятор TCP-соединения для любого определенного пользователем сервиса, базирующегося на TCP, осуществлять контроль доступа к этому сервису и сбор статистики по его использованию.

Шлюз следит за подтверждением (квитированием) связи между авторизованным клиентом и внешним хостом, определяя, является ли запрашиваемый сеанс связи допустимым. Чтобы выявить допустимость запроса на сеанс связи, шлюз выполняет следующую процедуру. Когда авторизованный клиент запрашивает некоторый сервис, шлюз принимает этот запрос, проверяя, удовлетворяет ли данный клиент базовым критериям фильтрации. Затем, действуя от имени клиента, шлюз устанавливает соединение с внешним хостом и следит за выполнением процедуры квитирования связи по протоколу TCP. Эта процедура состоит из обмена TCP-пакетами, которые помечаются флагами SYN (синхронизировать) и ACK (подтвердить).

Первый пакет сеанса TCP, помеченный флагом SYN и содержащий произвольное число, например 500, является

запросом клиента на открытие сеанса. Внешний хост, получивший этот пакет, посылает в ответ другой, помеченный флагом ACK и содержащий число на единицу большее, чем в принятом пакете (в нашем случае 501), подтверждая тем самым прием пакета SYN от клиента.

Далее осуществляется обратная процедура: хост посылает клиенту пакет SYN с исходным числом, например 700, а клиент подтверждает его получение передачей пакета ACK, содержащего число 701. На этом процесс квитирования связи завершается.

Шлюз сеансового уровня признает завершенное соединение допустимым только в том случае, если при выполнении процедуры квитирования связи флаги SYN и ACK, а также числа, содержащиеся в TCP-пакетах, оказываются логически связанными между собой.

После того как шлюз определил, что доверенный клиент и внешний хост являются авторизованными участниками сеанса TCP, и проверил его допустимость, он устанавливает соединение. Начиная с этого момента шлюз копирует и перенаправляет пакеты туда и обратно, не проводя никакой фильтрации. Он поддерживает таблицу установленных соединений, пропуская данные, которые относятся к одному из сеансов связи, зафиксированных в данной таблице. Когда сеанс завершается, шлюз удаляет соответствующий элемент из таблицы и разрывает сеть, использовавшуюся в текущем сеансе.

Недостатком шлюзов сеансового уровня является отсутствие проверки содержимого передаваемых пакетов, что дает возможность нарушителю проникнуть через такой шлюз.

5.2.3 Шлюзы уровня приложений

С целью защиты ряда уязвимых мест, присущих фильтрующим маршрутизаторам, межсетевые экраны должны использовать прикладные программы для фильтрации соединений с такими сервисами, как Telnet и FTP. Подобное приложение называется проху-службой, а хост, на котором работает проху-служба, — шлюзом уровня приложений. Такой шлюз исключает прямое взаимодействие между авторизованным клиентом и внешним хостом. Шлюз фильтрует все входящие и исходящие пакеты на прикладном уровне.

Обнаружив сетевой сеанс, шлюз приложений останавливает его и вызывает уполномоченное приложение для оказания завершаемой услуги. Для достижения более высокого уровня безопасности и гибкости шлюзы уровня приложений и фильтрующие маршрутизаторы могут быть объединены в межсетевом экране.

Шлюзы прикладного уровня позволяют обеспечить надежную защиту, поскольку взаимодействие с внешним миром реализуется через небольшое число уполномоченных приложений, полностью контролирующих весь входящий и исходящий трафик.

Следует отметить, что шлюзы уровня приложений требуют отдельного приложения для каждого сетевого сервиса.

По сравнению с работающими в обычном режиме, при котором прикладной трафик пропускается непосредственно к внутренним хостам, шлюзы прикладного уровня имеют ряд преимуществ:

- ❖ невидимость структуры защищаемой сети из глобальной сети Интернет. Имена внутренних систем можно не сообщать внешним системам через DNS, поскольку шлюз прикладного уровня может быть единственным хостом, имя которого будет известно внешним системам;
- ❖ надежная аутентификация и регистрация. Прикладной трафик может быть аутентифицирован, прежде чем он достигнет внутренних хостов, и зарегистрирован более эффективно, чем с помощью стандартной регистрации;
- ❖ приемлемое соотношение цены и эффективности. Дополнительные программные или аппаратные средства аутентификации или регистрации нужно устанавливать только на шлюзе прикладного уровня;
- ❖ простые правила фильтрации. Правила на фильтрующем маршрутизаторе оказываются менее сложными, чем на маршрутизаторе, который самостоятельно фильтрует прикладной трафик и отправляет его большому числу внутренних систем. Маршрутизатор должен пропускать

прикладной трафик, предназначенный только для шлюза прикладного уровня, и блокировать весь остальной;

❖ возможность организации большого числа проверок.

6 ВЫПОЛНЕНИЕ РАБОТЫ

6.1 Активация встроенного межсетевого экрана

Для активизации межсетевого экрана на компьютере выполните следующие действия:

1. Откройте компонент «Сетевые подключения».
2. Для этого выберите последовательно Пуск-Панель управления-Сетевые подключения.
3. Выделите подключение удаленного доступа, подключение по локальной сети или высокоскоростное подключение к Интернету, которое требуется защитить брандмауэром и затем выберите в контекстном меню (при выделенном подключении нажать правую клавишу мыши) выберите команду «Свойства».
4. На вкладке «Дополнительно» в группе Брандмауэр подключению к Интернету (Рис. 1) отметьте пункт «Защитить мое подключение к Интернету».

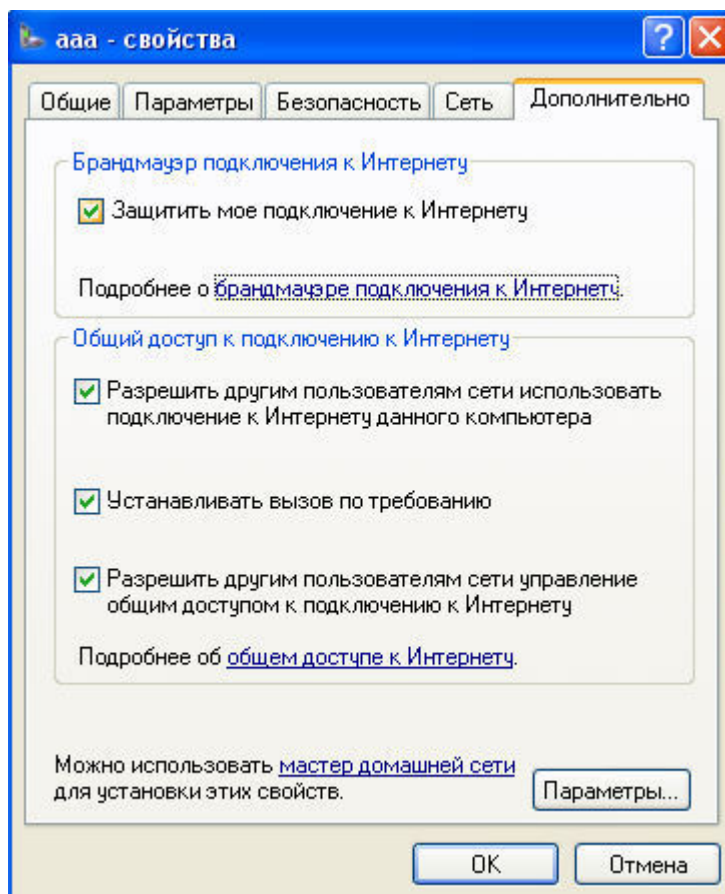


Рис. 1 – Включение встроенного межсетевого экрана

6.2 Настройка параметров брандмауэра

Для настройки параметров брандмауэра на компьютере выполните следующие действия.

1. Выполните пункты 1-3 предыдущего задания.
2. Выберите кнопку «Параметры» в нижней части открытого окна (Рис. 1).
3. В результате откроется окно «Дополнительные параметры» (Рис. 2) с тремя закладками («Службы», «Ведение журнала безопасности» и «ICMP»).
4. Выберите закладку «Службы».
5. Отметьте все службы.

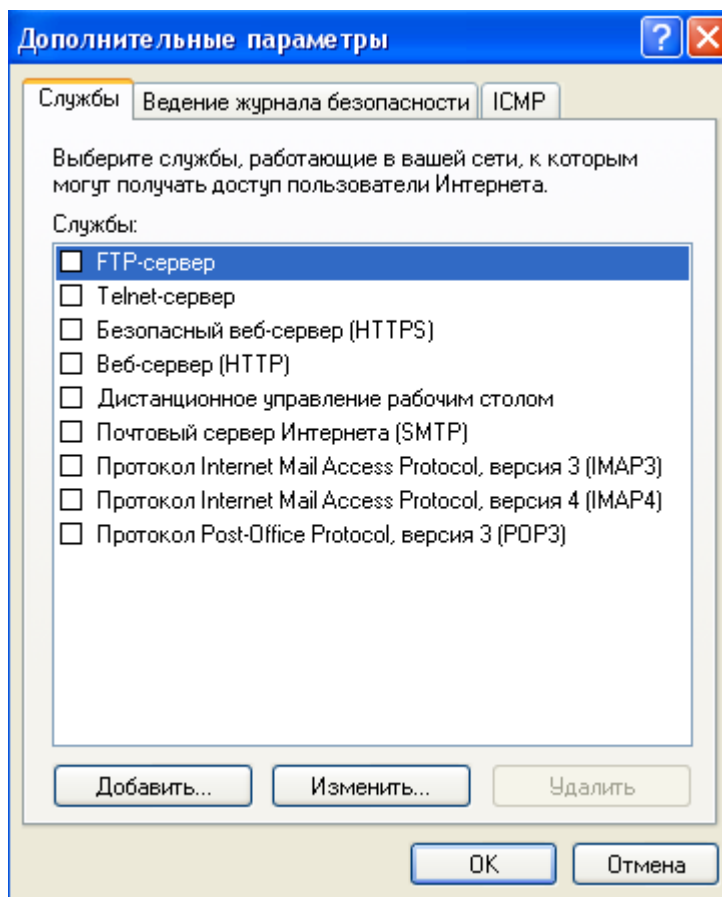


Рис. 2 – Окно дополнительных параметров

6. Выберите закладку «Ведение журнала безопасности» (Рис. 3).

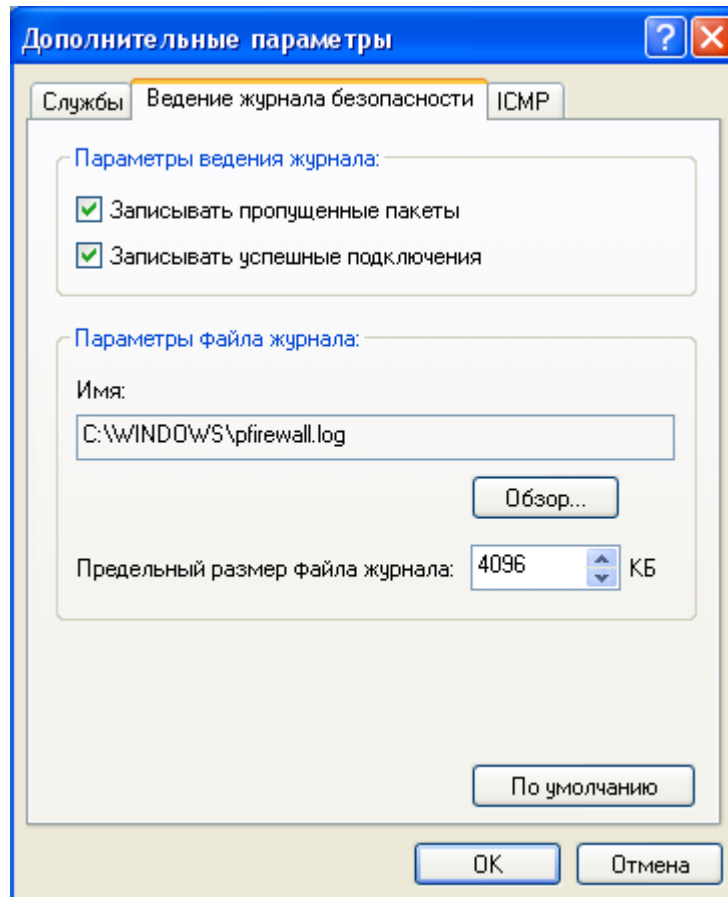


Рис. 3 – Окно ведения журнала безопасности

7. Отметь пункты «Записывать пропущенные пакеты» и «Записывать успешные подключения». Обратите внимание на расположение журнала безопасности.
8. Подключимся к Интернету и посетим любой сайт.
9. Посмотрим журнал безопасности (Рис. 4).

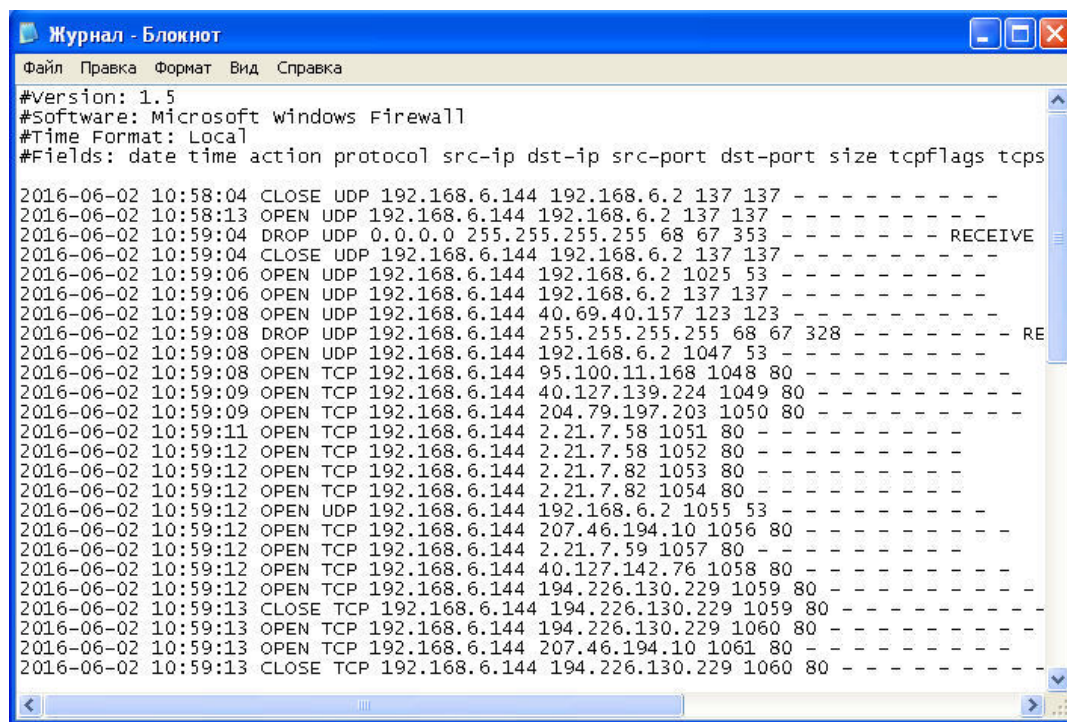


Рис. 4 – Журнал безопасности

Таблица 1 – Структура тела журнала безопасности брандмауэра Windows

Поле	Описание
Дата	Год, месяц и день, когда произошла записанная транзакция. Дата представляется в следующем формате: ГГ-ММ-ДД, где ГГГГ – год, ММ – месяц, а ДД – число.
Время	Время, когда произошла записанная транзакция, записываемое в формате: ЧЧ:ММ:СС, где ЧЧ- часы в 24-часовом формате, ММ - минуты, а СС – секунды
Действие	Операция, обнаруженная и зарегистрированная ОО. Могут записываться следующие действия: OPEN (открытие), CLOSE (закрытие), DROP (отклонение) и INFO-EVENTS-LOST (потерянные события). Для

Поле	Описание
	действия INFO-EVENTS-LOST указывается число событий, которые произошли, но не были записаны в журнал.
Протокол	Протокол, использовавшийся для передачи данных. Если протокол отличен от TCP, UDP и ICMP, в этом поле указывается число пакетов.
src-ip	IP-адрес источника (IP-адрес компьютера, пытавшегося установить подключение).
dst-ip	IP-адрес назначения (IP-адрес компьютера, с которым исходный компьютер пытался установить связь).
src-port	Номер порта источника – компьютера-отправителя. Правильное значение для параметра src-port определяется только для протоколов TCP и UDP. Номер порта задается целым числом в диапазоне от 1 до 65 535. Для всех остальных протоколов запись src-port отображается в виде «-» (дефис).
dst-port	Номер порта конечного компьютера. Правильное значение для параметра dst-port определяется только для протоколов TCP и UDP. Номер порта задается целым числом в диапазоне от 1 до 65 535. Для всех остальных протоколов запись dst-port отображается в виде «-» (дефис).
size	Размер пакета в байтах.
tcpflags	Флаги управления TCP, содержащиеся в заголовке

Поле	Описание
	<p>ТСР пакета IP:</p> <ul style="list-style-type: none"> – Ack Acknowledgment field significant (включение поля подтверждения); – Fin No more data from sender (конец массива данных отправителя); – Psh Push Function (функция принудительной доставки); – Rst Reset the connection (сброс подключения); – Syn Synchronize sequence numbers (синхронизация порядковых номеров); – Urg Urgent Pointer field significant (включение поля указателя срочных данных). <p>Флаги записываются прописными буквами.</p>
tcpsyn	Последовательность портов ТСР в пакете.
tcpack	Номер подтверждения ТСР в пакете.
tcpwin	Размер окна ТСР в байтах в пакете.
icmptype	Число, которое представляет поле Type (Тип) сообщения ICMP.
icmpcode	Число, которое представляет поле Code (Код) сообщения ICMP
info	Сведения, зависящие от типа случившегося действия

6.3 Задание для самостоятельной работы

1. Настроить брандмауэр на работу с Веб-сервером (HTTP), FTP-сервером.
2. Включить журнал безопасности.
3. После выполнения задания 1 и 2 подключиться к Интернету и посетить любой веб-сервер.
4. Завершить работу в Интернете и просмотреть журнал безопасности.

7 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое брандмауэр?
2. Какие бывают брандмауэры?
3. Что фиксирует журнал безопасности брандмауэра?

8 БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Влад Максимов. Межсетевые экраны. Способы организации защиты. [Электронный ресурс] : статья / КомпьютерПресс 3'2003 - Электрон. дан. - Режим доступа: <http://www.compress.ru/article.aspx?id=10145&iid=420#11> , свободный. - Загл. С экрана.

2 Э. Мэйволд. Безопасность сетей. [Электронный ресурс] : курс лекций / Э Мэйволд, 2006 г. - Электрон. дан. - Режим доступа: http://www.intuit.ru/department/security/netsec/10/netsec_10.html , свободный. - Загл. С экрана.

3 Лапони́на, О.Р. Межсетевое экранирование. [Электронный ресурс] : курс лекций / О.Р. Лапони́на, 2006 г. - Электрон. Дан. - Режим доступа: <http://www.intuit.ru/department/network/firewalls/> , свободный. - Загл. с экрана.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования «Юго-Западный
государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ

Проректор по учебной работе

О.Г. Локтионова

« 23 » 03

2023 г.



РАЗРАБОТКА МНОГОНИТЕВЫХ ПРОГРАММ

Методические указания по выполнению лабораторных работ для
студентов укрупненной группы специальностей и направлений
подготовки 10.00.00

Курск 2023

УДК 681.3

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры «Информационная безопасность» А.Л. Марухленко

Разработка многонитевых программ: методические указания по выполнению лабораторной работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 33 с. Библиогр.: с. 33.

Излагаются методические указания по выполнению лабораторной работы на персональной ЭВМ. Изучаются методы создания многонитевых приложений и взаимной синхронизации отдельных потоков процесса в среде Delphi.

Методические указания по выполнению лабораторных работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____, Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 145. Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

1. Цель работы	4
2. Краткая теория	4
2.1. Общие сведения о нитях	4
2.2. Проблема работы с нитями.....	8
2.3. Проблема синхронизации нитей	9
3. Создание многонитевых программ в среде Delphi	10
3.1. Класс TThread для создания потоков	10
3.2. Шаблон для создания нитей:	13
3.3. Класс <i>TCriticalSection</i> для синхронизации потоков с помощью критической секции	14
3.4. Пример многонитевой программы	17
4. Выполнение лабораторной работы.....	27
4.1. Задание на лабораторную работу	27
4.2. Индивидуальные варианты заданий.....	27
4.3. Содержание отчета	32
5. Контрольные вопросы.....	32
6. Библиографический список.....	33

1. ЦЕЛЬ РАБОТЫ

Изучить и научиться применять на практике организации вычислений с помощью нитей (поток) и овладеть приёмами передачи сообщений между ними.

2. КРАТКАЯ ТЕОРИЯ

2.1. Общие сведения о нитях

Процесс - экземпляр выполняемого приложения. При запуске приложения происходит выделение памяти под процесс, в часть которой и загружается код программы.

Поток - объект внутри процесса, отвечающий за выполнение кода и получающий для этого процессорное время.

В традиционных процессах есть только один поток управления и один счетчик команд. Но в некоторых современных ОС существует возможность создать несколько потоков управления в одном процессе. Такие потоки обычно называются просто **нитьями** или, иногда, **легковесными процессами**.

Точно также как многозадачная операционная система может делать несколько вещей одновременно при помощи разных процессов, один процесс может делать много вещей при помощи нескольких нитей. Каждая нить представляет собой независимо выполняющийся поток управления со своим счетчиком команд, регистровым контекстом и стеком. **Основные отличия процесса от нити заключаются в том, что, каждому процессу соответствует своя независимая от других область памяти, таблица открытых файлов, текущая директория и прочая информация уровня ядра.** Нити же не связаны непосредственно с этими сущностями. У всех нитей принадлежащих данному процессу всё выше перечисленное общее, поскольку принадлежит этому процессу. Кроме того, процесс всегда является сущностью уровня ядра, то есть ядро знает о его существовании, в то время как нити зачастую являются сущностями уровня пользователя и ядро может ничего не знать о ней. В подобных реализациях все данные о нити хранятся в пользовательской области памяти, и

соответственно такие процедуры как порождение или переключение между нитями не требуют обращения к ядру и занимают на порядок меньше времени.

В традиционных ОС понятие нити тождественно понятию процесса. В действительности желательно иметь несколько нитей управления, разделяющих единое адресное пространство, но выполняющихся квазипараллельно. Предположим, например, что файл-сервер блокируется, ожидания выполнения операции с диском. Если сервер имеет несколько нитей управления, вторая нить может выполняться, пока первая нить находится в состоянии ожидания. Это повышает пропускную способность и производительность. Другой пример приложения, в котором необходимы потоки управления, – это браузеры. Многие веб-страницы оформлены с помощью большого числа маленьких изображений. Чтобы загрузить каждое из них, браузеру необходимо каждый раз создавать новое соединение, и львиная доля времени уходит на процесс установки и завершения этих соединений. Если же один браузер будет использовать несколько потоков, он сможет запрашивать несколько картинок одновременно, что приводит к заметному увеличению скорости.

Ниже показана машина с тремя процессами (Рисунок 1а). Каждый процесс имеет собственный программный счетчик, собственный стек, собственный набор регистров и собственное адресное пространство. Каждый процесс не должен ничего делать с остальными, за исключением того, что они могут взаимодействовать посредством системных примитивов связи, таких как семафоры, мониторы, сообщения. На рисунке «б» показана другая машина с одним процессом. Каждая нить выполняется строго последовательно и имеет свой собственный программный счетчик и стек. Нити разделяют процессор так, как это делают процессы (разделение времени). Только на многопроцессорной системе они действительно выполняются параллельно. Нити могут, например, порождать нити-потомки, могут переходить в состояние ожидания до завершения системного вызова, как обычные процессы, пока одна нить заблокирована, другая нить того же процесса может выполняться.

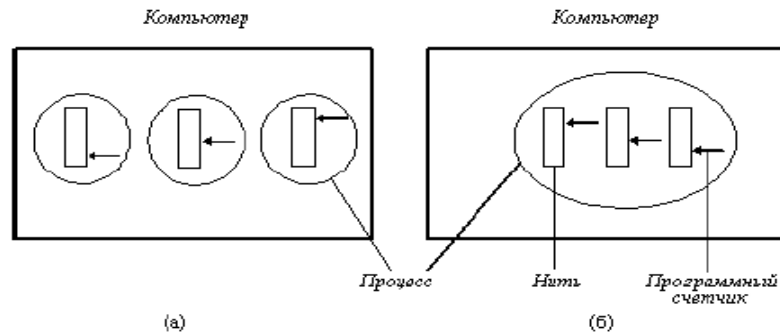


Рисунок 1. а) Три процесса с одной нитью каждый б) Один процесс с тремя нитями

Рассмотрим теперь жизненный цикл нити, а именно последовательность состояний в которых пребывает нить за время своего существования. В целом можно выделить четыре таких состояния:

Таблица 1 – Состояния потока

Состояние нити	Что означает
Готова /Ready/	Нить готова к выполнению, но ожидает процессора. Возможно она только что была создана, была вытеснена с процессора другой нитью, или только что была разблокирована (вышла из соответствующего состояния).
Выполняется /Running/	Нить сейчас выполняется. Следует заметить, что на многопроцессорной машине может быть несколько нитей в таком состоянии.
Заблокирована /Blocked/	Нить не может выполняться, так как ожидает чего-либо. Например, окончания операции ввода-вывода, сигнала от условной переменной, получения mutex и т.п.
Завершена /Terminated/	Нить была завершена, например, вследствие возврата из функции нити, прерывания выполнения нити /cancellation/.

Различные частные реализации могут вводить дополнительные к этим четырем состояния, но все они будут в сущности лишь подсостояниями этих. В целом диаграмму переходов между этими состояниями можно изобразить следующим образом:

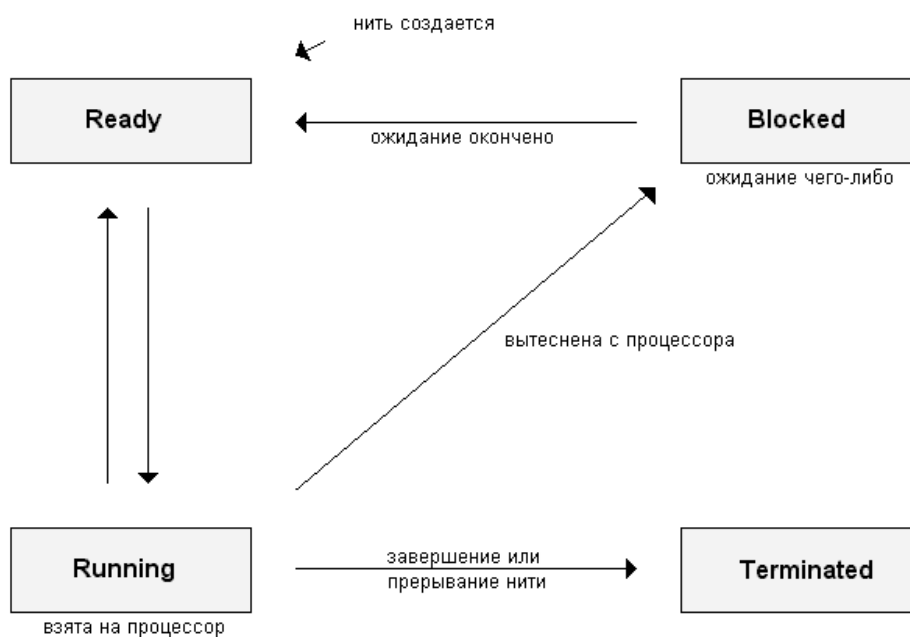


Рисунок 2 – Состояние нитей

Нити могут создаваться системой, например, начальная нить, которая создается при создании процесса, или могут создаваться при помощи явных вызовов пользовательским процессом. Однако любая создаваемая нить начинает свою жизнь в состоянии "готова". После чего в зависимости от политики планирования системы она может либо сразу перейти в состояние "выполняется" либо перейти в него через некоторое время.

Выполняющаяся нить, скорее всего, рано или поздно либо перейдет в состояние "заблокирована", вызвав операцию, ожидающую чего-то, например, окончания ввода-вывода, прихода сигнала или поднятия семафора, либо перейдет в состояние "готова" будучи снята с процессора или более высокоприоритетной нитью или просто потому, что исчерпала свой квант времени. Здесь надо подчеркнуть разницу между *вытеснением /preemption/* то есть снятием с процессора вследствие появления готовой более

приоритетной задачи, и снятием нити вследствие истечения ее кванта времени.

Заблокированная нить, дождавшись события, которого она ожидала, переходит в состояние "готова" при этом, конечно в случае если есть такая возможность, она сразу перейдет в состояние выполнения.

Наконец выполняющаяся нить может завершиться тем или иным способом. Например, вследствие возврата из функции нити, вызова функции завершения нити или вследствие насильственного прерывания ее выполнения. При этом, если нить была отсоединена, то она сразу освобождает все связанные с ней ресурсы и перестает существовать (на самом деле она скорее всего просто будет повторно использована библиотекой поддержки нитей, поскольку создание нити не самая дешевая операция).

2.2. Проблема работы с нитями

Нити существенно усложняют программную модель. Представьте себе системный вызов, создающий процесс – потомок. Если родительский процесс состоял из множества нитей, должно ли это свойство распространяться на дочерний процесс? Если нет, то ожидаемо неправильное функционирование процесса, поскольку все нити могут оказаться необходимы.

Другой класс проблем связан с тем, что нити совместно используют большое количество структур данных. Одна нить может закрыть файл в то время, когда другая считывает из него данные. Если одной нити стало недостаточно памяти, и она просит выделить дополнительную, но происходит переключение потоков, и теперь новая нить также замечает, что памяти недостаточно и запрашивает её для себя. В этой ситуации память может быть выделена дважды.

Еще одна проблема связана с сообщениями об ошибках. В операционных системах, после системного вызова, система помещает информацию о состоянии операции в глобальную переменную, *errno*. Если сразу после того, как первая нить сделала системный вызов, управление было передано другой, которая также сделает системный вызов – значение глобальной переменной будет затёрто.

Последняя проблема, порождаемая нитями, — управление стеками. Во многих системах при переполнении стека процесса ядро автоматически увеличивает его. Если у процесса несколько нитей, стеков тоже должно быть несколько. Если ядро не знает о существовании этих стеков, оно не может их автоматически наращивать при переполнении. Ядро может даже не связать ошибки памяти с переполнением стеков.

При запуске приложения система автоматически создает поток для его выполнения. Поэтому если приложение однопоточное, то весь код будет выполняться последовательно, учитывая все условные и безусловные переходы.

Каждый поток может создать другой поток и т.д. Потоки не могут существовать отдельно от процесса, т.е. каждый поток принадлежит какому-то процессу и этот поток выполняет код, только в адресном пространстве этого процесса.

2.3. Проблема синхронизации нитей

Многопоточность обеспечивает псевдопараллельную работу множества программ. В некоторых случаях без создания потоков нельзя обойтись, например, при работе с сокетами в блокирующем режиме.

Очень часто на практике возникают конфликты между потоками, обращающимися к разделяемым ресурсам, а также к глобальным переменным. Типичными ситуациями, при которых требуется синхронизация потоков, являются *гонки* и *тупики*.

Ситуация *гонок* возникает, когда два и более потока пытаются получить доступ к общему ресурсу и изменить его состояние. Например, пусть *поток А* получил доступ к ресурсу и изменил его в своих интересах; затем активизировался *поток Б* и модифицировал этот же ресурс до завершения *потока А*. *Поток А* полагает, что ресурс остался в том же состоянии, в каком был до переключения. В зависимости от того, когда именно был изменен ресурс, результаты могут варьироваться - иногда код будет выполняться нормально, иногда нет. Но программист не должен строить никаких гипотез относительно порядка исполнения потоков, т.к. планировщик операционной системы может запускать их и останавливать в любое время. Приведем простой пример

ситуации гонок. Пусть два потока выполняют одновременно следующий код:

```
Inc(i);  
if i=Value then DoSomething;
```

Здесь *i* - глобальная переменная, доступная для обоих потоков. Допустим, *поток А* инкрементировал значение переменной *i* и хочет проверить ее значение для выполнения тех или иных условий. Но тут активизировался *поток Б*, который еще увеличивает значение переменной *i*. В результате *поток А* может "проскочить" мимо условия, которое, казалось бы, должно было быть выполнено.

Другими случаями взаимодействия потоков, требующими синхронизации, являются *тупики*. Ситуации тупиков имеют место, когда поток ожидает ресурс, который в данный момент принадлежит другому потоку. Рассмотрим пример. Допустим, *поток 1* захватывает *ресурс А*, и для того, чтобы продолжить работу, ждет возможности захватить *ресурс Б*. В тоже время *поток 2* захватывает *ресурс Б* и ждет возможности захватить *ресурс А*. Развитие этого сценария заблокирует оба потока, и ни один из них не будет выполняться. Ресурсами могут выступать любые совместно используемые объекты системы - файлы, массивы в памяти, устройства ввода/вывода и т.п.

3. СОЗДАНИЕ МНОГОНИТЕВЫХ ПРОГРАММ В СРЕДЕ

DELPHI

3.1. Класс TThread для создания потоков

Для реализации потоков в Delphi присутствует класс TThread. Сам класс не является полностью функциональным, так как метод *Execute*, который должен содержать код потока, в данном классе абстрактный. Некоторые свойства, методы и события класса TThread представлены в таблице.

Таблица 2 – Свойства и методы класса TThread

Свойства класса TThread	
Свойство	Описание
FreeOnTerminate	Освободить ли память, выделенную под экземпляр класса процесса, когда этот процесс завершается. Если <i>True</i> - при завершении Процесса (или при вызове метода <i>Terminate</i>) экземпляр класса автоматически освобождается (аналогично вызову метода <i>Free</i>). <i>Тип: Boolean;</i>
Handle (ThreadID)	Дескриптор процесса. Эта величина может быть использована для управления процессом через функции WinAPI. <i>Тип: THandle;</i>
Priority	Приоритет процесса. Возможные значения этого свойства мы разберем немного позже. <i>Тип: TThreadPriority;</i>
Suspended	Показывает, в каком состоянии находится процесс: <i>True</i> - приостановлен, <i>False</i> - в нормальном. <i>Тип: Boolean;</i>
Terminated	Показывает, завершен ли процесс. <i>True</i> - завершен, <i>False</i> - нет. <i>Тип: Boolean;</i>

Методы класса TThread	
<i>Method</i>	<i>Описание</i>
Create (CreateSuspended: Boolean)	Создает экземпляр класса. Параметр <i>CreateSuspended</i> указывает на то, нужно ли создавать приостановленную задачу (<i>True</i>), или запускать ее сразу (<i>False</i>);
Suspend	Приостанавливает выполнение процесса;
Resume	Продолжает выполнение процесса после <i>Suspend</i> ;
Terminate	Полностью прекращает выполнение процесса; На самом деле просто помечает нить, как завершенной.
WaitFor	Ждет завершения процесса, возвращая затем код его завершения (<i>ReturnValue</i>);
Synchronize (Method : TThreadMethod)	Синхронизирует выполнение метода процесса, позволяя ему работать параллельно с другими процессами.
События класса TThread	
<i>Событие</i>	<i>Описание</i>
OnTerminate	Возникает, когда процесс находится в стадии завершения.

Приоритет - это величина, определяющая, насколько данный процесс должен выполняться быстрее по сравнению с другими. Т.е., другими словами, чем выше приоритет процесса, тем больше времени он отбирает у системы и других, параллельно работающих

процессов. Разберем возможные значения свойства `Priority` класса `TThread` в порядке возрастания приоритета:

- **`tpIdle`** - процесс выполняется только тогда, когда система не занята и больше нет работающих в данный момент процессов;
- **`tpLowest`** - на два пункта ниже нормального;
- **`tpLower`** - на один пункт ниже нормального;
- **`tpNormal`** - нормальный. Такой приоритет у большинства задач;
- **`tpHigher`** - на один пункт выше нормального;
- **`tpHighest`** - на два пункта выше нормального;
- **`tpTimeCritical`** - самый высокий приоритет - занимает все время процессора и системы. Это приоритет для систем реального времени, для которых важна каждая секунда, и даже малейшая задержка может привести к сбою. Будьте осторожны с этим приоритетом.

3.2. Шаблон для создания нитей:

{Определение класса TMyThread}

```
type
  TMyThread = class(TThread)
  private
    { Private declarations }
  protected
    procedure Work;
    procedure Execute; override;
  end;
```

implementation

```
procedure TMyThread.Execute;
begin
  {Если Вы хотите, чтобы процедура Work
  выполнялась лишь один раз - удалите цикл while}
  while not Terminated do
    Synchronize(Work);
end;
```

```

procedure TMyThread.Work;
begin
    {Здесь можно уже выполнять те задачи,
    которые должны быть исполнены процессом}
end;

```

3.3. Класс *TCriticalSection* для синхронизации потоков с помощью критической секции

Наиболее простым в понимании способом синхронизации является *критическая секция*. Код, расположенный в критической секции может выполняться только одним потоком. В принципе код ни как не выделяется, а происходит обращение к коду через критическую секцию. В начале кода находится функция входа в секцию, а по завершению его выход из секции. Если секция занята потоком, то остальные потоки ждут, пока критическая секция не освободится. В библиотеке **VCL** критические секции представлены классом **TCriticalSection**. Рассмотрим методы этого класса.

Таблица 3 – Методы класса *TCriticalSection*

<i>Метод</i>	<i>Описание</i>
procedure Acquire; override;	Привязывает критическую секцию к вызывающему потоку.
constructor Create;	Создает объект критической секции.
destructor Destroy; override;	Уничтожает объект критической секции.
procedure Enter;	Блокирует прочие потоки, когда вызывающий поток заходит в критическую секцию.
procedure Leave;	Позволяет другим потокам использовать критическую секцию.
procedure Release; override;	Освобождает критическую секцию.

Здесь нужно уточнить, что методы **Enter** и **Leave** содержат вызовы методов **Acquire** и **Release** соответственно. Поэтому можно сказать, что эти соответствующие пары методов (**Enter** и **Acquire**, **Leave** и **Release**) выполняют одни и те же действия.

Использовать критические секции довольно просто. Рассмотрим небольшой пример. Допустим, имеется функция, увеличивающая значения элементов глобального массива на единицу.

```
function IncElem;
var i: byte;
begin
    for i:=1 to 10 do
        Inc(mas[i]);
end;
```

Если эту функцию будут использовать несколько потоков, то может возникнуть конфликт по данным. Чтобы этого не произошло, будем использовать критическую секцию. Для начала следует подключить модуль **SyncObjs.pas**, содержащий реализацию класса **TCriticalSection**. Теперь мы можем описать глобальную переменную **Section**.

```
Var Section: TCriticalSection;
```

Затем создадим объект критической секции с помощью метода **Create**.

```
Section:=TCriticalSection.Create;
```

Далее используем критическую секцию следующим образом:

```
procedure IncElem;
var i: byte;
begin
    Section.Enter;
```

```

for i:=1 to 100 do
    Inc(mas[i]);
    Section.Leave;
end;

```

Что будет происходить, когда потоки начнут выполнять данную процедуру? Когда поток встретит в процедуре **Section.Enter**, он проверит, не занята ли секция другим потоком, и, если она свободна, начнет выполнять код, пока не встретит вызов метода **Section.Leave**. После использования критической секции, когда она уже больше не нужна, ее необходимо уничтожить.

Section.Free;

Вообще, критических секций может быть несколько. Поэтому при использовании нескольких функций, в которых могут быть конфликты по данным, надо для каждой функции создавать свою критическую секцию. Вход и выход из критической секции не обязательно должны находиться в одной функции. Но нужно внимательно следить за тем, чтобы поток имел выход из критической секции, иначе остальные потоки так и не получат к ней доступ.

Очевидно, что такой метод надежно обеспечивает задачам монополярный доступ к общим (критическим) ресурсам.

Метод критической секции имеет ряд преимуществ перед его аналогами. Так, например, использование семафоров (Semaphore) сложнее в реализации. Другой метод взаимных исключений — mutex — в целом похож на метод критической секции, но он требует больше системных ресурсов и имеет свое время тайм-аута, по истечении которого ожидающий процесс может все-таки войти в защищенный блок, в то время как в критической секции подобного механизма нет.

3.4. Пример многонитевой программы

Рассмотрим применение компонентов классов **TThread** и **TCriticalSection** на конкретном примере: существуют 5 нитей A, B, C, D, E и один ресурс *RES* типа *integer*, с которым они работают – увеличивают значение *res* на единицу. Вначале запускаются 3 нити A, B, C. Нити A и B периодически обращаются к *RES*, а нить C через определенный момент времени занимает ресурс и работает с ним какое-то время. Когда все три нити завершат свою работу, запускаются нити D и E. Нить D периодически обращается к *RES*, а E останавливает нить D, работает с ресурсом и запускает её снова. Программа завершает работу после завершения работы всех нитей.

Для того, чтобы отследить работу нитей создадим 5 меток и 5 полей. В метках будем выводить состояния нитей, а в полях текущее значение *RES*. Добавим в форму из панели Standard 10 label, 5 Memo и 1 Button.

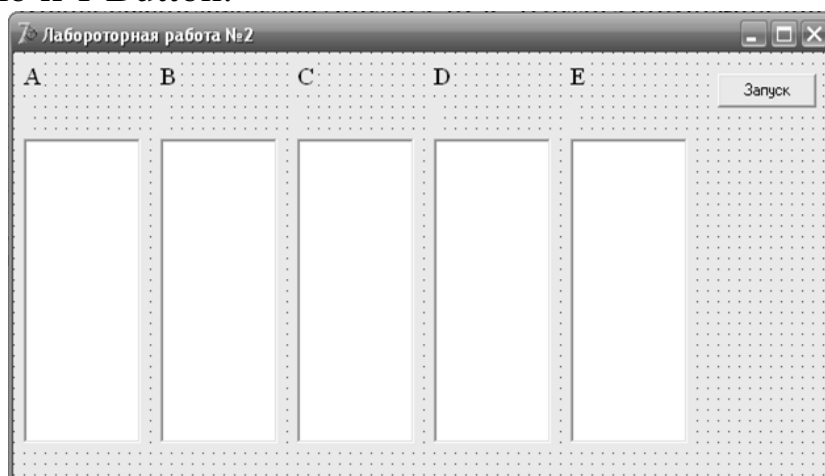


Рисунок 3 – Форма для отображения состояний пяти нитей

Под каждой буквой находится метка для вывода состояния нити. Подключаем модуль **SyncObjs.pas**, содержащий реализацию класса **TCriticalSection**. Добавляем к компонентам в **user StdCtrls**. Затем записываем объявление класса для каждой нити в разделе **type**:

```
TMyThread1 = class(TThread) //объявление
  класса для нити A
  private
```

```

    { Private declarations }
protected
    procedure Work;           //процедура, к
котой будет обращаться нить,
                               //и      которая
будет увеличивает переменную RES
    procedure Execute; override; //основная
процедура, которая запускается
                               //при создании
нити
end;

TMyThread2 = class(TThread) //объявление
класса для нити B
private
    { Private declarations }
protected
    procedure Work;
    procedure Execute; override;
end;

TMyThread3 = class(TThread) //объявление
класса для нити C
private
    { Private declarations }
protected
    procedure Execute; override;
end;

TMyThread4 = class(TThread) //объявление
класса для нити D
private
    { Private declarations }
protected
    procedure Work;
    procedure Execute; override;
end;

```

```

    TMyThread5 = class(TThread) //объявление
класса для нити E
    private
    { Private declarations }
    protected
    procedure Execute; override;
end;

```

У нитей С(3ей) и Е(5ой) нет процедуры **Work**, потому что они им не нужны. Нить С – блокирует ресурс и работает с ним определенное время, не периодически. Аналогично и нить Е, только она останавливает D.

Создаем глобальные переменные на основе классов **TThread** и **TCriticalSection** и общий ресурс в разделе **var**:

```

    A:TMyThread1; //создание переменной A на
основе класса TThread1
    B:TMyThread2; //создание переменной B на
основе класса TThread2
    C:TMyThread3; //создание переменной C на
основе класса TThread3
    D:TMyThread4; //создание переменной D на
основе класса TThread4
    E:TMyThread5; //создание переменной E на
основе класса TThread5
    SecRes:TCriticalSection; //создание переменной
SecRes на основе класса TCriticalSection
(критической секции)
    start:TCriticalSection; //создание критической
переменной для синхронизации запуска нитей D и E
    res:integer; //создание общего ресурса

```

Затем необходимо описать основную процедуру **Execute** и **Work**(если есть) для каждой нити.

Команда **sleep(n)**, где **n** – количество тактов (1000 тактов – 1 сек.), заставляет нить «покурить» в течении **n** тактов;

предназначена для того, чтобы мы смогли лучше отследить работу нити.

В конце каждой процедуры **Execute** необходимо указать **'Переменная нити'.Terminate** для завершения работы нити. Но перед тем как завершать нити А, В и С нам надо произвести запуск D и E (точнее возобновить работу этих нитей, так как они находятся в приостановленном режиме), а так как эти нити могут работать только после окончания работы тех, то необходимо воспользоваться оператором **if** и свойством **Terminated**, которое возвращает **true** если процесс завершен и **false** если нет. Для того чтобы не возникло ошибок при проверке этих условий мы будем пользоваться переменной **start** (критической секцией). Нам понадобятся два компонента этого класса: **Enter** и **Leave**. Когда нить встретит **start.Enter**, то она проверит занята ли критическая секция другой нитью, если занята, то будет дожидаться её освобождения, если свободна, то пометит секцию как занятой и продолжит выполнять код программы. На выходе надо обязательно указать **start.Leave**, чтобы нить по завершению работы помечала критическую секцию свободной.

```
start.Enter; //проверка секции - занята или нет
if (B.Terminated=true) and (C.Terminated=true)
then D.Resume;
if (B.Terminated=true) and (C.Terminated=true)
then E.Resume;
A.Terminate; //помечает нить как завершенной
(хотя в источниках указано, что завершает
процесс нити)
start.Leave; //помечает секцию свободной
```

Нить E должна остановить работу D, а потом возобновить для этого используются методы:

D.Suspend; //приостанов нити D

D.Resume; //возобновление работы нити D

Так как в поставленной задаче нити должны обратиться к **res** определенное количество раз, то лучше использовать цикл **For**.

Если необходимо, чтобы нить работа до завершения, то нужно указать цикл **While**:

```
while not Terminated do begin
  команды
end;
```

Для того, чтобы нити работали нормально с переменной **res** мы будем пользоваться переменной **SecRes** (критической секцией).

Записываем данные процедуры в разделе **implementation**:

```
procedure TMyThread1.Execute; //нить А
var
i:integer;
begin
Form1.label6.Caption:='Запущена'; //выводит
сообщение что нить А запущена
For i:=1 to 10 do begin //обратиться к процедуре
10 раз
Work;
Form1.Memo1.Lines.Add(inttostr(res)); //выводит
значение res
sleep(500); end;
//если нити В и С завершили свою работу, то
ВОЗОБНОВИТЬ работу D и E,
//т. к. нити D и E будут уже запущенными, но
приостановленными
start.Enter; // вход в секцию, связанную с
синхронизацией
if (B.Terminated=true) and (C.Terminated=true)
then D.Resume;
if (B.Terminated=true) and (C.Terminated=true)
then E.Resume;
Form1.label6.Caption:='Завершена';
A.Terminate;
start.Leave; //помечает секцию свободной
```

```
end;
```

```
procedure TMyThread1.Work;  
begin  
  SecRes.Enter; //проверка секции - занята или нет  
  Inc(res);  
  SecRes.Leave; //помечает секцию свободной  
end;
```

```
procedure TMyThread2.Execute; //нить В  
var  
  i:integer;  
begin  
  Form1.label7.Caption:='Запущена';  
  For i:=1 to 15 do begin //обратиться к процедуре  
    15 раз  
    Work;  
    Form1.Memo2.Lines.Add(inttostr(res));  
    sleep(500); end;  
  start.Enter; //вход в секцию, связанную с  
  синхронизацией  
  if (A.Terminated=true) and (C.Terminated=true)  
  then D.Resume;  
  if (A.Terminated=true) and (C.Terminated=true)  
  then E.Resume;  
  Form1.label7.Caption:='Завершена';  
  B.Terminate;  
  start.Leave; //помечает секцию свободной  
end;
```

```
procedure TMyThread2.Work;  
begin  
  SecRes.Enter; //проверка секции - занята или нет  
  Inc(res);  
  SecRes.Leave; //помечает секцию свободной  
end;
```

```

procedure TMyThread3.Execute; //нить С
var
i:integer;
begin
Form1.label8.Caption:='Запущена';
sleep(3000);
SecRes.Enter; //проверка секции - занята или нет
For i:=1 to 10 do begin //обратиться к процедуре
10 раз
Form1.Memo3.Lines.Add(inttostr(res));
sleep(500); end;
SecRes.Leave; //помечает секцию свободной
start.Enter; // вход в секцию, связанную с
синхронизацией
if (B.Terminated=true) and (A.Terminated=true)
then D.Resume;
if (B.Terminated=true) and (A.Terminated=true)
then E.Resume;
Form1.label8.Caption:='Завершена';
C.Terminate;
start.Leave; //помечает секцию свободной
end;

```

```

procedure TMyThread4.Execute; //нить D
var
i:integer;
begin
Form1.label9.Caption:='Запущена';
For i:=1 to 15 do begin //обратиться к процедуре
15 раз
Work;
Form1.Memo4.Lines.Add(inttostr(res));
sleep(500); end;
Form1.label9.Caption:='Завершена';
SecRes.Free; //уничтожает критическую секцию
D.Terminate;
end;

```

```

procedure TMyThread4.Work;
begin
  SecRes.Enter; //проверка секции - занята или нет
  Inc(res);
  SecRes.Leave; //помечает секцию свободной
end;

procedure TMyThread5.Execute; //нить E
var
  i:integer;
begin
  Form1.label10.Caption:='Запущена';
  sleep(3000);
  D.Suspend; //приостанов нити D
  form1.label9.Caption:='Приостановлена';
  For i:=1 to 10 do begin //обратиться к процедуре
  10 раз
  Form1.Memo5.Lines.Add(inttostr(res));
  sleep(500); end;
  D.Resume; //возобновление работы нити D
  form1.label9.Caption:='Запущена';
  Form1.label10.Caption:='Завершена';
  E.Terminate;
end;

```

После завершения работы с критической секцией необходимо её удалить – это можно сделать с помощью команды **Имя.Free**. В нашем случае:

```

SecRes.Free;
start.Free;

```

Она находится в процедуре Execute нити D, так как по расчетам она должна завершиться последней.

Теперь необходимо создать и запустить нити. Для этого используем метод **Переменная :=ИмяКласса.Create(False**

or True), *false* – нить сразу запускается, *true* – нить приостановлена. Воспользуемся кнопкой для запуска нитей – щелкните два раза по ней и добавьте код:

```

SecRes:=TCriticalSection.Create;           //создает
объект критической секции
start:=TCriticalSection.Create; //создает объект
критической секции
A:=TMyThread1.Create(False);
B:=TMyThread2.Create(False);
C:=TMyThread3.Create(False);
D:=TMyThread4.Create(True);               //нить
приостановлена
E:=TMyThread5.Create(True);

//проверяет приостановлена ли нить D
if          D.Suspended          then
label9.Caption:='Приостановлена';

if E.Suspended then
label10.Caption:='Приостановлена';
end;
```

Компилируем и запускаем программу.

На рисунке 4 проиллюстрирован момент времени, когда нить С работает с ресурсом, А и В ждут пока ресурс освободиться, а D и E приостановлены.

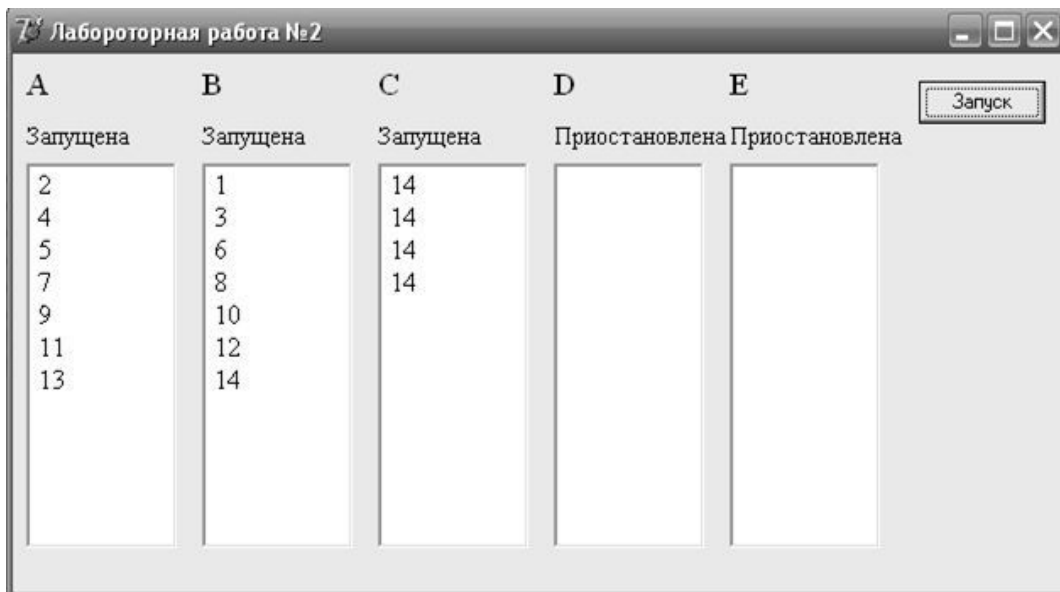


Рисунок 4 – Состояние нитей в работающей программе

Момент, когда нити А, В и С завершили работу, запустив D и E. Нить E уже завершила свою работу, а D заканчивает показан на рисунка 5.



Рисунок 5 – Состояние нитей в завершающейся программе

4. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

4.1. Задание на лабораторную работу

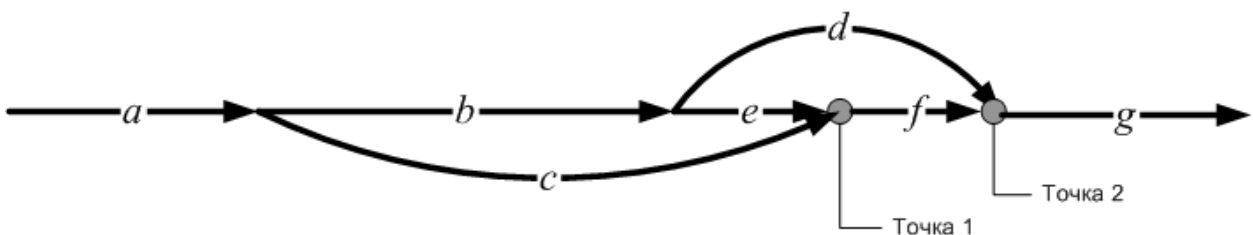
Написать многопоточную программу с определённым порядком работы нитей, заданным индивидуальным вариантом задания. Состояние каждой нити должно отображаться на главном окне приложения. В каждом варианте существует несколько точек, в которых должна произойти синхронизация работы нитей.

Для запуска нитей и их синхронизации используйте предусмотренные Delphi стандартные классы.

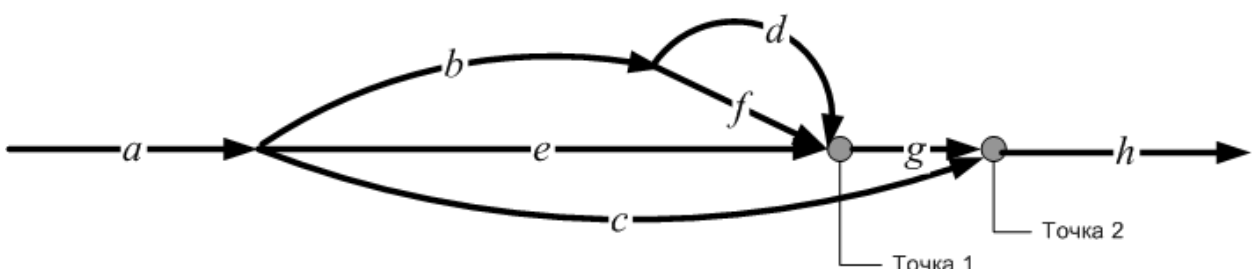
4.2. Индивидуальные варианты заданий

Пояснения к варианту. Для *первого* варианта программа должна состоять из 7 нитей. После начала работы программы запускается нить “a”, работающая некоторое время и по завершении своей работы запускающая нити “b” и “c”. Нить “b” по завершении работы запускает нити “d” и “e”. По завершении работы нитей “c” и “e” запускается нить “f”, по завершении работы нитей “d” и “f” запускается нить “g”, завершение работы которой означает завершение выполнения программы. Остальное на усмотрение студента.

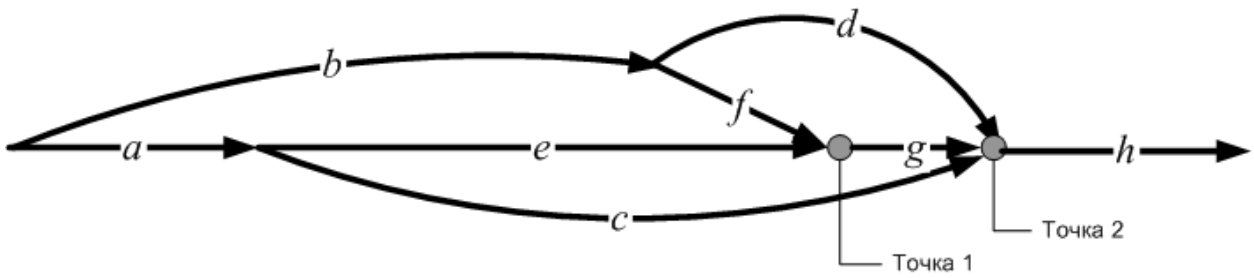
Вариант 1:



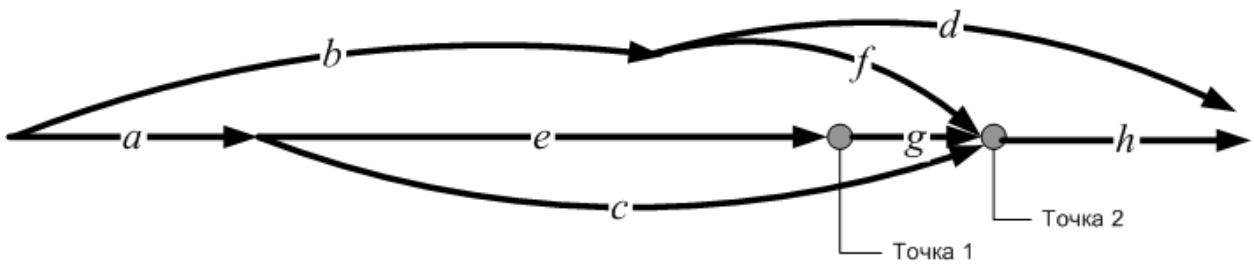
Вариант 2:



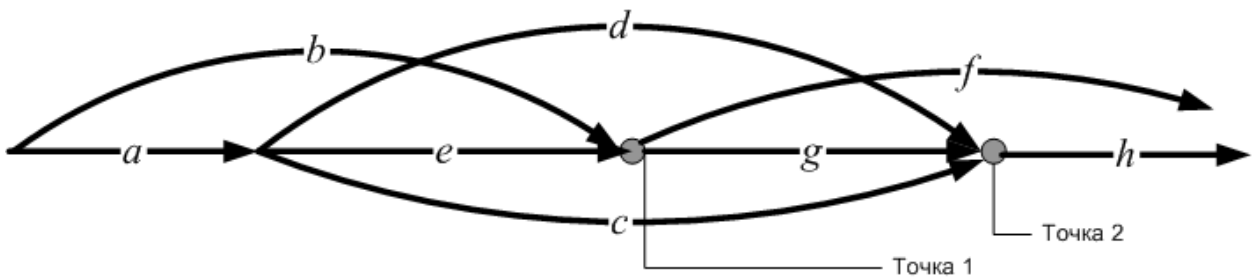
Вариант 3:



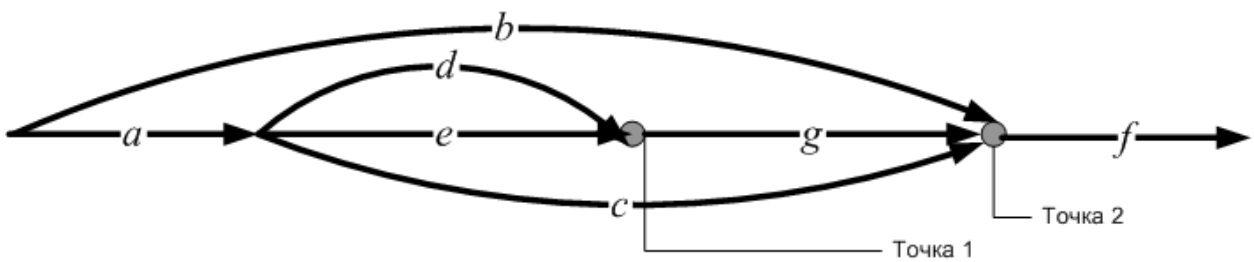
Вариант 4:



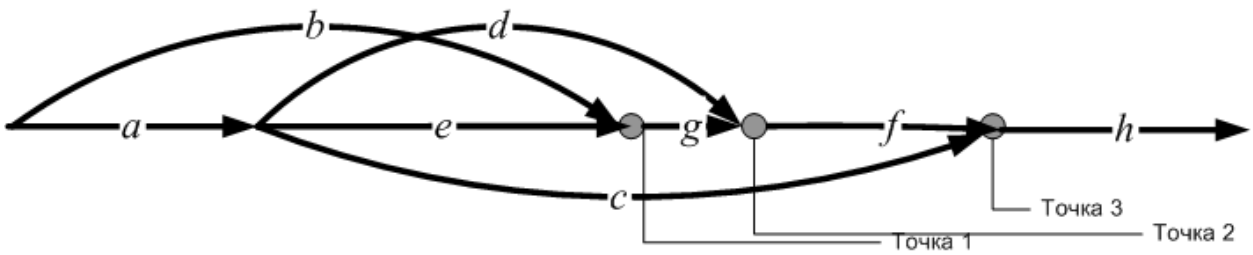
Вариант 5:



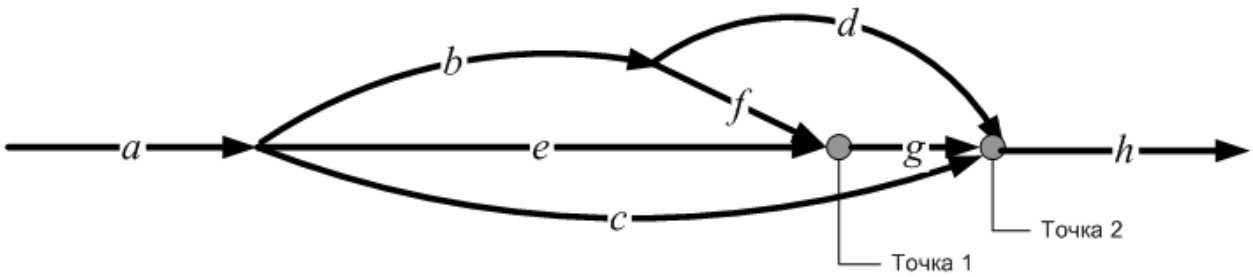
Вариант 6:



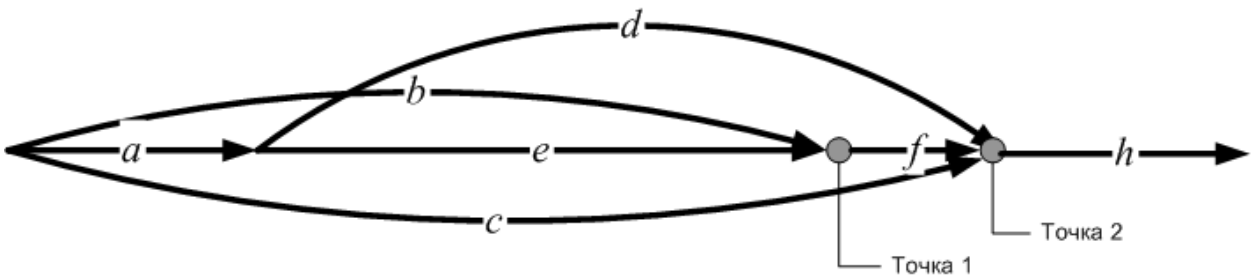
Вариант 7:



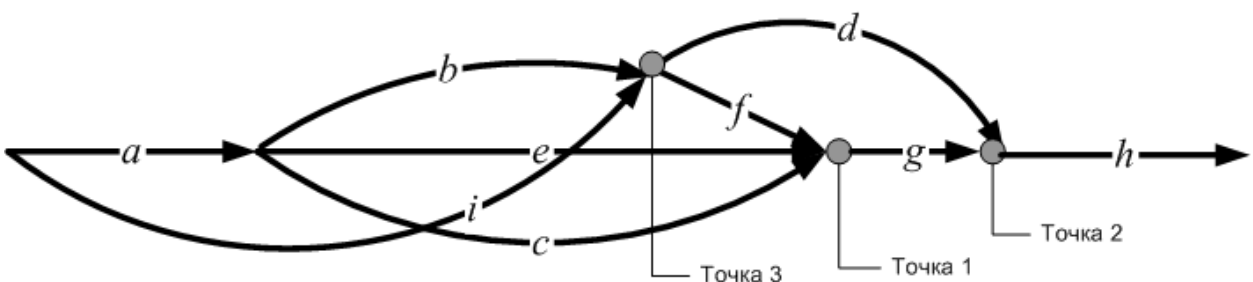
Вариант 8:



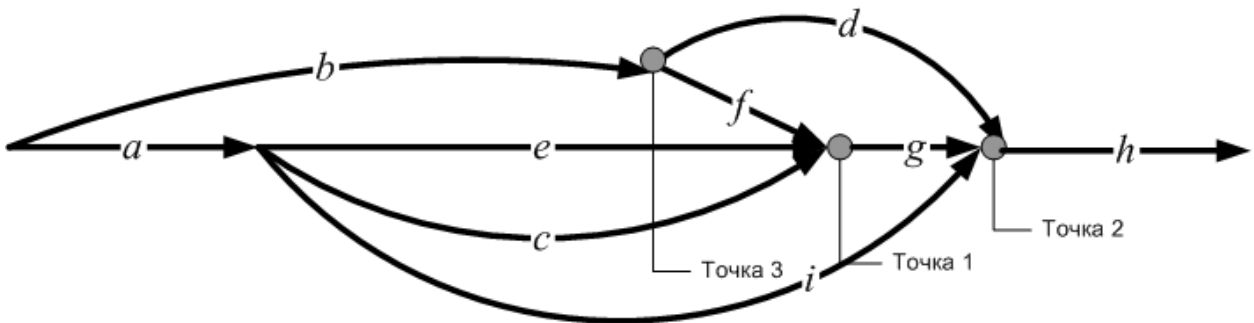
Вариант 9:



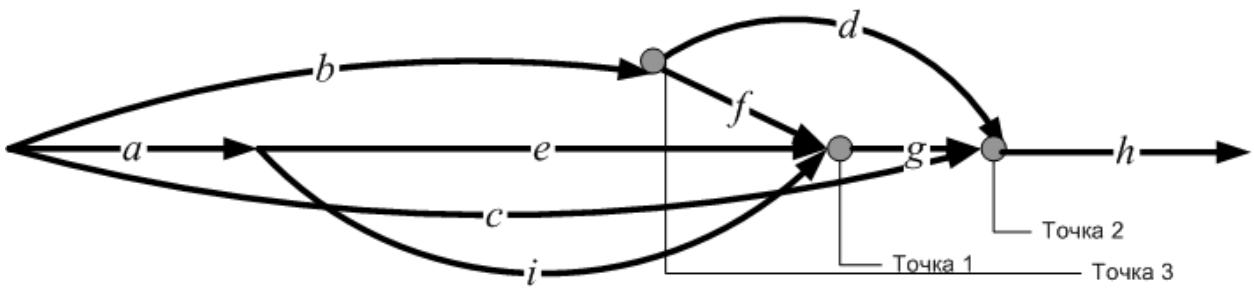
Вариант 10:



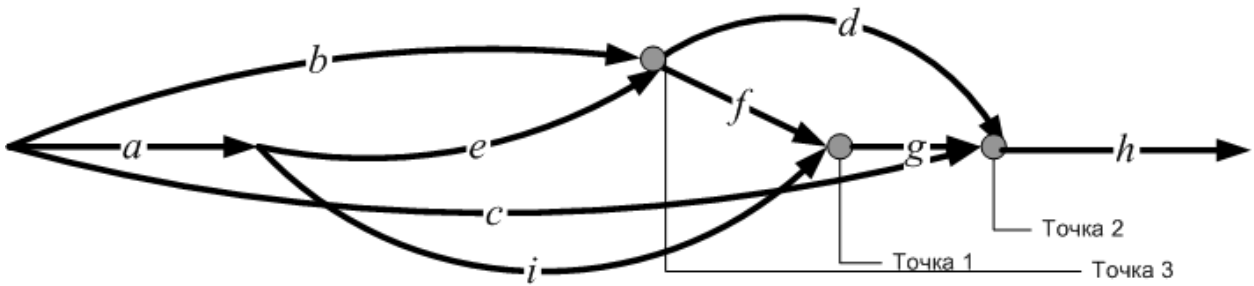
Вариант 11:



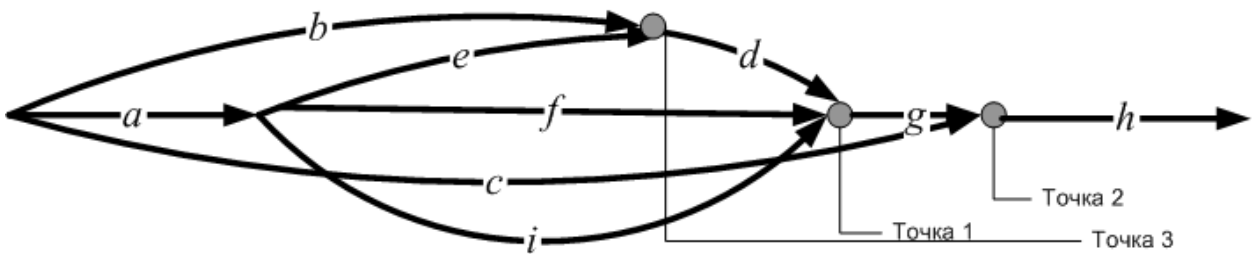
Вариант 12:



Вариант 13:

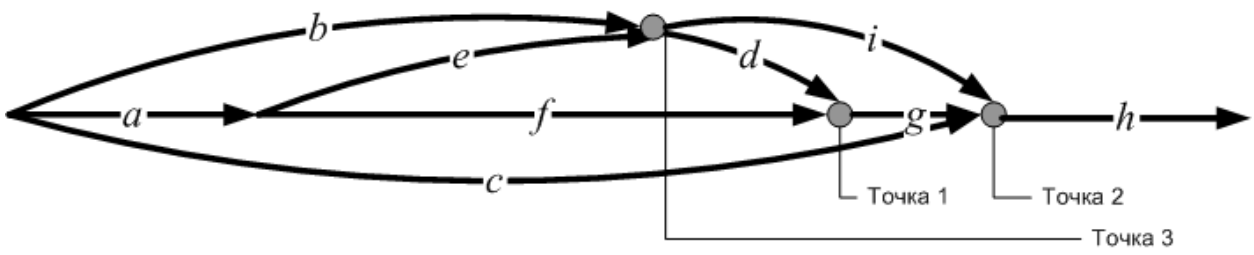


Вариант 14:

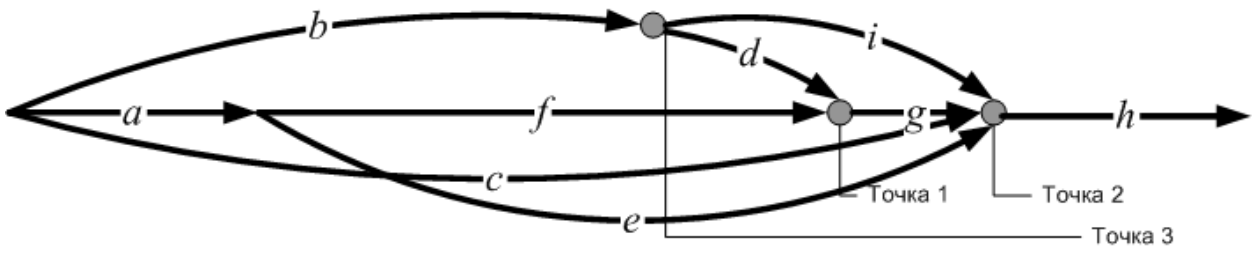


Вариант 15:

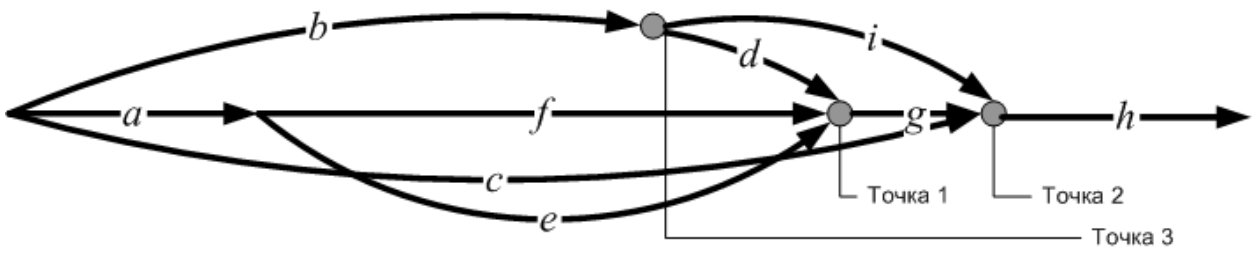
31



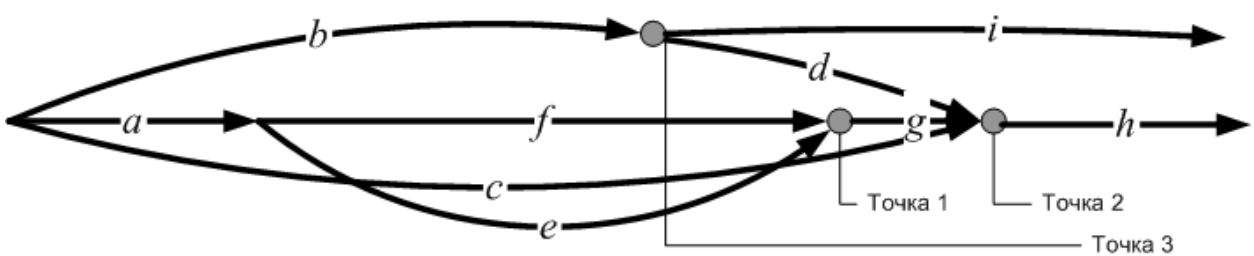
Вариант 16:



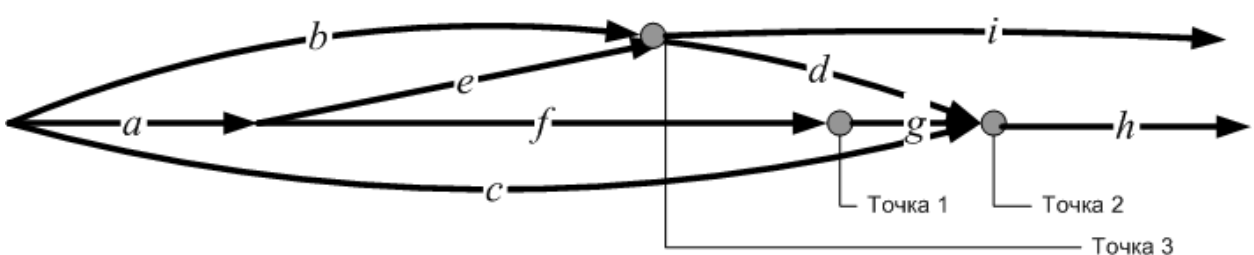
Вариант 17:



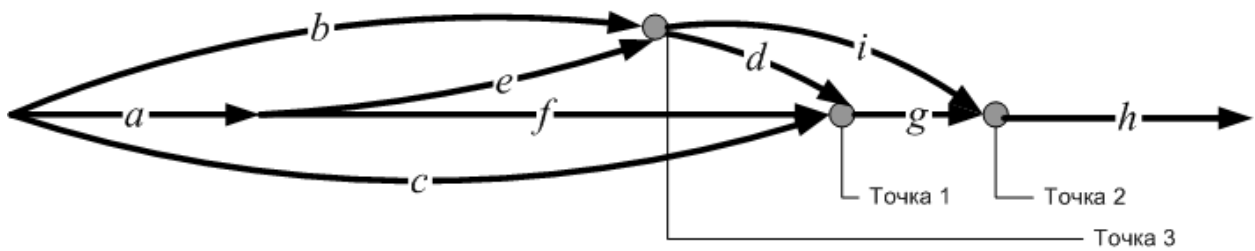
Вариант 18:



Вариант 19:



Вариант 20:



4.3. Содержание отчета

1. Вариант задания (рисунок). Описание части отведенной на усмотрение студента.
2. Листинг программы.
3. Вид главного окна программ в некоторые моменты её исполнения.

5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дать определение понятию «нить».
2. Отличие нитей от процессов.
3. Проблемы, возникающие при использовании нитей.
4. Принципы и механизмы создания многонитевых приложений и их синхронизации в среде Delphi.
5. Приведите пример вычислительного процесса, соответствующего индивидуальному варианту заданий.

6. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Кнут, Д. Искусство программирования на ЭВМ. Т.2: Получисленные алгоритмы [Текст]/ Дональд Кнут – М.: Вильямс, 2008 – 185 с.
2. Кауфман, В. Ш. Языки программирования. Концепции и принципы [Текст] / В. Ш. Кауфман – М.: ДМК Пресс, 2010 – 464с.
3. Ахо, А.В. Компиляторы. Принципы, технологии и инструментарий [Текст] / Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман – М.: Вильямс, 2008.– 1184 с., ил.
4. Борисенко, В. В. Основы программирования [Текст] / В. В. Борисенко – М.: Интернет-университет информационных технологий, 2005 – 318 с.
5. Харт, Д. М. Системное программирование в среде Windows [Текст] / Джонсон М. Харт – М.: Вильямс, 2005 – 592 с.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования «Юго-Западный государственный
университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ

Проректор по учебной работе

О.Г. Локтионова

« 3 » 03

2023 г.



УСТАНОВКА И АДМИНИСТРИРОВАНИЕ ОПЕРАЦИОННОЙ СИСТЕМЫ FREEBSD

Методические указания по выполнению лабораторных работ для
студентов укрупненной группы специальностей и направлений
подготовки 10.00.00

Курск 2023

УДК 004

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры «Информационная безопасность» А.Л. Марухленко

Установка и администрирование операционной системы FreeBSD: методические указания по выполнению лабораторной работы по дисциплине «Безопасность операционных систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В. Митрофанов. Курск, 2023. 40с. Библиогр.: с. 40.

Излагаются методические указания об администрировании, управлении и настройке систем на ОС FreeBSD. Указывается порядок выполнения лабораторной работы, правила оформления, содержание отчета.

Методические указания по выполнению лабораторных работ по дисциплине «Безопасность операционных систем», предназначены для студентов укрупненной группы специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____ . Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 150. Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

Содержание	3
1. Цель работы	4
2. Установка операционной системы FreeBSD	4
2.1. Загрузка программы–установщика.....	4
2.2. Утилита sysinstall.....	5
2.2.1. Начало стандартной установки (Standart).....	6
2.2.2. Выделение дискового пространства.....	6
2.2.3. Установка менеджера загрузки (Boot Manager).....	9
2.2.4. Создание разделов с помощью Disklabel	9
2.2.5. Выбор устанавливаемых компонентов. Выбор дистрибутивного набора (Distribution Set).....	11
2.2.6. Выбор источника для установки.....	11
2.2.7. Подтверждение установки.....	12
3. Пользователи и основы управления учетными записями.....	13
3.1. Идентификационная информация пользователей	13
3.2. Типы учётных записей	14
3.2.1. Учетная запись суперпользователя	14
3.2.2. Системные учетные записи	14
3.2.3. Учетные записи пользователей.....	15
3.3. Изменение учетных записей	15
3.3.1. Команда adduser.....	15
3.3.2. Команда gmuser	16
3.3.3. Команда chpass	17
3.3.4. Команда passwd	18
4. Файлы и каталоги	19
4.1. Создание и удаление файлов.....	19
4.2. Каталоги	20
4.3. Копирование, перемещение, переименование файлов.....	22
4.4. Создание имен файлов	23
4.5. Экранирование спецсимволов	25
4.6. Справочные материалы	26
5. Файлы и права доступа	29
5.1. Перенаправление ввода-вывода.....	29
5.2. Стандартные файлы-устройства.....	32
5.3. "Владение" файлом и "права" на него.....	33
5.4. Маска прав по умолчанию.....	36
5.5. Особенности прав на каталоги.....	37
5.6. Справочные материалы	37
6. Задание на лабораторную работу	38
7. Содержание отчёта	39
8. Вопросы для самопроверки	39
9. Библиографический список.....	40

1. ЦЕЛЬ РАБОТЫ

Изучение процедуры инсталляции операционной системы FreeBSD. Изучение средств администрирования пользователей в операционной системе FreeBSD.

2. УСТАНОВКА ОПЕРАЦИОННОЙ СИСТЕМЫ FREEBSD

2.1. Загрузка программы–установщика

1. Включите компьютер. Войдите в меню установки BIOS (с помощью клавиш **F2**, **F10**, **Del**, или **Alt+S**).

2. Найдите установки системы, указывающие ей с какого устройства загружаться. Измените установки, указав в качестве загрузочного диска привод CDROM. Сохраните изменения и перезагрузите компьютер, вставив в привод компакт–дисков установочный диск FreeBSD.

3. На экране увидите информацию следующего содержания (информация о версии удалена):

```
Verifying DMI Pool Data .....
Boot from ATAPI CD-ROM :
  1. FD 2.88MB  System Type-(00)
Uncompressing ... done

BTX loader 1.00 BTX version is 1.01
Console: internal video/keyboard
BIOS drive A: is disk0
BIOS drive B: is disk1
BIOS drive C: is disk2
BIOS drive D: is disk3
BIOS 639kB/261120kB available memory

FreeBSD/i386 bootstrap loader, Revision
0.8

/kernel text=0x277391
data=0x3268c+0x332a8 |
```

4. В процессе загрузки появится:

```
Hit [Enter] to boot immediately, or any
other key for command prompt.
Booting [kernel] in 9 seconds... _
```

5. Подождите десять секунд или нажмите **Enter**.

Через некоторое время появится диалог инсталлятора как на рис.

1.



Рис. 1 - Выбор страны.

После выбора страны переходим к выбору раскладки клавиатуры (KeuMap). Это необходимо только при использовании нестандартной клавиатуры. Для изменения раскладки выберите из меню с помощью клавиш навигации необходимый пункт и нажмите **Enter**.

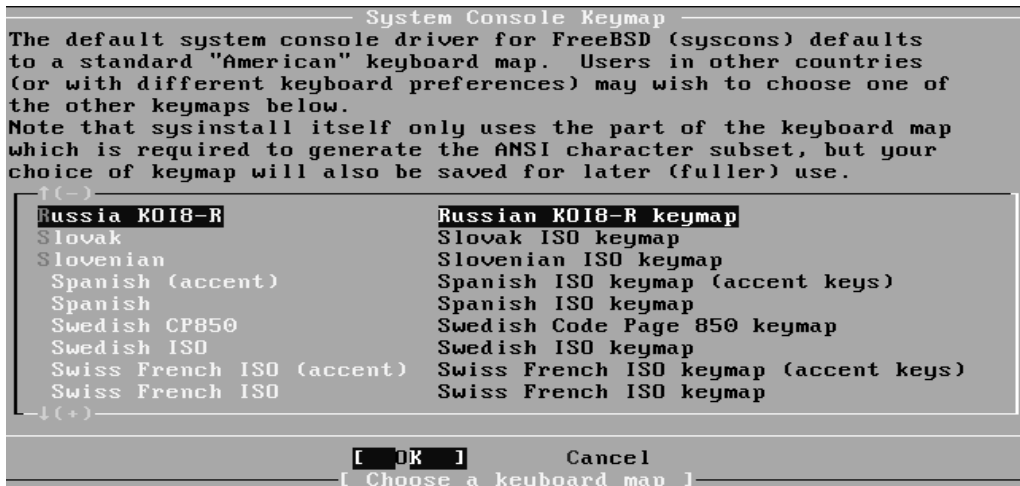


Рис. 2 - Выбор меню раскладки клавиатуры.

6. Выбираем пункт Russia KOI8-R.

2.2. Утилита sysinstall.

Следующий шаг установки – работа с утилитой sysinstall (рис.3).

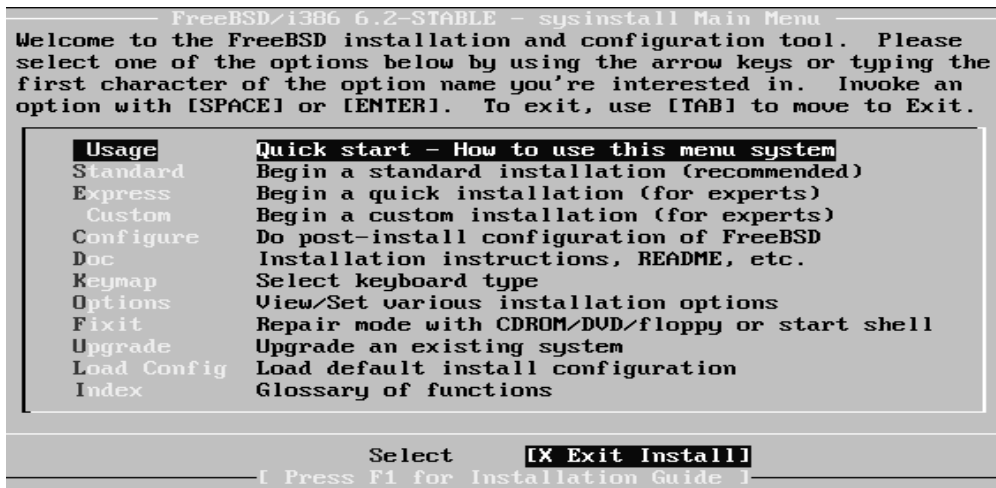


Рис. 3 - Утилита sysinstall.

Утилита **sysinstall** это программа установки, предоставляемая проектом FreeBSD. Это консольное приложение, разделенное на несколько меню и экранов. Меню **sysinstall** управляется клавишами навигации, **Enter**, пробелом, и другими.

2.2.1. Начало стандартной установки (Standart)

Выберете пункт **Standard**, нажмите **Enter** для запуска установки.

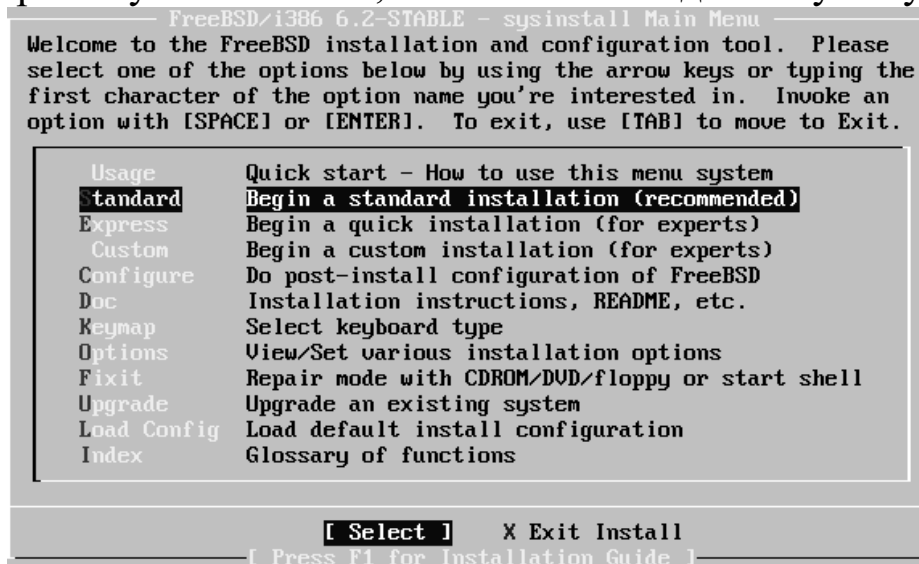


Рис. 4 - Начало стандартной установки

2.2.2. Выделение дискового пространства

Перед установкой операционной системы необходимо выделить дисковое пространство под FreeBSD и разметить его, чтобы **sysinstall** могла его подготовить.

В PC, работающем под BIOS-зависимой операционной системой, такой как или Microsoft® Windows®, BIOS может отходить от обычного порядка нумерации дисков. FreeBSD не использует BIOS, и не знает о "логическом отображении дисков в BIOS". При использовании FreeBSD всегда восстанавливайте настройки BIOS к первоначальной нумерации перед установкой системы и оставляйте их в таком виде.

После начала стандартной установки будет показан список всех жестких дисков, обнаруженных ядром во время тестирования устройств. Рис. 5. показывает пример системы с двумя IDE дисками.

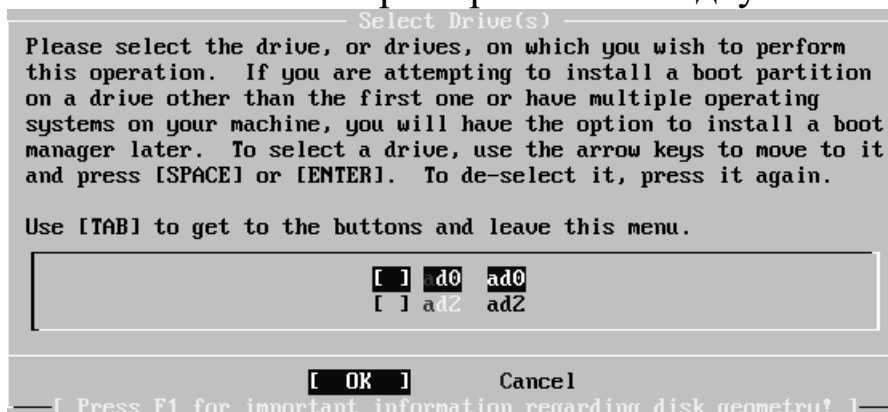


Рис. 5 - Выбор диска для Fdisk

Вы должны выбрать диск, на который хотите установить FreeBSD, и нажать [OK] (рис. 6).

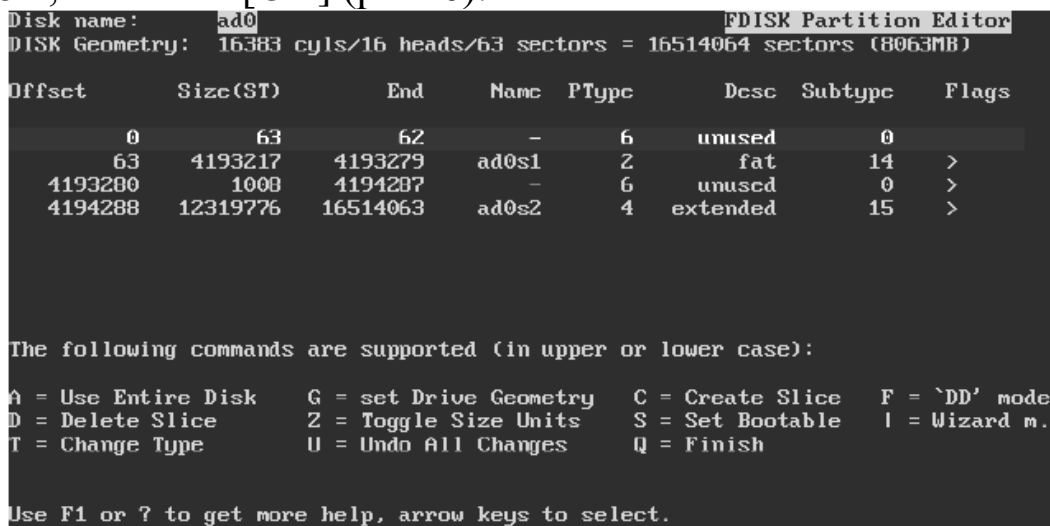


Рис. 6 - Разделы fdisk перед редактированием

Экран **FDisk** разбит на три раздела. Первый раздел показывает подробную информацию о выбранном в данный момент диске, включая его имя, геометрию и общий размер. Второй раздел показывает имеющиеся в данный момент на диске слайсы, где они

начинаются и заканчиваются, их размер, имя, которое им дала FreeBSD, описание и подтип. Третий раздел показывает команды, доступные в **Fdisk**.

Если вы хотите использовать для **FreeBSD** весь диск (это приведет к удалению всех других данных на нём), нажмите **A**, что соответствует опции «Использовать весь диск» (Use Entire Disk). Существующие слайсы будут удалены, и заменены на небольшую область, помеченную как неиспользуемая (unused), и один большой слайс для FreeBSD (рис. 7). После этого нужно выбрать вновь созданный слайс FreeBSD и нажать **S**, чтобы сделать его загрузочным.

```
Disk name: ad0 FDISK Partition Editor
DISK Geometry: 16383 cyls/16 heads/63 sectors = 16514064 sectors (8063MB)

Offset      Size(ST)      End      Name PType      Desc Subtype  Flags
-----
0           63           62      -    6      unused    0
63      16514001    16514063  ad0s1 3      freebsd   165    CA

The following commands are supported (in upper or lower case):
A = Use Entire Disk      G = set Drive Geometry  C = Create Slice      F = 'DD' mode
D = Delete Slice        Z = Toggle Size Units   S = Set Bootable     I = Wizard m.
T = Change Type         U = Undo All Changes    Q = Finish

Use F1 or ? to get more help, arrow keys to select.
```

Рис. 7 - Разбиение в Fdisk с использованием всего диска

Если вы будете удалять слайс для освобождения места под **FreeBSD**, выберите слайс, и нажмите **D**. Затем можете нажать **C**, и получить приглашение на ввод размера слайса, который вы хотите создать. Введите соответствующее значение и нажмите **Enter**. Значение по умолчанию в этом поле означает наибольший размер слайса, который может быть выбран; это может быть наибольший непрерывный блок неразмеченного пространства или размер всего жесткого диска.

В нашем случае FreeBSD ставится на освобождённый заранее слайс. Для этого надо нажать **C** для создания нового слайса. Будет также предложено ввести размер слайса, который вы хотите создать. Затем нажмите **Q**. Изменения будут сохранены в **sysinstall**, но еще не записаны на диск.

2.2.3. Установка менеджера загрузки (Boot Manager)

Далее необходимо установить менеджер загрузки. Это необходимо, если:

- у вас больше чем один диск и вы устанавливаете FreeBSD не на первый диск;
- Вы устанавливаете **FreeBSD** вместе с другой операционной на один и тот же диск, и хотите выбирать при загрузке FreeBSD или другую операционную систему.

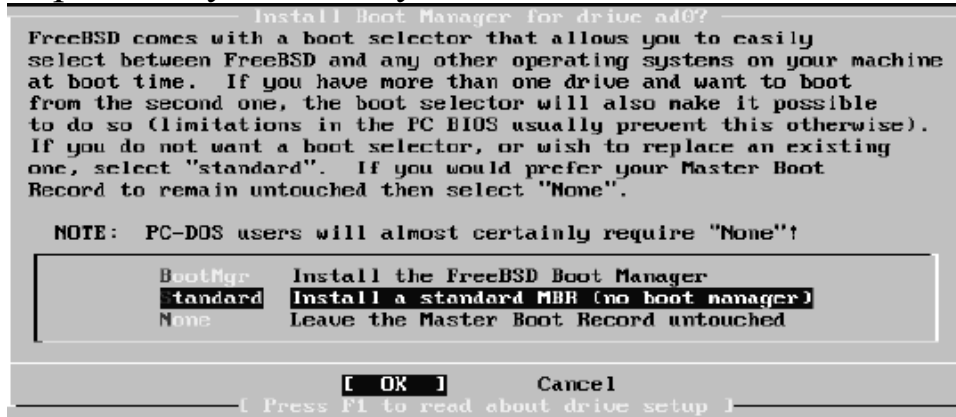


Рис. 8 - Меню менеджера загрузки Sysinstall

2.2.4. Создание разделов с помощью Disklabel

Теперь необходимо создать несколько разделов внутри только что созданного слайса. У каждого раздела есть буква с a до h, а разделы b, c, и d имеют соглашения, которых необходимо придерживаться.

Нижеприведённая схема показывает четыре раздела: один для подкачки и три для файловых систем.

Таблица 1 – Планирование разделов для диска.

Раздел	Файловая система	Размер	Описание
a	/	100 MB	Корневая файловая система. Любая другая файловая система будет смонтирована на эту. 100 MB это подходящий размер для этой файловой системы.
b	N/A	2-3xRAM	Раздел подкачки.

Раздел	Файловая система	Размер	Описание
e	/var	50 MB	Каталог /var содержит файлы, которые постоянно меняются; логи и другие административные файлы. Многие из этих файлов интенсивно читаются и записываются в процессе работы FreeBSD. Размещение их на отдельной файловой системе позволяет FreeBSD оптимизировать доступ к ним.
f	/usr	Остальная часть диска	Все другие файлы хранятся в каталоге /usr и его подкаталогах.

Начало разметки разделов начинается с запуска редактора разделов FreeBSD, называемого **Disklabel** (рис. 9).

```

FreeBSD Disklabel Editor
Disk: ad0      Partition name: ad0s1  Free: 16514001 blocks (8063MB)
Part  Mount      Size Newfs  Part  Mount      Size Newfs
----  -
The following commands are valid here (upper or lower case):
C = Create      D = Delete    M = Mount pt.
N = Newfs Opts  Q = Finish    S = Toggle SoftUpdates
T = Toggle Newfs U = Undo      A = Auto Defaults  R = Delete+Merge
Use F1 or ? to get more help, arrow keys to select.
    
```

Рис. 9 - Редактор Sysinstall Disklabel

Экран **Disklabel** поделен на три раздела. Первые несколько линий показывают имя диска, с которым вы работаете и слайс, содержащий раздел, который вы создаете. Этот экран также показывает объем свободного пространства на слайсе, т.е. пространство, выделенное под слайс, но еще не отданное под раздел.

В центре экрана показаны созданные разделы, имена файловых систем, содержащихся в разделах, их размер и опции, применяемые при создании файловых систем.

Нижняя треть экрана показывает управляющие клавиши.

Disklabel может автоматически создать разделы и присвоить им размеры по умолчанию (нажимается **A**). Однако рекомендуется создать разделы вручную (клавиша **C**)

2.2.5. Выбор устанавливаемых компонентов. Выбор дистрибутивного набора (Distribution Set)

При первой установке рекомендуется включить все устанавливаемые компоненты (при условии достаточности места на диске, как в нашем случае). Для этого надо выбрать **All** и нажать **Enter** (рис 10).

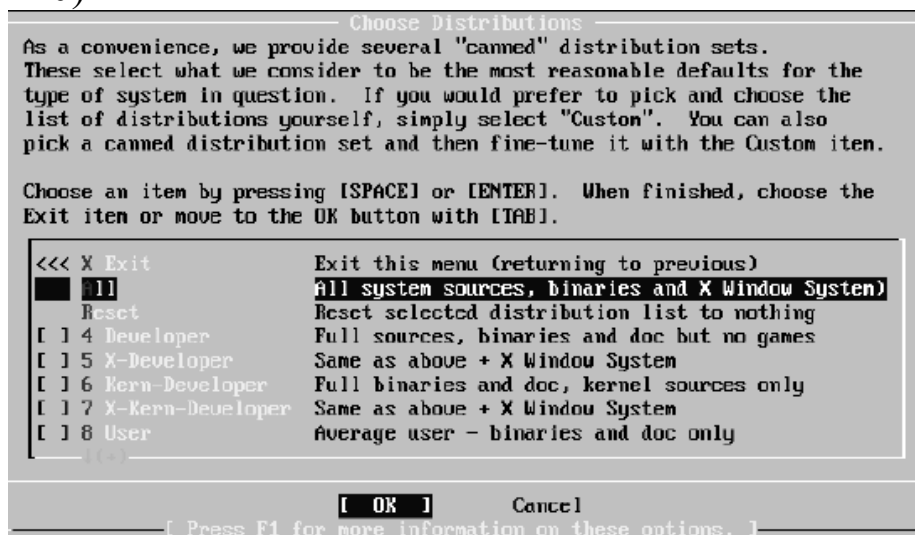


Рис. 10 - Выбор дистрибутивных наборов

2.2.6. Выбор источника для установки

При установке с CD-ROM используйте клавиши навигации, для перехода к пункту Install from a FreeBSD CD/DVD (рис. 11).

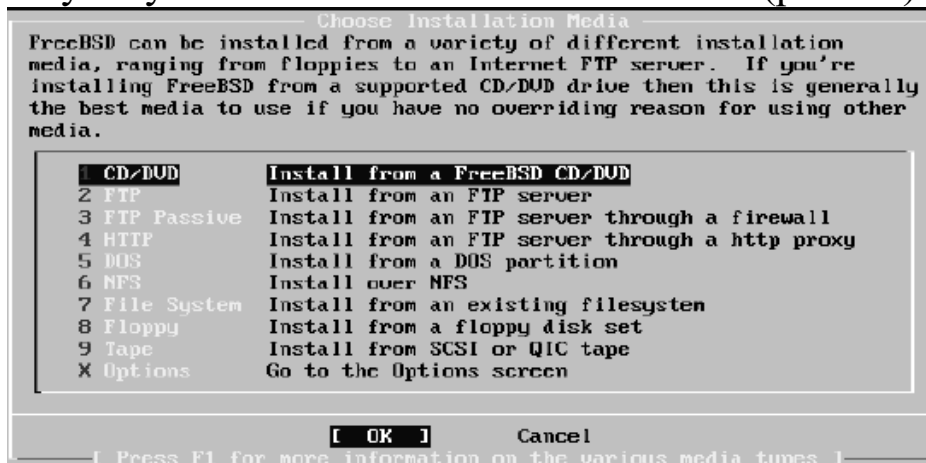


Рис. 11 - Выбор источника установки

2.2.7. Подтверждение установки

Появится сообщение о подтверждении начала установки:

```
User Confirmation Requested
Last Chance! Are you SURE you want to
continue the installation?

If you're running this on a disk with
data you wish to save then WE STRONGLY
ENCOURAGE YOU TO MAKE PROPER BACKUPS
before proceeding!
We can take no responsibility for lost
disk contents!

[ Yes ]      No
```

Выберите [Yes] и нажмите **Enter**. Установка будет завершена, когда отобразится следующее сообщение:

```
Message

Congratulations! You now have FreeBSD
installed on your system.

We will now move on to the final
configuration questions.
For any option you do not wish to
configure, simply select No.

If you wish to re-enter this utility
after the system is up, you may
do so by typing: /stand/sysinstall .
[ OK ]

[ Press enter to continue ]
```

Нажмите **Enter**, чтобы вернуться в главное меню установки и выйти из нее. Извлеките все установочные диски и дискеты. После выхода из утилиты **sysinstall** компьютер перезагрузится.

После загрузки системы вы увидите стандартное приглашение:

```
login:
```

Введите имя пользователя *root* (учетная запись суперпользователя, которая после установки не имеет пароля). Установите пароль суперпользователя (см. ниже).

3. ПОЛЬЗОВАТЕЛИ И ОСНОВЫ УПРАВЛЕНИЯ УЧЕТНЫМИ ЗАПИСЯМИ

FreeBSD позволяет одновременную работу множества пользователей на одном компьютере. Для использования системы у каждого пользователя должна быть учетная запись.

3.1. Идентификационная информация пользователей

С каждой учетной записью в системе FreeBSD связана определенная идентификационная информация.

Имя пользователя. Имя пользователя в том виде, в каком оно вводится в приглашение `login:`. Имена пользователей должны быть уникальны в пределах одного компьютера.

Пароль. С каждой учетной записью связан пароль. Пароль может быть пустым, в этом случае для доступа к системе не нужен пароль.

ID пользователя (User ID, UID). The UID это номер от 0 до 65535, используемый для однозначной идентификации пользователя в системе. Сама система FreeBSD для идентификации пользователей использует UID – любая команда FreeBSD, позволяющая вам указывать имя пользователя, первым делом преобразует его к UID. Вы можете создать несколько учетных записей с различными именами, но с одним UID. FreeBSD будет воспринимать эти учетные записи как одного пользователя.

ID группы (Group ID, GID). GID это номер от 0 до 65535, используемый для однозначной идентификации группы, к которой принадлежит пользователь. Группы – это механизм для контроля доступа к ресурсам на основе GID пользователя вместо его UID. Пользователь может быть включен более чем в одну группу.

Класс логина. Классы логинов это расширение к механизму групп, позволяющее системе более гибко управлять различными пользователями.

Полное имя пользователя. Имя пользователя является уникальным идентификатором учетной записи, но недостаточно для

сопоставления с реальным пользователем. Эта информация может быть добавлена в учетную запись.

Домашний каталог. Домашний каталог это полный путь к каталогу в системе, в котором пользователь начнет работать после входа в систему. По общепринятому соглашению все домашние каталоги пользователей помещаются в `/home/username` или `/usr/home/username`. Пользователи хранят личные файлы в домашнем каталоге и в любых подкаталогах, создаваемых внутри домашнего каталога.

Оболочка пользователя. Оболочка необходима как средство взаимодействия с системой. Существует множество различных видов оболочек.

3.2. Типы учётных записей

Существует три основных типа учетных записей:

- учетная запись суперпользователя,
- системные учетные записи,
- учетные записи пользователей.

3.2.1. Учетная запись суперпользователя

Учетная запись суперпользователя, называемая `root`, существует в системе изначально для целей системного администрирования, и не должна использоваться для повседневных задач. Суперпользователь, в отличие от обычных пользователей, может работать без ограничений. Учетные записи пользователей не способны уничтожить систему вследствие ошибки.

Необходимо тщательно проверять команды, выполняемые под учетной записью суперпользователя, поскольку даже один лишний пробел или отсутствующий символ может привести к безвозвратной потере данных.

3.2.2. Системные учетные записи

Системные пользователи предназначены для запуска сервисов, таких как DNS, почта, веб серверы и так далее. Это необходимо по соображениям безопасности; если все сервисы работают от суперпользователя, они могут действовать без ограничений.

3.2.3. Учетные записи пользователей

Учетные записи пользователей в основном означают доступ в систему для обычных людей, и эти учетные записи отделяют пользователя и его рабочую среду, предотвращая повреждение пользователем системы или данных других пользователей, и позволяя пользователям настраивать свою рабочую среду без влияния на других пользователей.

Каждый пользователь может настраивать свою собственную рабочую среду для приспособления системы под свои нужды с помощью альтернативных оболочек, редакторов, привязки клавиш и настроек языка.

3.3. Изменение учетных записей

В среде UNIX® существуют различные команды для работы с учетными записями пользователей. Наиболее часто используемые команды приведены в таблице 2.

Таблица 2 – Команды управления учётными записями

Команда	Краткое описание
<code>adduser</code>	Рекомендуемое приложение командной строки для добавления пользователей.
<code>rmuser</code>	Рекомендуемое приложение командной строки для удаления пользователей.
<code>chpass</code>	Инструмент для изменения информации в базе данных пользователей.
<code>passwd</code>	Инструмент командной строки для изменения паролей пользователей.

3.3.1. Команда **adduser**

Adduser – это простая программа для добавления новых пользователей. Она создает записи в системных файлах `passwd` и `group`. Она также создает домашний каталог для нового пользователя, копируя файлы настройки по умолчанию из `/usr/share/skel` и может отправлять новому пользователю приветственное сообщение.

Пример добавления пользователя в FreeBSD

```
# adduser
Username: jru
Full name: J. Random User
Uid (Leave empty for default):
Login group [jru]:
Login group is jru. Invite jru into other
groups? []: wheel
Login class [default]:
Shell (sh csh tcsh zsh nologin) [sh]: zsh
Home directory [/home/jru]:
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation?
[no]:
Username      : jru
Password      : ****
Full Name     : J. Random User
Uid           : 1001
Class         :
Groups        : jru wheel
Home          : /home/jru
Shell         : /usr/local/bin/zsh
Locked        : no
OK? (yes/no) : yes
adduser: INFO: Successfully added (jru)
to the user database.
Add another user? (yes/no): no
Goodbye!
#
```

3.3.2. Команда **rmuser**

Данная программа выполняет следующие действия:

- Уничтожает все процессы, принадлежащие пользователю.
- Удаляет пользователя из локального файла паролей.

- Удаляет домашний каталог пользователя (если он принадлежит пользователю).
- Удаляет принадлежащую пользователю входящую почту из /var/mail.
- Удаляет все файлы, принадлежащие пользователю, из каталогов с временными файлами, например /tmp.
- Наконец, удаляет имя пользователя из всех групп, которым оно принадлежит, в /etc/group.

Rmuser не может использоваться для удаления учетной записи суперпользователя, поскольку это означает разрушение системы.

Пример интерактивного удаления учетной записи:

```
# rmuser jru
Matching password entry:
jru:*:1001:1001::0:0:J. Random
User:/home/jru:/usr/local/bin/zsh
Is this the entry you wish to remove? y
Remove user's home directory(/home/jru)?y
Updating password file, updating
databases, done.
Updating group file: trusted (removing
group jru -- personal group is empty)
done.
Removing user's incoming mail file
/var/mail/jru: done.
Removing files belonging to jru from
/tmp: done.
Removing files belonging to jru from
/var/tmp: done.
Removing files belonging to jru from
/var/tmp/vi.recover: done.
#
```

3.3.3. Команда **chpass**

Команда изменяет информацию в базе данных пользователей: пароли, оболочки, персональную информацию. Только системные администраторы с правами суперпользователя могут изменять информацию и пароли других пользователей с помощью **chpass**.

При запуске без параметров программа вызывает редактор, содержащий информацию о пользователе. Когда пользователь выходит из редактора, база данных пользователей обновляется этой информацией.

Пример работы с **chpass** суперпользователя:

```
#Changing user database information for
jru.
Login: jru
Password: *
Uid [#]: 1001
Gid [# or name]: 1001
Change [month day year]:
Expire [month day year]:
Class:
Home directory: /home/jru
Shell: /usr/local/bin/zsh
Full Name: J. Random User
Office Location:
Office Phone:
Home Phone:
Other information:
```

Обычные пользователи могут изменять лишь часть информации, и только для своей учетной записи.

Пример работы с **chpass** обычного пользователя:

```
#Changing user database information for
jru.
Shell: /usr/local/bin/zsh
Full Name: J. Random User
Office Location:
Office Phone:
Home Phone:
Other information:
```

3.3.4. Команда **passwd**

Команда **passwd** – это обычный способ изменения собственного пароля пользователя, или пароля другого пользователя суперпользователем.

Пример изменения пароля:

```
% passwd
```

```
Changing local password for jru.  
Old password:  
New password:  
Retype new password:  
passwd: updating the database...  
passwd: done
```

Пример изменение пароля пользователя суперпользователем:

```
# passwd jru  
Changing local password for jru.  
New password:  
Retype new password:  
passwd: updating the database...  
passwd: done
```

4. ФАЙЛЫ И КАТАЛОГИ

4.1. Создание и удаление файлов

Именем файла в операционной системе **FreeBSD** может быть любая строка поддерживаемых символов, не содержащая нулевого символа и символа косой черты ("/"). Регистр букв является значимым.

Создать файл можно командой **touch**, указав ей в качестве аргумента имя несуществующего файла. До этого стоит удостовериться, что такого файла действительно не существует, с помощью команды **ls** (*list* — перечислить), выводящей список файлов, имена которых перечислены в качестве ее аргументов:

```
[user]$ ls 1_файл  
ls: 1_файл: Такого файла или каталога нет  
[user]$ touch 1_файл  
[user]$ ls 1_файл  
1_файл
```

Аргумент, начинающийся с дефиса, называется ключом команды. Большинство команд могут применяться с ключами, модифицирующими их действие. Аргумент, не являющийся ключом, называется операндом:

```
[user]$ ls -l 1_файл  
Итого: 0 -rw-r--r--      1 user      user  
0 июнь 16 23:18 1_файл
```


Ключ -l задает "длинный" формат вывода команды. Перед списком файлов выводится строка с количеством блоков (обычно 512-байтных), занимаемых перечисленными файлами. Файлу соответствует при этом строка-список из семи полей (они разделены символом табуляции, отображаемым при выводе на экран пробелом или серией пробелов, перечисленных в таблице 3:

Таблица 3 – Атрибуты файла.

Тип файла и права доступа	Кол-во указателей на файл	Имя владельца	Имя группы владельца	Размер	Время модиф-ии	Имя файла
-rw-r--r--	1	User	user	0	Июнь 16 23:18	1_файл

Поле "Размер" – это размер файла в байтах. Даже у пустого файла есть все атрибуты файла, включая время последней модификации и собственно имя.

Удалить файл можно командой `rm` с именем файла в качестве аргумента:

```
[user@localhost user]$ rm 1_файл
```

4.2. Каталоги

Если команда `ls` подается без операндов, на экран будет выведен список всех файлов каталога (аналогично команде `dir` в DOS):

```
[user]$ ls
1_файл tmp
[user]$ ls -l
Итого 1
-rw-r--r--      1 user user      0  Июнь 16
23:18 1_файл
drwx-----    2 user  user     48  Июнь 10
08:17 tmp
```

Файлы организуются в структуру, задающую их логическое расположение. Файловая структура **FreeBSD** является иерархической, как и в операционной системе Windows®: файлы

содержатся в особых файлах — каталогах; каталоги, в свою очередь, могут содержаться в других каталогах и т.д. Вершиной файловой структуры служит корневой каталог `/`.

В выводе команды `ls -l` файлы–каталоги отличаются от обычных файлов тем, что в первой позиции поля "тип файла и права доступа" стоит буква "d". Тип обычного файла обозначается символом дефиса ("-").

Полное (или абсолютное) имя файла однозначно идентифицирует конкретный файл в системе. Краткое имя однозначно идентифицирует файл лишь в отдельном каталоге. Полное имя в документации всегда указывается с ведущим слэшем (начинается с `/`); таким же образом его следует указывать в качестве аргумента команд операционной системы.

При регистрации пользователя текущим становится домашний каталог пользователя, определенный администратором при создании учетной записи, этого пользователя. Обычно домашние каталоги пользователей создаются в каталоге `/home/` с именами, совпадающими с регистрационными именами пользователей. Узнать свой домашний каталог пользователь может в любой момент, подав команду `echo ~`.

Создать каталог можно командой `mkdir`, а удалить пустой (не содержащий файлов) каталог — командой `rmdir` с именем каталога в качестве параметра:

```
[user]$ mkdir 1_каталог
[user]$ ls -l
Итого 1
-rw-r--r--      1 user user      0  Июн 16
23:18 1_файл
drwxr-xr-x      2 user  user     48  Июн 22
21:03 1_каталог
drwx-           3 user  user     72  Июн 20 18:27
tmp
[user]$ rmdir 1_каталог
[user]$ ls -l
-rw-r--r--      1 user user      0  Июн 16
23:18 1_файл
drwx-           3 user  user     72  Июн 20 18:27
tmp
```

Сделать каталог текущим можно командой **cd** (*change directory*) с именем каталога в качестве параметра:

```
[user]$ cd 1_каталог
[1_каталог]$ ls
[1_каталог]$ pwd
/home/user/1_каталог
```

После смены текущего каталога, его краткое имя появилось в подсказке. Узнать полное имя текущего каталога можно, подав команду **pwd** без параметров.

"Подняться" на одну ступень по иерархии каталогов можно, используя специальное имя каталога ".", содержащееся в любом каталоге. Его нет в списках файлов, выводившихся по команде **ls**, поскольку это имя начинается с точки и файл является "скрытым". Увидеть имена скрытых файлов можно командой **ls -a**:

```
[user]$ ls -a
.  ..
[user]$ ls -a -l
Итого: 2
drwxr-xr-x    2 user  user    48  Июнь 22
21:03  .
drwx-----    5 user  user   528  Июнь 22
21:35  ..
```

В данном случае пустой каталог содержит два файла–каталога: "." и "..". Первый является самим каталогом, в котором он содержится, второй – каталогом на ступень выше в иерархии. Два ключа (-a и -l) можно указать в сокращенной форме — дефис и следующие за ним буквы ключей без пробела (в данном случае -al). Перейти в домашний каталог из любой вершины в файловой структуре можно командой **cd** без параметров.

4.3. Копирование, перемещение, переименование файлов

Команда **cp** служит для копирования файлов, команда **mv** – для перемещения. Обе команды имеют два отличающихся по семантике варианта.

- Если последним операндом является имя существующего каталога, то файлы, имена которых указаны в качестве предшествующих операндов, копируются или перемещаются в этот каталог.

- Если последним операндом является имя обычного файла, то файл, имя которого указано в качестве предшествовавшего операнда, копируется или переименовывается в этот файл:

```
[user]$ mkdir 1_каталог
[user]$ cd 1_каталог
[ещё_каталог]$ touch 1_файл 2_файл 3_файл
[ещё_каталог]$ mkdir 2_каталог
[1_каталог]$ ls
2_каталог 1_файл 2_файл 3_файл
[ещё_каталог] mv 2_файл 3_файл 2_каталог
[ещё_каталог]$ ls
2_каталог 1_файл
[1_каталог]$ ls 2_каталог
2_файл 3_файл
[1_каталог] mv 1_файл 4_файл
[1_каталог] ls
2_каталог 4_файл
[1_каталог] cp 2_каталог/2_файл 2_каталог
/3_файл .
[1_каталог] ls
2_каталог 2_файл 3_файл 4_файл
[1_каталог] ls 2_каталог
2_файл 3_файл
```

Указание в этих командах единственного операнда, а также указание более двух операндов в случае, если последний из них не является именем существующего каталога, - ошибка.

Чтобы избежать случайного удаления файлов, если при копировании или перемещении файлов их имена совпадают с именами существующих, можно использовать ключ `-i`:

```
[1_каталог]$ cp -i 2_каталог/2_файл .
Ср: переписать './второй'? у
[1_каталог]$ mv -i 2_файл 2_каталог
mv: переписать '2_каталог/2_файл'? у
```

4.4. Создание имен файлов

Во многих случаях операции выполняются не над одним, а над целым списком файлов. оболочка реализует механизм для указания списков имен файлов, если эти имена синтаксически схожи (начинаются с одной буквы, заканчиваются одним расширением и

т.п.). Этот механизм заимствован многими альтернативными системами, но в отличие от большинства из них в открытых системах раскрытие метасимвола осуществляется оболочкой, а не командой.

Вопросительный знак ("?") соответствует любому одному символу в имени файла. Если у нас в каталоге присутствуют файлы *a1, a2, a3, b, b2, b3, aa1*, шаблон имени (метаимя) "a?" раскроется в список *a1 a2 a3*, а шаблон "?1" — в *a1 b1*.

Звездочка ("*") соответствует последовательности из нуля или большего количества любых символов. В том же каталоге "a*" раскроется в список *a1 a2 a3 aa1*, а "*1" - в *a1 b1 aa1*.

Метаконструкция из последовательности символов, заключенных в квадратные скобки ("[" и "]"), соответствует любому одному символу из этой последовательности. В том же каталоге "[abc] 2" раскроется в список *a2 b2*. В квадратных скобках могут содержаться диапазоны, разделенные дефисом ("–"). Они означают любой символ, входящий в этот диапазон, с учетом алфавитного порядка следования символов. В нашем случае "[a-c] 3" раскроется в *a3 b3*.

Список может быть предварен знаком отрицания "^", в этом случае он означает любой символ, не входящий в список. Если в шаблон нужно буквально включить символ "-", его следует поставить на первое или последнее место, а "^" - на любое место, кроме первого. Конструкция в квадратных скобках может быть сколь угодно сложной (например, "[a-sk-w-y]" означает — «любой символ с "a" по "с", или "к", или с "w" по "у"»), и она всегда соответствует одному символу в раскрываемых именах.

Вопросительные знаки, звездочки и квадратно-скобочные конструкции могут произвольно сочетаться. Список всегда раскрывается в алфавитном порядке.

Еще одним метасимволом является тильда ("~"), выступающая в качестве такового только когда стоит первой в аргументе. Отдельная тильда раскрывается в полное имя домашнего каталога текущего пользователя. Тильда, за которой без пробела следует регистрационное имя пользователя, раскрывается в полное имя его домашнего каталога. Если оболочке не удастся раскрыть метасимвол, он передается команде в буквальном виде:

```
[1_каталог]$ echo ~  
/home/user  
[user] echo ~rabbit
```

```
/home/rabbit  
[user] echo ~bob  
~bob
```

4.5. Экранирование спецсимволов

Специальное значение символов "?", "*", "[", "]", "~" при указании имен файлов и является причиной, по которой их не рекомендуется вводить в имена файлов. Однако пользователь может столкнуться с ситуацией, в которой ему все же нужно выполнить некоторые действия с файлом, чье имя содержит такие символы.

В каталог *файлы/* перенесен файл *страница[13].htm*. Как к нему обратиться?

Буквальное указание в командной строке цепочки символов, совпадающей с именем файла будет интерпретировано как список, состоящий из шаблона "страница [13].htm" (которому могут соответствовать файлы *страница1.htm* и *страница3.htm*).

В лучшем случае эти файлы не будут найдены, в худшем будут найдены другие файлы, чьи названия случайно совпадут с элементами невольно введенного "списка" или результатами раскрытия "шаблона".

Чтобы указать в командной строке файл, чье имя содержит специальные символы, эти символы необходимо экранировать, т.е. "защитить" от раскрытия. Экранировать отдельный символ можно, поставив перед ним символ обратной косой черты ("\"). Цепочка "страница\[13\].htm" раскрывается в цепочку "страница [13].htm":

Экранировать обратной косой чертой можно любой специальный символ. Если необходимо, чтобы в цепочке был раскрыт сам символ "\", он также экранируется:

Другой способ экранировать специальные символы от интерпретации как шаблонных – заключить имя файла в апострофы или кавычки:

```
[файлы]$ ls "страница [13].htm"  
страница[13].htm
```

Хотя, используя экранирование, можно создавать, перемещать, копировать и уничтожать файлы, имена которых состоят практически из любых символов, включать специальные (как шаблонные, так и прочие) символы в имена файлов крайне не рекомендуется, так как это повышает вероятность ошибки при вводе.

Оболочка не придает никакого особого значения точке в имени файла (кроме случаев, когда имя начинается с точки), и "расширение имени файла" — это лишь интерпретация пользователя (и, возможно, некоторых программ). Поэтому, в отличие от ряда альтернативных систем, шаблон "*" означает не "все файлы с именами без расширений", но буквально "все файлы с именами, заканчивающимися на точку".

4.6. Справочные материалы

touch - изменить временные атрибуты доступа и модификации файлов

Синтаксис: touch [-act] [-r *справ_файл* | -t *время*] *файл...*

Семантика: touch изменяет атрибуты времени последней модификации или времени последнего доступа файлов, или (по умолчанию) оба. Значение атрибута указывается аргументом ключа -t или заимствуется у файла, указанного в качестве аргумента ключа -r. Если оно не указано, используется текущее время.

Если файлы не существуют, они создаются.

Ключи: -a - изменить время доступа; -c - не создавать несуществующих файлов; -m - изменить время модификации; -r *справ_файл* - заимствовать атрибут у файла *справ_файл*; -t *время* - использовать вместо текущего указанное время в формате "[[ВВ]ГГ]ММДДччмм[.сс]", где ММ - номер месяца, ДД - день месяца, чч - час дня, мм - минуты, ВВ - первые две цифры года, ГГ - последние две цифры года, сс - секунды.

Операнды: *файл* - имя файла.

ls - вывести содержимое каталога

Синтаксис: ls [-CFRacdilqrtul] [-H | -L] [-fgmnoptsx] [*файл...*]

Семантика: Для каждого операнда, именуемого файл типа иного, нежели каталог или ссылка на каталог, ls выводит имя и требуемую ключами информацию. Для каждого операнда, именуемого каталог или ссылку на каталог, ls выводит имена и требуемую ключами информацию о каждом файле, содержащемся в этом каталоге. Если операнды не указаны, ls выводит информацию о файлах в текущем каталоге. Для ссылок на каталоги выводится информация о каталоге, если даны ключи -d, -F или -l и не даны ключи -H или -L, и информация о файлах в каталоге, если не даны ключи -d, -F или -l или даны ключи -H или -L.

Важнейшие ключи: -R - выводить рекурсивно информацию о подкаталогах; -a - включить информацию о скрытых файлах (файлах с именами, начинающимися на точку); -l ("эль") - выводить информацию в "длинном" формате; -p - выводить после имен каталогов "/"; -t - отсортировать в порядке времени изменения.

Операнды: файл— имя файла.

Переменные: COLUMNS - количество столбцов на терминале; TZ - часовой пояс.

Вывод: по умолчанию выводится по одной записи в строке. -l - выводятся тип и права файла, количество ссылок, имя владельца, имя группы, длина файла, дата и время, имя файла.

rm - удалить записи о файлах

Синтаксис: rm [-fiRr] файл...

Семантика: rm удаляет запись в каталоге для каждого операнда за исключением файлов "." или ".." в любом каталоге и за исключением (если не даны ключи -г, -R) каталогов.

Ключи: -f - не запрашивать подтверждения; -i - запрашивать подтверждение; -г, -R - рекурсивно удалять содержимое указанных каталогов.

Операнды: файл - имя файла.

Вывод ошибок: стандартный вывод ошибок используется для вывода запросов на подтверждение удаления файлов (-i).

mkdir - создать каталог

Синтаксис: mkdir [-p] [-m права] каталог...

Семантика: mkdir создает перечисленные каталоги.

Операнды: каталог - создаваемый каталог.

rmdir - удалить каталоги

Синтаксис: rmdir [-p] каталог...

Семантика: rmdir удаляет записи, соответствующие указанным пустым каталогам.

Операнды: каталог - удаляемый каталог.

cp - копировать файлы

Синтаксис: cp [-fir] исх_файл цел_файл ; cp [-fir] исх_файл... каталог ; cp -R [-H | -L | -P] [-fir] исх_файл... каталог ; cp -г [-H | -L | -P] [-fir] исх_файл... каталог

Семантика: первая синтаксическая форма характеризуется двумя файлами, ни один из которых не должен быть существующим каталогом. cp копирует исх_файл в цел_файл. Если исх_файл - символическая ссылка, копируется целевой файл этой ссылки.

Вторая синтаксическая форма характеризуется двумя или более операндами, отсутствием ключей -R или -r и неприменимостью первой формы. Исходные файлы не должны быть каталогами, а каталог должен быть существующим каталогом. `cp` копирует исходные файлы в указанный каталог под именами, совпадающими с краткими именами исходных файлов.

Третья и четвертая формы характеризуются двумя или более операндами и ключами -r или -R. `cp` копирует все указанные файлы, а также рекурсивно каталоги с их содержимым в каталог.

Важнейшие ключи: -i - запрашивать подтверждение перед копированием в существующие файлы; -p - сохранять по возможности времена изменения и доступа к файлу, владельца и группу, права доступа; -R, -r - рекурсивно копировать содержимое каталогов.

Операнды: `исх_файл` - исходный файл; `цел_файл` - целевой файл; `каталог` - целевой каталог.

Стандартный вывод ошибок: стандартный вывод ошибок используется для вывода запросов на подтверждение перезаписи существующих файлов (-i).

mv - переместить файлы

Синтаксис: `mv [-fi] исх_файл цел_файл ; mv [-fi] исх_файл... каталог`

Семантика: в первой синтаксической форме, характеризующейся тем, что последний операнд не является ни каталогом, ни символической ссылкой на каталог, `mv` перемещает `исх_файл` в `цел_файл`.

Во второй синтаксической форме `cp` копирует исходные файлы в указанный каталог под именами, совпадающими с краткими именами исходных файлов.

Ключи: -f - не запрашивать подтверждения перезаписи существующих файлов; -i - запрашивать подтверждение перезаписи существующих файлов.

Операнды: `исх_файл` - исходный файл; `цел_файл` - целевой файл; `каталог` - целевой каталог.

Стандартный вывод ошибок: стандартный вывод ошибок используется для вывода запросов на подтверждение перезаписи существующих файлов (-i).

echo - вывести аргументы

Синтаксис: `echo [строка...]`

Семантика: `echo` выводит свои аргументы после раскрытия специальных символов в стандартный вывод, завершая вывод символом новой строки.

Операнды: строка - строка, подлежащая выводу. В строке после раскрытия спецсимволов оболочки раскрываются следующие символы: `"\a"` - звуковой сигнал, `"\b"` - пробел, `"\c"` - подавить вывод символа новой строки, `"\f"` - перевод страницы, `"\n"` - символ конца строки, `"\r"` - символ возврата каретки, `"\t"` --табуляция, `"\v"` - вертикальная табуляция, `"\"` - обратная косая черта, `"\0код"` - символ с восьмеричным кодом код.

Стандартный вывод: между аргументами выводятся пробелы.

5. ФАЙЛЫ И ПРАВА ДОСТУПА

5.1. Перенаправление ввода-вывода

Команды `cp`, `ls`, `mkdir`, `mv`, `rm`, `rmdir`, `touch`, являются "файловыми утилитами". Любые действия файловых утилит совершенно безразличны к содержимому файлов.

Команда `cat` относится к "текстовым утилитам". Поданная без аргументов команда не приводит ни к какому видимому результату: не появляется вывод на экране, нет подсказки, которая была бы знаком успешного завершения.

Команда `cat` *вводит* данные. Каждая введенная ею строка после нажатия [Enter] вновь выводится на терминал. Команда `cat`, поданная без аргумента, копирует содержимое ввода в вывод построчно. Предусмотрен ключ `-i`, обеспечивающий побайтное копирование.

Закончить ввод можно, введя в начале очередной строки символ конца файла `Ctrl+D`. Ввод этого символа приводит к немедленному завершению ввода.

```
[user@localhost user]$ cat
Введение в ОС
Введение в ОС
LINUX
LINUX
^D
[user]$
```

Одним из самых важных свойств открытых операционных систем является возможность *перенаправления ввода-вывода*.

В примере *перенаправляется вывод* команды `cat`, используя символ `>` ("`>`" – «в файл») со следующим за ним именем файла (**ОС**), в который перенаправляется вывод:

```
[user]$ cat > ОС
Введение в ОС
LINUX
^D
[user]$ls -l ОС
-rw-r--r-1 1 user user 20 Июн 23
3:09 ОС
```

В этом примере строки ввода уже не дублируются выводом на терминал, а поданная после завершения ввода команда `ls` показывает, что появился файл с названием **ОС** и размером 20 байт, что соответствует длине введенного текста.

Перенаправляется может и ввод. С помощью команды `cat` можно вывести содержимое созданного ею файла на терминал. Перенаправление ввода осуществляется указанием имени файла после символа `<` ("`<`" – "из файла"):

```
[user] $ cat < ОС
Введение в ОС
LINUX
```

Промежутки до и после символов `<` и `>` игнорируются.

Перенаправление ввода и вывода может использоваться и одновременно. Команда копирует построчно файл **ОС** в файл **ОС2**.

```
[user] $ cat < ОС > ОС2
[user] $ ls -l ОС*
-rw-r--r-1 1 user user 20 Июн 23
03:25 ОС2
-rw-r--r-1 1 user user 20 Июн 23
03:09 ОС
```

Результат совпадает с результатом копирования `cp ОС ОС2`, хотя и достигается другим способом. Порядок указания файлов перенаправления ввода и вывода значения не имеет.

Если файл, в который перенаправляется вывод, уже существует, он будет опустошен. оболочка при этом может запрашивать подтверждение на опустошение файла.

Часто бывает желательно перенаправить вывод в файл, не уничтожая его содержимого, а *дописывая* новые строки к

существующим. Для этого оболочка поддерживает *перенаправление вывода в конец существующего файла*, обозначаемое "двойной стрелкой" ">>":

```
[user] $ cat >>OC
FREEBSD
^D
[user] $ cat <OC
Введение в ОС
LINUX
FREEBSD
```

Перенаправление вывода в файл, в конец файла может применяться совместно с переназначением ввода.

Возможность переназначения ввода-вывода является свойством не отдельных программ, а системы в целом. Символы перенаправления и соответствующие имена файлов *не* передаются самим командам в качестве аргументов. Оболочка самостоятельно назначает файлы ввода-вывода любой команды. В частности, можно переназначить вывод любой из команд, например, **ls**:

```
[user] $ ls OC* >список
[user] $ cat <список
OC
OC2
```

Кроме стандартного ввода и стандартного вывода, каждая команда открывает еще один файл для вывода ошибок. Команда записывает в стандартный вывод то, что от нее ожидается, а в вывод пишет сообщения об ошибках, предупреждения и диагностические сообщения. Как и первые два файла, по умолчанию вывод ошибок ассоциирован с терминалом. Перенаправление стандартного ввода и стандартного вывода не влияет на вывод ошибок:

```
[user] $ ls OC3 >список
ls: OC3: No such file or directory
[user] $ ls -l список
-rw-r--r-- 1 user user 0  Июнь 23
22:34 список
```

В примере сообщение об отсутствии файла было выведено на терминал, хотя стандартный вывод был перенаправлен в файл *список*.

Перенаправить вывод ошибок можно, используя конструкцию **2>** со следующим за ней именем файла:

```
[user@localhost user] $ ls OC3 2>список
[user@localhost user] $ cat <список
ls: OC3: No such file or directory
```

Двойка перед символом перенаправления в этой конструкции означает дескриптор канала ввода-вывода; стандартный ввод и стандартный вывод имеют дескрипторы 0 и 1, соответственно (дескрипторы 0, 1, 2 известны также как `STDIN`, `STDOUT`, `STDERR`), а запись "<"; ">", ">>" является сокращением от "0<", "1>", "1>>", соответственно (команда может открывать и большее количество файлов ввода-вывода, но лишь три из них ассоциируются с терминалом автоматически при ее подаче).

Так же, как и стандартный вывод, вывод ошибок может быть переназначен в конец существующего файла конструкцией `2>>`. Если же необходимо переназначить и стандартный вывод, и вывод ошибок в один файл, в командную строку следует, помимо переназначения стандартного вывода, включить еще и конструкцию `2>&1`, означающую "переназначить второй канал туда же, куда и первый":

5.2. Стандартные файлы-устройства

Бывает желательно подавить стандартный вывод или вывод ошибок вообще. Для этого в любой стандартной системе существует специальный файл, представляющий собою фиктивное "нуль-устройство". Его полное имя `/dev/null`. Запись в него любых данных не приводит к какому-либо результату, они как бы "бесследно исчезают". В `/dev/null` можно переназначить как стандартный вывод, так и вывод ошибок.

На `/dev/null` можно также переназначить и стандартный ввод; из него всегда читается пустой файл: подача команды `cat </dev/null >пустой_файл` приведет к появлению в текущем каталоге пустого файла *пустой_файл* (или опустошению существующего файла с таким именем).

Еще один специальный файл-устройство – `/dev/tty`. Это абстрактное устройство, соответствующее терминалу, с которого запущена оболочка. Перенаправление вывода или ввода из этого устройства не дает никакого видимого эффекта, поскольку совпадает с умолчанием, - но его указание может пригодиться для команд, вводящих текст из файла, указанного в качестве операнда, или выводящих текст в такой файл. Можно считать, что, если в команде

явным образом не присутствуют перенаправления, ввод-вывод неявным образом направлен так: `</dev/tty >/ dev/tty 2>/dev/tty`.

Если взглянуть на перечисленные файлы с помощью команды `ls -l`, мы обнаружим, что в поле тип присутствует символ «с», означающий "устройство с посимвольным вводом-выводом":

```
[user] $ ls -l /dev/null /dev/tty
crw-rw-rw- 1 root root 1,3 Янв 20
19:24 /dev/null
crw-rw-rw- 1 root root 5,0 Июн 24
20:11 /dev/tty
```

Такие файлы устройств присутствуют в любой стандартной открытой системе, так же как и содержащий их каталог `/dev/`. В большинстве реализаций этот каталог содержит также множество файлов, представляющих физические или виртуальные устройства. Любое устройство в открытой ОС представлено в виде файла. Некоторые из них (терминалы) представляют собой устройства с посимвольным вводом-выводом, другие (магнитные диски) – с поблочным. Тип файла-устройства с поблочным вводом-выводом обозначается буквой «b».

5.3. "Владение" файлом и "права" на него

Рассмотрим пример, в котором командами **touch** и **ls** создадим файл *файл* и убедимся в том, что он создан. Затем выполним команду `chmod u-w файл` и обнаружим, что поле тип и права доступа в выдаче "длинного" списка файлов претерпело изменения. Попытка перенаправить в этот файл ввод порождает сообщение оболочки "В доступе отказано".

```
[user] $ touch файл
[user] $ ls -l файл
-rw-rw-rw- 1 user user 0 Июл 1 10:42
файл
[user] $ chmod u-w файл
[user] $ ls -l файл
-r--r--r-- 1 user user 0 Июл 1 10:42
файл
[user] $ cat >файл
-bash: файл: в доступе отказано
```

Значения полей в "длинном" формате списка файлов, получаемого по команде `ls -l`, приведены в таблице 3 (см. выше). Поле тип раскрыто в таблице 4.

Таблица 4 – Устройство поля тип и права.

Тип файла	Права владельца			Права группы			Права остальных пользователей		
	Чтен.	Зап.	Исполн.	Чтен.	Зап.	Исполн.	Чтен.	Зап.	Исполн.
-	r	-	-	r	-	-	r	-	-

Оно всегда содержит десять символов. Значение первого символа: это *тип файла*, которому могут соответствовать "-" (обычный файл), d (каталог) и т. д. Остальные девять символов составляют три триады, выражающие *права на файл* в так называемой "rwx"-нотации. Они соответствуют трем категориям пользователей, определяемым относительно каждого файла:

- "Владелец" файла. Это пользователь, чье имя указано в соответствующем (третьем) поле "длинного" формата списка файлов. Обычно владелец файла – это создавший его пользователь.
- Пользователи, входящие в группу пользователей владельца.
- Все остальные пользователи.

Для каждой категории определяются отдельные "правомочия" доступа к файлу:

- Правомочие чтения разрешает чтение содержимого файла. Значение этого символа может быть "-" (запрещено) или "r" (разрешено; *read* - читать).
- Правомочие записи разрешает модификацию файла, его значение может быть "-" (запрещено) или "w" (разрешено; *write* - писать).
- Правомочие исполнения разрешает выполнение программы, содержащейся в файле, путем указания ее имени. Значение этого бита может быть "-" (запрещено) или "x" (разрешено). В этом же поле отображаются и другие правомочия. Если в этом поле присутствует другая буква, строчная буква (например, "s") означает исполняемый файл, а заглавная (например, "S") – неисполняемый.

Ограничения системы прав на файлы не действуют для суперпользователя (root). Он может читать любые файлы в структуре,

писать любые файлы, кроме расположенных в подструктурах, смонтированных только для чтения, и исполнять любые файлы, исполнение которых разрешено хотя бы одной категории пользователей.

Изменять права доступа к файлу могут лишь его владелец и суперпользователь. Для этого служит команда `chmod` (*change mode* - изменить режим доступа к файлу). Синтаксис этой команды поддерживает две нотации - символическую и числовую.

Символическая нотация представляет собой операнд-слитную запись клауз из трех составляющих: категории пользователя, вид назначения прав и собственно назначаемые правомочия. Операнд может включать в себя более одной клаузы, разделенных запятыми (*без промежутков*).

Категории пользователей соответствуют описанным выше и обозначаются последовательностями букв:

"u" владелец файла (*user* - пользователь);

"g" группа - владелец файла (*group* - группа);

"o" остальные пользователи (*other* - прочие);

"a" все пользователи (*all* - все), это сокращенная запись для "ugo".

Вид назначения прав может быть трояким:

"+" добавить правомочия;

"-" отнять правомочия;

"=" установить права, в точности соответствующие назначаемым.

Назначаемые правомочия обозначаются последовательностями уже известных нам букв "rwx"-нотации "r", "w", "x", соответствующим правам на чтение, запись и исполнение.

Таким образом,

chmod u-w файл отнимет правомочие записи у владельца;

chmod g+rw файл добавит правомочия чтения и записи группе-владельцу;

chmod go=r установит правомочия группы-владельца и прочих пользователей в точности равными "только чтению";

chmod a+x добавит правомочие исполнения всем пользователям;

chmod u=rwx, g=rw, o=r установит правомочия чтения, записи и исполнения для пользователя, чтения и записи для группы и чтения для всех остальных.

5.4. Маска прав по умолчанию

Когда пользователь создает файл, права доступа к нему устанавливаются равными *маске прав по умолчанию*, за исключением того, что правомочие исполнения обычному файлу не присваиваются. Права по умолчанию задаются командой **umask**.

Команда **umask -S** без параметров выводит в символическом виде маску прав по умолчанию. Команда **umask** с параметром в "ugo"-нотации (такой же, как; у команды **chmod**) добавляет, отнимает или устанавливает права в маске прав.

```
[user] $ umask -S
u=rwx,g=rx,o=rx
[user] $ touch файл_1
[user] $ ls -l файл_1
-rw-r--r-- 1 user user 0 Июл 1 13:43
файл_1
[user] $ umask o-r
[user] $ umask -S
u=rwx,g=rx,o=x
[user] $ touch файл_2
[user] $ ls -l файл_2
-rw-r----- 1 user user 0 Июл 1 13:43
файл_2
```

В примере выводится маска, создается файл *файл_1*, убеждаются в том, что права на вновь созданный файл соответствуют маске, отнимают у прочих пользователей вновь создаваемых файлов правомочие чтения, создают файл *файл_2* и убеждаются в том, что права на него соответствуют новому значению маски.

Утилита **umask** не является файловой, и изменение значения маски не влияет на права существующих файлов. Значение маски сохраняется до нового их изменения командой **umask** или конца сеанса работы с оболочкой.

5.5. Особенности прав на каталоги

В рассматриваемой операционной системе существует возможность удалять файл из каталога, даже если у вас нет прав доступа на запись в него. Дело в том, что удаление файла не является изменением его содержания. Удаление файла – это изменение

каталога, в котором он содержится, и, соответственно, разрешение или запрещение удаления файла зависит не от прав на него, но от прав на каталог (каталог - это тоже файл).

```
[user]$ umask -S
u=rwx,g=rx,o=rx
[ user]$ mkdir каталог_1
[user]$ touch каталог_1/файл_1
[ user] $ chmod u-w каталог_1/файл_1
[user]$ rm каталог_1/файл_1
rm: удалить файл 'каталог_1/файл_1'? y
[user]$ touch каталог_1/файл_1
[user]$ chmod u-w каталог_1/
[user]$ rm каталог_1/файл_1
rm: удалить файл 'каталог_1/файл_1'? y
rm: невозможно удалить
'каталог_1/файл_1': Недостаточно прав
[user]$ touch каталог_1/файл_2
touch: создание 'каталог_1/файл_2':
Недостаточно прав
```

В примере выше создан каталог *каталог_1*, в нем создан файл *файл_1*, у владельца отобраны права на запись, тем не менее он удаляет его, затем создает такой же файл и отнимает у себя права на запись в этот каталог. После этого попытка удаления файла приводит к выводу сообщения о нехватке прав для совершения этой операции.

Соответственно, и создать файл в каталоге, прав записи на который нет, невозможно. Отсутствие права записи в каталог не отнимает права на изменение содержимого находящихся в нем файлов.

5.6. Справочные материалы

cat - вывести содержимое файлов

Синтаксис: cat [-u] [файл...]

Семантика: cat последовательно выводит содержимое файлов.

Ключ: -u - читать и выводить файлы побайтно (по умолчанию - построчно).

Операнды: *файл* - выводимый файл. Если файл не указан, читается стандартный ввод. Если в списке файлов присутствует имя "-", вместо этого файла читается стандартный ввод.

Реализация: в большинстве систем ключ -u не реализован.

chmod - изменить права на файл

Синтаксис: chmod [-R] режим файл...

Семантика: chmod изменяет биты режима доступа к каждому указанному файлу в соответствии с указанным режимом. Изменить режим доступа к файлу может только процесс с действующим идентификатором пользователя, совпадающим с владельцем файла, или привилегированный процесс.

Ключ: -R - рекурсивно изменять режим доступа к файлам, расположенным в указанных каталогах.

Операнды: режим - устанавливаемый режим доступа (в "rwx"-или числовой нотации); файл - имя файла.

umask - вывести или установить маску прав доступа

Синтаксис: umask [-S] [маска]

Семантика: umask устанавливает маску прав вновь создаваемых в окружении текущей оболочки файлов в соответствии с указанным значением. Если операнд *маска* не указан, umask выводит текущую маску.

Ключ: -S - вывести маску в "rwx"-нотации.

Операнд: маска - маска прав в "rwx"- или числовой нотации.

6. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Студенты определяются в бригады по 2 человека
2. Установить операционную систему FreeBSD на выделенный пустой раздел
3. Создать трех пользователей с именами:
<инициалы первого студента>_<инициалы второго студента>_1
<инициалы первого студента>_<инициалы второго студента>_2
<инициалы первого студента>_<инициалы второго студента>_3
первые два пользователя принадлежат группе group_<инициалы первого студента>_1, третий – группе group_<инициалы первого студента>_3.
4. В домашнем каталоге каждого пользователя, создать несколько файлов.
В каталоге пользователя 1 создается 3 файла file1 доступен для чтения всем, для редактирования, только членам группы пользователя 1, file2 доступен только для чтения всем,

5. В домашнем каталоге второго пользователя изменить владельца одного из файлов (изменение владельца файла доступно только суперпользователю).

6. В домашнем каталоге первого пользователя создать подкаталог. Запретить его модификацию всеми пользователями кроме владельца. Повернуть запрещение другими пользователями.

7. СОДЕРЖАНИЕ ОТЧЁТА

1. Титульный лист.
2. Цель работы.
3. Описание процедуры установки операционной системы.
4. Скриншоты процедур создания пользователей, наделения их правами, проверки правомочности доступа.

8. ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Опишите процедуру установки операционной системы FreeBSD.
2. Что такое слайсы?
3. Для чего нужна учётная запись суперпользователя?
4. Назовите основные команды оболочки для управления учётными записями пользователей.
5. Назовите основные команды для работы с каталогами и файлами.
6. Что такое маска прав каталога и как она устанавливается?

9. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Алексей, Викторович Федорчук Доступный UNIX: Linux, FreeBSD, DragonFlyBSD, NetBSD, OpenBSD / Алексей Викторович Федорчук. - М.: БХВ-Петербург, 2017. - 558 с.
2. Маккузик, М.К. FreeBSD: архитектура и реализация / М.К. Маккузик, Д.В. Невилл-Нил. - М.: КУДИЦ-Образ, 2015. - 800
3. Федорчук, А. Доступный UNIX. Linux, FreeBSD, DragonFlyBSD, NetBSD, OpenBSD / А. Федорчук. - М.: БХВ-Петербург, 2016. - 872 с.
4. Эбен FreeBSD. Энциклопедия пользователя / Эбен, Таймэн Майкл; , Брайан. - К.: ООО ТИД ДС, 2013. - 736 с.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования «Юго-Западный
государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ

Проректор по учебной работе

О.Г. Локтионова

« 23 » 03

2023 г.



МОДЕЛИРОВАНИЕ ДОСТУПА К РАЗДЕЛЯЕМОМУ РЕСУРСУ

Методические указания по выполнению лабораторных и
практических работ для студентов укрупненной группы
специальностей и направлений подготовки 10.00.00

Курск 2023

УДК 681.3

Составители: М.О. Таныгин, А.В. Митрофанов

Рецензент

Кандидат технических наук, доцент кафедры
«Информационная безопасность» А.Л. Марухленко

Моделирование доступа к разделяемому ресурсу:
методические указания по выполнению лабораторных и
практических работы по дисциплине «Безопасность операционных
систем» / Юго–Зап. гос. ун-т; сост. М.О. Таныгин, А.В.
Митрофанов. Курск, 2023. 15 с. Библиогр.: с. 15.

Излагаются методические указания по выполнению работы на
персональной ЭВМ. Изучаются методы моделирования доступа к
разделяемым ресурсам с помощью программных сред.

Методические указания по выполнению лабораторных и
практических работ по дисциплине «Безопасность операционных
систем», предназначены для студентов укрупненной группы
специальностей и направлений подготовки 10.00.00

Текст печатается в авторской редакции

Подписано в печать _____ . Формат 60×84 1/16.
Усл.печ.л. . Уч.-изд.л. . Тираж 50 экз. Заказ 146 . Бесплатно
Юго–Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

СОДЕРЖАНИЕ

Содержание.....	3
1. Цель работы.	5
2. Краткая теория.....	5
2.1. Проблема межпроцессного взаимодействия.....	5
2.2. Критическая секция	7
2.3. Методы предотвращения гонок.....	7
2.3.1. Запрет на прерывания.....	7
2.3.2. Переменные блокировки.....	8
2.3.3. Алгоритм Петерсона.....	9
2.4. Классические проблемы межпроцессного взаимодействия.....	10
3. Выполнение лабораторной работы	11
3.1. Задание на лабораторную работу.....	11
3.2. Индивидуальные варианты заданий.....	12
3.3. Обеспечение монопольного использования разделяемого ресурса	14
3.4. Указания к выполнению работы.....	14
3.5. Содержание отчёта.....	14
4. Контрольные вопросы.....	15
5. БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	16

1. ЦЕЛЬ РАБОТЫ.

Изучить и научиться применять на практике механизмы работы с критическими секциями в операционной системе приложениями, а также механизмы предотвращения гонок при обращении к разделяемым ресурсам.

2. КРАТКАЯ ТЕОРИЯ.

2.1. Проблема межпроцессного взаимодействия.

Процессам часто нужно взаимодействовать друг с другом, например, один процесс может передавать данные другому процессу, или несколько процессов могут обрабатывать данные из общего файла. Во всех этих случаях возникает проблема синхронизации процессов, которая может решаться приостановкой и активизацией процессов, организацией очередей, блокированием и освобождением ресурсов.

Проблема разбивается на три компонента: передача информации от одного процесса к другому; контроль над деятельностью процессов (гарантии «непересечения» процессов в критических ситуациях); согласование действий процессов – если процесс А отвечает за поставку данных, а процесс В за их вывод на печать, то процесс В должен подождать, пока не поступят данные от процесса А.

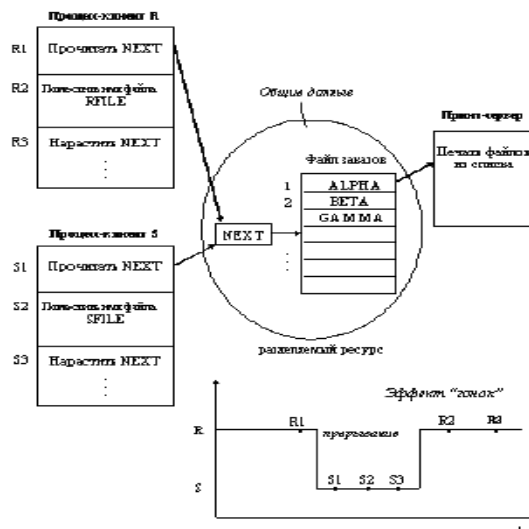


Рисунок 2.1. Пример необходимости синхронизации

Рассмотрим, например (Рисунок 2.1), программу печати файлов (принт-сервер). Эта программа печатает по очереди все файлы, имена которых последовательно в порядке поступления записывают в специальный общедоступный файл "заказов" другие программы. Особая переменная NEXT, также доступная всем процессам-клиентам, содержит номер первой свободной для записи имени файла позиции файла "заказов". Процессы - клиенты читают эту переменную, записывают в соответствующую позицию файла "заказов" имя своего файла и наращивают значение NEXT на единицу. Предположим, что в некоторый момент процесс R решил распечатать свой файл, для этого он прочитал значение переменной NEXT, значение которой для определенности предположим равным 4. Процесс запомнил это значение, но поместить имя файла не успел, так как его выполнение было прервано (например, в следствие исчерпания кванта). Очередной процесс S, желающий распечатать файл, прочитал то же самое значение переменной NEXT, поместил в четвертую позицию имя своего файла и нарастил значение переменной на единицу. Когда в очередной раз управление будет передано процессу R, то он, продолжая свое выполнение, в полном соответствии со значением текущей свободной позиции, полученным во время предыдущей итерации, запишет имя файла также в позицию 4, поверх имени файла процесса S. Таким образом, процесс S никогда не увидит свой файл распечатанным. Сложность проблемы синхронизации состоит в нерегулярности возникающих ситуаций: в предыдущем примере можно представить и другое развитие событий: были потеряны файлы нескольких процессов или, напротив, не был потерян ни один файл. В данном случае все определяется взаимными скоростями процессов и моментами их прерывания. Поэтому отладка взаимодействующих процессов является сложной задачей.

Ситуации подобные той, когда два или более процессов обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей процессов, называются *гонками*. Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме

мультипрограммирования, может привести к их неправильной работе или даже к краху системы.

2.2. Критическая секция

Важным понятием синхронизации процессов является понятие "критическая секция" программы (CS). *Критическая секция* – это часть программы, в которой осуществляется доступ к разделяемым данным. Для правильной совместной работы параллельных процессов и эффективного использования общих данных необходимо выполнение четырех условий:

- Два процесса не должны одновременно находиться в критических областях.
- В программе не должно быть предположений о скорости или количестве процессоров.
- Процесс, в состоянии вне критической области, не может блокировать другие процессы.
- Недопустима ситуация, в которой процесс вечно ждет попадания в критическую секцию.

Чтобы исключить эффект гонок по отношению к некоторому ресурсу, необходимо обеспечить, чтобы в каждый момент в критической секции, связанной с этим ресурсом, находился максимум один процесс. Этот прием называют взаимным исключением.

2.3. Методы предотвращения гонок.

2.3.1. Запрет на прерывания.

Простейший способ обеспечить взаимное исключение – позволить процессу, находящемуся в критической секции, запрещать все прерывания. Однако этот способ непригоден, так как опасно доверять управление системой пользовательскому процессу; он может надолго занять процессор, а при крахе процесса в критической области крах потерпит вся система, потому что прерывания никогда не будут разрешены.

С другой стороны, для ядра характерна блокировка прерываний для некоторых команд при работе с переменными или списками. Возникновение прерывания в момент, когда, например, список готовых процессов находится в неопределенном состоянии, могло бы привести к состоянию состязания. Запрет прерываний бывает полезным в самой ОС, но это решение неприемлемо в качестве механизма взаимного исключения для пользовательских процессов.

2.3.2. Переменные блокировки.

С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен (то есть ни один процесс не находится в данный момент в критической секции, связанной с данным процессом), и значение 0, если ресурс занят. Ниже показан фрагмент алгоритма процесса, использующего для реализации взаимного исключения доступа к разделяемому ресурсу D блокирующую переменную $F(D)$. Перед входом в критическую секцию процесс проверяет, свободен ли ресурс D . Если он занят, то проверка циклически повторяется, если свободен, то значение переменной $F(D)$ устанавливается в 0, и процесс входит в критическую секцию. После того, как процесс выполнит все действия с разделяемым ресурсом D , значение переменной $F(D)$ снова устанавливается равным 1.

Если все процессы написаны с использованием вышеописанных соглашений, то взаимное исключение гарантируется. Следует заметить, что операция проверки и установки блокирующей переменной должна быть неделимой. Пусть в результате проверки переменной процесс определил, что ресурс свободен, но сразу после этого, не успев установить переменную в 0, был прерван. За время его приостановки другой процесс занял ресурс, вошел в свою критическую секцию, но также был прерван, не завершив работы с разделяемым ресурсом. Когда управление было возвращено первому процессу, он, считая ресурс свободным, установил признак занятости и начал

выполнять свою критическую секцию. Таким образом, был нарушен принцип взаимного исключения, что может привести к нежелательным последствиям.

Во избежание таких ситуаций в системе команд машины можно иметь единую команду "проверка-установка", или же реализовывать системными средствами соответствующие программные примитивы, которые бы запрещали прерывания на протяжении всей операции проверки и установки. Гарантируется, что операция "проверка-установка" неделима – другой процесс не может обратиться к слову в памяти, пока команда не выполнена. Процессор, выполняющий такую команду, блокирует шину памяти, препятствуя обращениям к памяти со стороны остальных процессоров, кроме того, работа процесса не может быть прервана в ходе выполнения данной команды.

2.3.3. Алгоритм Петерсона.

В 1981 году Петерсон (G. L. Peterson) придумал алгоритм взаимного исключения, представленный ниже и состоящий из двух процедур, написанных на C.

```

#define N 2 /* Количество процессов */
int turn; /* Чья сейчас очередь? */
int interested[N]; /* Все переменные изначально равны 0 (FALSE) */

void enter_region(int process): /* Процесс 0 или 1 */
{
    int other; /* Номер второго процесса */
    other = 1 - process; /* "противоположный" процесс */
    interested[process] = TRUE; /* Индикатор интереса */
    turn = process; /* Установка флага */
    while (turn == process && interested[other] == TRUE) /* Пустой цикл */;
}

void leave_region(int process) /* Процесс, покидающий критическую область */
{
    interested[process] = FALSE; /*Индикатор выхода из критической области*/
}

```

Прежде чем обратиться к совместно используемым переменным (то есть перед тем, как войти в критическую область), процесс вызывает процедуру `enter_region` со своим номером (0 или 1) в качестве аргумента.

Поэтому процессу при необходимости придется подождать, прежде чем входить в критическую область. После выхода из критической области процесс вызывает процедуру `leave_region`, чтобы обозначить свой выход и тем самым разрешить другому процессу вход в критическую область.

Пусть процесс 0 вызывает `enter_region`, задает элементы массива и устанавливает переменную `turn` равной 0. Поскольку процесс 1 не заинтересован в попадании в критическую область, происходит возврат из процедуры. Теперь, если процесс 1 вызовет `enter_region`, ему придется подождать, пока `interested [0]` примет значение `FALSE`, а это произойдет только в тот момент, когда процесс 0 вызовет процедуру `leave_region` при покидании критической области.

Например, оба процесса вызвали `enter_region` практически одновременно. Оба запомнят свои номера в `turn`. Но сохранится номер того процесса, который был вторым, а предыдущий номер будет утерян. Предположим, что вторым был процесс 1, отсюда значение `turn` равно 1. Когда оба процесса дойдут до конструкции `while`, процесс 0 войдет в критическую область, а процесс 1 останется в цикле и будет ждать, пока процесс 0 выйдет из нее.

Реализация критических секций с вышеописанных механизмов имеет существенный недостаток: в течение времени, когда один процесс находится в критической секции, другой процесс, которому требуется тот же ресурс, будет выполнять рутинные действия по опросу блокирующей переменной, бесполезно тратя процессорное время.

2.4. Классические проблемы межпроцессного взаимодействия.

Существует четыре основных типа проблем межпроцессного взаимодействия:

Проблема монопольного использования двух или нескольких ресурсов. Для работы процессу нужно монопольно завладеть двумя ресурсами. Обладание только одним из них недостаточно, поэтому следует

избегать ситуации, когда каждый работающий процесс завладевает лишь одним ресурсом, не давая завершиться другому и не имея возможность завершиться самому (ситуация взаимоблокировки, но в данном случае она разрешается за счёт дополнительных вставок в код программ, а не средствами операционной системы)

Проблема использования буфера на одну запись. Программа – производитель пишет в буфер, программа – потребитель, читает из него. Производитель не должен писать в буфер, пока информация из него не будет прочитана потребителем, потребитель не должен читать, пока буфер пуст.

Проблема использования бесконечно большого буфера. Программа – производитель пишет в буфер, программа – потребитель, читает из него. Потребитель не должен читать из пустого буфера.

Проблема использования буфера на несколько записей. Аналогична проблеме с буфером на одну запись, но требует от производителя учёта записей в буфере.

3. ВЫПОЛНЕНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

3.1. Задание на лабораторную работу.

Необходимо в соответствии с индивидуальным вариантом задания (см. табл. 1) написать программы двух типов, содержащую потоки, предусмотрев при этом в каждой механизмы предотвращения гонок.

Программы выполняют обращения к файлам с заранее заданным периодом (рекомендуется 1 раз в 10 – 20 секунд), определяемым числом, вводимым пользователем при запуске программы, но необходимо предусмотреть средства, позволяющие пользователю принудительно выполнять обращения к файлам до истечения указанного времени (например по нажатию определённой кнопки на панели окна приложения).

Так как сами обращения к файлам достаточно быстротечны, то для того, чтобы процессы ожидания освобождения ресурсов были наглядны, в

код программы необходимо вести операторы, формирующие задержку (порядка 5 секунд) при работе с файлами.

Для визуализации состояния программы (простаивает, работает с ресурсом, ожидает освобождения ресурса) предусмотреть средства индикации в рабочем окне программы.

3.2. Индивидуальные варианты заданий.

Таблица 1. – Варианты заданий.

Номер варианта	Действия «Программы 1»	Действия «Программы 2»	Примечания
1.	Записать в конец любых двух файлов из трёх* заданную заранее текстовую строку	Записать в конец любых двух файлов из трёх заданную заранее текстовую строку	Программа открывает оба файла одновременно, а не по очереди
2.	Писать в конец рабочего файла текстовую строку	Удалять из начала рабочего файла текстовую строку	Недопустимо удаление строки из пустого файла
3.	Писать в конец рабочего файла текстовую строку	Удалять из начала рабочего файла текстовую строку	В файле не менее 2 и не более 6 строк
4.	Писать в рабочий файл строку	Выводить записанную первой программой строку на экран	Не писать в файл, если его ещё не прочитали, не читать, если ничего не записали
5.	Писать в конец рабочего файла текстовую очередную строку из заранее подготовленного файла	Инвертировать (записать задом наперёд) текстовую строку	Писать только незаписанную строку. Инвертировать только последнюю неинвертированную строку.
6.	Писать в конец рабочего файла текстовую очередную строку из заранее подготовленного файла	Конкатенировать две строки файла	Писать только незаписанную строку. Конкатенировать только две последние неконкатенированные строки.
7.	Записать в конец любых двух файлов из трёх заданную заранее текстовую строку	Записать в начало любых двух файлов из трёх заданную заранее текстовую строку	Программа открывает оба файла одновременно, а не по очереди

* Файлы выбираются случайно.

Номер варианта	Действия «Программы 1»	Действия «Программы 2»	Примечания
8.	Писать строку к конец одного файла из двух	Менять местами строки двух файлов	Не менять местами строки файлов, если хотя бы в одном из файлов все строки уже поменяны
9.	Писать строку в конец одного файла из двух	Удалить первые строки из обоих файлов	Не удалять строки, если хотя бы один из файлов пуст
10.	Писать строку к конец двух файлов	Удалить первую строку из одного файла из двух	Не писать строки, если в хотя бы одном из файлов строк больше 5
11.	Дописывать строку в файл только в нечётные позиции	Дописывать строку в файл только в чётные позиции	Не дописывать строку, если последняя позиция не удовлетворяет по чётности
12.	Дописывать строку в файл	Удаляет 6 первых строк из файла, если число строк больше 6 и кратно 3	
13.	Дописывать строку в файл только в позиции, номера которых кратны трём	Дописывать строку в файл только в позиции, номера которых не кратны 6	
14.	Писать две строки в конец файла	Удалить три последние строки из файла	Не писать строки, если в файле более 7 строк
15.	Дописывать строку в файл	Дописывать строку в предпоследнюю позицию файла	
16.	Дописывать строку в файл	Дописывать к строке, записанной 1-й программой определенную последовательность	Вторая программа на выполняет своих действий, если в файле нет необработанных строк
17.	Писать две строки в начало файла	Удалить первую и третью строки файла	Не удалять, если это невозможно
18.	Писать две строки в начало файла (в первую и третью позиции)	Удалить две строки из конца файла	Не писать строки, если в файле более 9 строк

Студенты формируют бригады по два человека в каждой. Бригада выполняет вариант, указанный преподавателем.

3.3. Обеспечение монопольного использования разделяемого ресурса

Гонки при доступе к разделяемому ресурсу предотвращаются за счёт использования механизма блокирующих переменных. Блокирующей переменной может служить содержимое какого либо служебного файла. Обращаясь к нему, программа читает его запись. Если она нулевая, то записывает ненулевую и закрывает файл, устанавливая тем самым блокирующую переменную и получая монопольный доступ к ресурсу. Если запись ненулевая, то программа периодически читает файл, ожидая его обнуления, которое выполняет только программа, завершающая работу с разделяемым ресурсом

3.4. Указания к выполнению работы.

При работе с фалами программы входят в критические секции, связанные с разделяемыми ресурсами (файлами и переменными, хранящими параметры работы с файлами). Для предотвращения гонок использовать внутренние механизмы работы с критическими секциями среды Delphi/

Рекомендуется выбирать расширение рабочих файлов «.htm», чтобы иметь возможность быстро отслеживать изменения, вносимые в них программой с помощью браузера. Чтобы иметь возможность контролировать работу программ, строки, добавляемые ими, не должны повторяться.

3.5. Содержание отчёта

- 1) Вариант задания.
- 2) Листинг программ.
- 3) Вид главного окна программ.
- 4) Временная диаграмма работы программ, отражающая их состояния в течение 3 – 4 минут работы (см. пример в табл. 2)

Таблица 2. – Образец временной диаграммы.

Время с начала работы	Состояние потока 1	Состояние потока 2	Содержимое рабочих файлов
-----------------------	--------------------	--------------------	---------------------------

--	--	--	--

4. КОНТРОЛЬНЫЕ ВОПРОСЫ.

- 1) Дать определение понятию «критическая секция».
- 2) Что такое разделяемый ресурс? Привести несколько реальных примеров использования программами разделяемых ресурсов.
- 3) Для чего необходимо предусматривать средства устранения гонок процессов?
- 4) Использование переменных блокировки при обращении к разделяемым ресурсам.
- 5) Альтернативные использованию блокирующих переменных механизмы устранения проблемы гонок процессов.
- 6) Основные преимущества и недостатки использования блокирующих переменных при устранении проблемы гонок процессов.
- 7) Алгоритм Питерсона.
- 8) Перечислить основные типы проблем межпроцессного взаимодействия и охарактеризовать каждый из них
- 9) Охарактеризовать тип проблемы межпроцессного взаимодействия, решаемой в ходе выполнения данной лабораторной работы

5. БИБЛИОГРАФИЧЕСКИЙ СПИСОК.

- 1) Таненбаум Э., Вудхал А. Операционные системы: разработка и реализация. – СПб.: Издательский дом «Питер», 2006.
- 2) Олифер В.Г., Олифер Н.А.. Сетевые операционные системы. - СПб.: Издательский дом «Питер», 2003.
- 3) Ахо В., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. - М.: Издательский дом «Вильямс», 2001.
- 4) Вильямс А. Системное программирование в Windows 2000 для профессионалов. - СПб.: Издательский дом «Питер», 2001.
- 5) Гордеев А.В., Молчанов А.Ю.. Системное программное обеспечение – СПб.: Питер, 2001. – 736с. илл.
- 6) Гордеев, А.В., Кучин, Н.В. Проектирование взаимодействующих процессов в операционных системах: Учебное пособие. – Л.: ЛИАП, 1991. - 72 с.
- 7) Кэнту, М. Delphi 5 для профессионалов. – СПб.: Издательский дом «Питер», 2001.