

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное  
образовательное учреждение высшего образования  
«Тульский государственный университет»

Институт прикладной математики и компьютерных наук  
Кафедра «Вычислительная техника»

Утверждено на заседании кафедры  
«Вычислительная техника»  
«28» января 2021 г., протокол № 8

Заведующий кафедрой

\_\_\_\_\_ А.Н. Ивутин

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
по выполнению курсовой работы  
по дисциплине (модулю)  
«Операционные системы семейства UNIX»**

основной профессиональной образовательной программы  
высшего образования – программы бакалавриата

по направлению подготовки  
09.03.01 «Информатика и вычислительная техника»

с профилем  
**«Электронно-вычислительные машины, комплексы, системы и сети»**

Формы обучения: очная, заочная

Идентификационный номер образовательной программы: 090301-02-21

Тула 2021 год

**Разработчик(и) методических указаний**

Берсенов Г.Б., доцент каф. ВТ, к.т.н.

(ФИО, должность, ученая степень, ученое звание)

(подпись)

## Содержание

|                                                                                                 | стр. |
|-------------------------------------------------------------------------------------------------|------|
| Введение . . . . .                                                                              | 3    |
| 1. Цель и задачи выполнения курсовой работы . . . . .                                           | 4    |
| 2. Основные требования к курсовой работе . . . . .                                              | 5    |
| 2.1. Тематика курсовой работы . . . . .                                                         | 5    |
| 2.2. Исходные данные к курсовой работе . . . . .                                                | 11   |
| 2.3. Задание на курсовую работу . . . . .                                                       | 16   |
| 2.4. Объем курсовой работы . . . . .                                                            | 16   |
| 2.5. Защита курсовой работы . . . . .                                                           | 16   |
| 3. Методические указания к работе над курсовой работой . . . . .                                | 17   |
| 3.1. Основные этапы проектирования . . . . .                                                    | 17   |
| 3.2. Методические указания к выполнению отдельных<br>этапов проектирования . . . . .            | 17   |
| 3.3. Содержание разделов пояснительной<br>записки (текстовой части) к курсовой работе . . . . . | 18   |
| Список литературы . . . . .                                                                     | 18   |

## Введение

Первые вычислительные машины (на лампах) не имели операционных систем, а программы для них составлялись на машинных языках. В полупроводниковых электронных вычислительных машинах (ЭВМ) появились ассемблеры, макроязыки, библиотеки подпрограмм, а затем и языки высокого уровня. Операторы ЭВМ вручную запускали сначала трансляцию программы, а затем, посредством специальных программ–загрузчиков, загружали в оперативную память исполнимые коды программ и выполняли их.

Для автоматизации работы операторов и уменьшения простоев ЭВМ появились программы–мониторы для организации пакетной обработки программ (заданий). С увеличением быстродействия процессора появились системы мультипрограммирования – сначала для пакетной обработки, а затем для систем разделения времени (многотерминальных систем). Этот комплекс системных программ получил название операционной системы ЭВМ.

В рамках операционных систем были созданы файловая система, виртуальная память и аппаратная поддержка для программ операционной системы (система прерываний, защита памяти, привилегированный режим работы и др.). Появились сети ЭВМ, обеспечивающие удаленный доступ и удаленное выполнение заданий. И в этот период развития ЭВМ (в конце шестидесятых) была разработана операционная система Unix.

Операционная система UNIX за 50 лет своего существования подтвердила правильность основных концепций, заложенных изначально в нее разработчиками. Она многократно модернизировалась и дополнялась, в результате чего

в середине восьмидесятых появились две базовые версии системы UNIX: System V Release 4 (SVR4) и 4BSD. На их основе затем было создано множество новых версий: FreeBSD, AIX, HP-UX, IRIX, Digital UNIX, SCO UNIX, Solaris, Minix, Linux и др. UNIX явилась первой действительно переносимой операционной системой, спецификация и реализация наиболее важных компонентов которой регламентируется стандартами Posix, XPG, System V, UNIX98 и др. Система UNIX послужила основой (прототипом) при создании операционных систем MS DOS и MS Windows. UNIX традиционно использовалась для изучения операционных систем в университетах, поскольку предоставляла исходные коды и свободно распространяемые версии своей системы. И сейчас такие версии системы UNIX, как, например, FreeBSD и Linux, предоставляют возможность свободного доступа к исходным кодам операционной системы.

## **1. Цель и задачи выполнения курсовой работы**

Курсовая работа «Разработка оболочки для ядра операционной системы» выполняется для закрепления знаний по курсу "Операционные системы семейства Unix" и приобретения навыков создания командного интерфейса (оболочки) ядра операционной системы посредством программной реализации интерпретатора для внутренних команд оболочки и заданного набора внешних команд, использующих утилиты операционной системы.

Курсовая работа является продолжением бакалаврского цикла дисциплин: "Информатика", "Программирование", "Технологии программирования", "Операционные системы и среды".

Задачами курсовой работы являются:

- ознакомление с современными ядрами и дистрибутивами операционной системы Linux;
- получение опыта использования эмулятора виртуальных машин VirtualBox для установки и эксплуатации конкретного дистрибутива операционной системы Linux;
- приобретение навыков разработки и отладки на языке GNU C99 многомодульных (многопоточковых и многопроцессных) программ типа клиент-сервер с использованием системных вызовов и библиотек стандартных подпрограмм;
- практическое овладение методами и средствами создания объектов IPC для взаимодействия потоков и процессов;
- практическое овладение методикой создания интерпретатора команд операционной системы;
- приобретение навыков оформления и выпуска документации в соответствии с ГОСТ.

## **2. Основные требования к курсовой работе**

### **2.1. Тематика курсовой работы**

#### **2.1.1. Ядро системы Linux**

Операционная система Linux имеет двухуровневую структуру, состоящую из ядра на нижнем уровне и приложений на верхнем уровне. Ядро (kernel) непосредственно взаимодействует с аппаратной частью компьютера, изолируя прикладные программы от особенностей ее архитектуры. Ядро имеет набор услуг, предоставляемых прикладным программам. К услугам ядра относятся операции ввода/вывода (открытия, чтения, записи и управления файлами), создания и управления потоками и процессами, их синхронизации и межпроцессного взаимодействия. Все приложения запрашивают услуги ядра посредством системных вызовов.

Приложения или задачи подразделяются на системные, определяющие функциональность операционной системы (регистрация пользователей, конфигурирование устройств, управление работой устройств, реализация консольного и графического интерфейса, интерпретация команд (скриптов) и компиляция программ), и прикладные, обеспечивающие пользовательский интерфейс системы Linux (например, текстовые редакторы и утилиты). Несмотря на внешнюю разнородность этих приложений, схемы их взаимодействия с ядром одинаковы. Важное место в этом взаимодействии занимает оболочка ядра, содержащая интерпретатор команд, реализующий командный интерфейс операционной системы.

Ядро ОС Linux монолитное, т.е. оно выполняется в одном адресном пространстве в режиме ядра. Тем не менее ядро Linux позаимствовало некоторые хорошие свойства микроядерной модели: в нем используется преemptивное (т.е. с вытеснением выполняющихся процессов более приоритетными) ядро, поддерживаются потоки пространства ядра и возможность динамической загрузки в ядро внешних бинарных файлов (модулей ядра). Ядро Linux не использует никаких функций микроядерной модели, которые приводят к снижению производительности: все выполняется в режиме ядра с непосредственным вызовом функций вместо передачи сообщений. Следовательно, операционная система Linux — модульная, многопоточная, а выполнение самого ядра можно планировать.

#### **2.1.2. Дистрибутивы системы Linux**

Сначала версии Linux распространялись на обыкновенных дискетах - одна содержала ядро, а другая – корневую файловую систему и необходимые программы. Потом появились дистрибутивы, содержащие кроме этого программу установки операционной системы на компьютер. Компания Red Hat стала одной из первых, выпустивших свой дистрибутив системы Linux на лазерном компакт-диске. Отметим, что в настоящее время:

- основными дистрибутивами для настольных операционных систем являются Red Hat, Slackware и Debian, а все остальные – лишь производные от них. Так, Mandrake и ASPLinux произошли от Red Hat, дистрибутив ALT

Linux взял за основу Mandrake, а Ubuntu основан на Debian. Потом на смену Red Hat пришел дистрибутив Fedora Core (сейчас просто Fedora), а на смену Mandrake – сначала Mandriva, а затем Mageia. Существует два типа версий Ubuntu: обычные и LTS (Long Term Support). Обычные версии выходят два раза в год, а LTS – только один раз. Однако техническая поддержка и обновление программ для LTS-дистрибутивов доступны на протяжении 5 лет;

- номер версии дистрибутива не совпадает с номером версии ядра – это принципиально разные вещи;
- самыми популярными дистрибутивами на сегодняшний день считаются Ubuntu и Fedora – для настольного применения (для дома и офиса), а также CentOS и Debian – для серверного. Также широко используется немецкий дистрибутив openSUSE, существующий в двух вариантах: Tumbleweed и Leap. Первый содержит все самое новое ПО и постоянно обновляется, а второй – только стабильное ПО;
- основными графическими интерфейсами ОС Linux являются GNOME и KDE. По умолчанию GNOME используется в Fedora, Ubuntu, Debian, а KDE – в openSUSE. Однако это можно изменить. Например, в Ubuntu для установки KDE следует выполнить в терминале следующую команду:

```
sudo apt-get install kubuntu-desktop
```

Эта команда установит интерфейс KDE со стандартным набором приложений. Чтобы получить KDE с полным набором приложений, установите пакет `kubuntu-full`. В Fedora можно установить графические интерфейсы Cinnamon, MATE, XFCF, LXDE. Все эти интерфейсы отличаются ограниченной функциональностью и подходят для очень слабых компьютеров. Более или менее удобным из них считается LXDE.

GNU/Linux – мультинациональная ОС. И каждая страна создает свои дистрибутивы, которые используются как на рабочих станциях, так и на серверах. Россия не отстает, она создала несколько хороших (и не очень) дистрибутивов Linux.

### 2.1.3. Установка системы Linux

Система Linux устанавливается либо в раздел файловой системы, либо в контейнер виртуальной машины. В первом случае Linux нужно устанавливать после установки операционной системы (ОС) Windows, поскольку загрузчик Linux без проблем загружает все имеющиеся версии Windows, а вот заставить загрузчик Windows загружать Linux очень сложно.

При установке Linux из настольного дистрибутива необходимо зарезервировать следующую дисковую память:

- для корневого раздела (/), где будут находиться файлы операционной системы и приложения, примерно 8 – 10 Гбайт;
- для раздела с пользовательскими файлами (/home) память выделяется в соответствии с предполагаемыми объемами обрабатываемых данных;
- под область свопинга (swap) выделяется примерно 1 – 2 Гбайт.

В процессе установки пользователь вводит запрашиваемые данные. Поскольку современные компьютеры имеют достаточную оперативную память (несколько Гбайт) и высокую производительность, установка системы Linux производится обычно в контейнер виртуальной машины. Предполагается использование эмулятора виртуальной машины VirtualBox, которая является бесплатной и распространяется по лицензии GPL.

При создании новой виртуальной машины необходимо запустить менеджер виртуальных машин и нажать кнопку **Создать**. Отметим, что создаваемая виртуальная машина (ВМ) будет работать под управлением гостевой операционной системы (системы Linux), а основная операционная система (например, Windows), внутри которой создается ВМ, называется хостом. При создании ВМ необходимо выделить оперативную память для гостевой ОС (рекомендуется отдать примерно половину объема оперативной памяти хоста). Следующий шаг – это выбор файла образа жесткого диска. Выбираем опцию **Создать новый виртуальный жесткий диск**, а затем выбираем формат виртуального диска. Обычно используется формат VDI (VirtualBox Disk Image). Далее задается имя жесткого диска ВМ и метод резервирования места на диске хоста. Можно выбрать опцию **Динамический виртуальный жесткий диск** – в этом случае размер файла, эмулирующего жесткий диск ВМ, будет увеличиваться по мере необходимости, но не превысит заданного ранее размера.

После создания ВМ необходимо нажать кнопку **Настроить** для корректировки параметров ВМ. Появится список разделов для возможной настройки. Например, раздел **Носители** позволяет изменить образы загружаемых ОС. Поскольку по умолчанию предполагается использование образа ОС на CD/DVD, а обычно установочный образ диска гостевой ОС мы загружаем из Интернета и записываем в виде ISO-файла на винчестер, то необходимо выполнить следующее. Щелкаем по диску **Пусто** в дереве носителей и нажмем кнопку выбора диска, а из открывшегося меню выбираем команду **Выбрать образ оптического диска**. Откроется диалоговое окно выбора файлов, в котором нужно выбрать записанный на винчестер образ диска. В разделе **Сеть** Вы можете определить, как гостевая ОС будет взаимодействовать по сети с хостом.

#### 2.1.4. Оболочка ядра системы Linux

Несмотря на то, что Linux имеет графический интерфейс пользователя, большинство программистов и продвинутые пользователи по-прежнему предпочитают интерфейс командной строки, называемый оболочкой (shell). Они часто запускают одно или несколько окон с оболочками и работают в них. Интерфейс командной строки, реализуемый оболочкой, значительно быстрее и эффективнее в использовании (особенно при использовании профессионалами различных комбинаций клавиш, называемых горячими клавишами или акселераторами), проще расширяется и не раздражает пользователя необходимостью постоянно хвататься за мышь.

В операционной системе Linux имеется несколько оболочек, но по умолчанию обычно используется оболочка *bash*. Когда оболочка запускается,

она инициализируется, затем выводит на экран промптер – символ или строку приглашения к вводу. Пользователь может изменить промптер, например, сделать так, чтобы он содержал имя текущего каталога и автоматически изменялся при смене каталога. После того, как пользователь введет командную строку, оболочка (или ее потомок, которого она обычно создает для выполнения команды) извлекает первое слово командной строки и ищет подпрограмму в оболочке (если команда относится к внутренним командам оболочки) либо готовую программу (обычно утилиту, находящуюся в виде файла в текущем каталоге или в одном из системных каталогов).

У команд могут быть аргументы, которые передаются запускаемой программе в виде текстовых строк. Например, командная строка

```
cp src dest
```

запускает программу *cp*, которая делает копию файла *src* и называет эту копию именем *dest*.

Аргументами могут быть флаги (обычно со знаком тире), которые управляют работой команды. Например, команда

```
head -20 file
```

напечатает первые двадцать строк файла *file* (вместо принятых по умолчанию 10 строк), а команда

```
head 20 file
```

даст указание программе *head* вывести первые 10 строк файла 20, а затем первые 10 строк файла *file*. Большинство команд Linux-систем могут принимать несколько флагов и аргументов.

Чтобы упростить указание нескольких файлов в команде, оболочка принимает так называемые групповые символы. Например, символ звездочка означает все возможные текстовые строки, так что строка

```
ls *.c
```

даст указание программе *ls* вывести список всех файлов текущего каталога, имя которых оканчивается на *.c*. Если в каталоге находятся файлы *x.c*, *y.c* и *z.c*, то данная команда будет эквивалентна команде

```
ls x.c y.c z.c
```

Другим групповым символом является вопросительный знак, который заменяет один любой символ. Кроме того, в квадратных скобках можно указать множество символов, из которых программа должна будет выбрать один. Например, команда

```
ls [ame]*
```

выводит список файлов, имя которых начинается с символов «а», «т» или «е».

Оболочка не должна открывать терминал (клавиатуру и монитор), чтобы прочитать с него или сделать на него вывод. Вместо этого запускаемые программы автоматически получают доступ для чтения к файлу (точнее, к устрой-

ству, поскольку в Linux каждое устройство является одним из видов файла), называемому стандартным устройством ввода (`stdin`), а для записи – к файлу, называемому стандартным устройством вывода (`stdout`), и к файлу, называемому стандартным устройством для вывода сообщений об ошибках (`stderr`). По умолчанию всем этим трем устройствам соответствует терминал, т.е. чтение производится с клавиатуры, а запись – на монитор. Многие Linux-программы читают данные со стандартного устройства ввода и пишут на стандартное устройство вывода. Например, команда

```
sort
```

запускает программу-утилиту, находящуюся в файле *sort*, которая читает строки с терминала (пока пользователь не нажмет комбинацию клавиш **CTRL-D**, чтобы обозначить конец файла), а затем сортирует их в алфавитном порядке и выводит результат на экран.

Стандартные ввод и вывод можно перенаправить, что является очень полезным свойством. Для этого используются символы «<» и «>» соответственно. Например, команда

```
sort <in >out
```

заставляет программу *sort* взять в качестве входного файла *in* и направить вывод в файл *out*. Поскольку стандартный вывод сообщений об ошибках не был перенаправлен, то все сообщения об ошибках попадут на экран. Программа, которая считывает данные со стандартного устройства ввода, выполняет определенную обработку этих данных и записывает результат в поток стандартного вывода, называется фильтром.

Рассмотрим следующую командную строку, состоящую из трех отдельных команд:

```
sort <in >temp; head -30 <temp; rm temp
```

Сначала запускается программа *sort*, которая принимает данные из файла *in* и записывает результат в файл *temp*. Когда она завершает свою работу, оболочка запускает программу *head*, дав ей указание вывести первые 30 строк из файла *temp* на стандартное устройство вывода, которым по умолчанию является терминал. По завершению вывода временный файл *temp* удаляется.

Часто используются командные строки, в которых первая программа в командной строке формирует вывод, используемый второй программой в качестве входа. В приведенном выше примере для этого использовался временный файл *temp*. Однако система Linux (как и Unix) предоставляет для этого более простой способ. В командной строке

```
sort <in | head -30
```

используется вертикальная черта, называемая символом канала. Она означает, что вывод программы *sort* должен использоваться в качестве входа для программы *head*, что позволяет обойтись без создания, использования и удаления временного файла.

Набор команд, соединенных символом канала, называется конвейером. Он может содержать произвольное количество команд. Пример четырехкомпонентного конвейера показан в следующей строке:

```
grep ter *.t | sort | head -20 | tail -5 >foo
```

Здесь в стандартное устройство вывода записываются все строки, содержащие строку «ter» во всех файлах, оканчивающихся на *.t*, после чего они сортируются. Первые 20 строк выбираются программой *head*, которая передает их программе *tail*, записывающей последние пять строк (из переданных двадцати) в файл *foo*.

Linux является универсальной многозадачной системой. Один пользователь может одновременно запустить несколько программ, каждую в виде отдельного процесса. Синтаксис оболочки для запуска фонового процесса состоит в использовании амперсанда в конце строки. Таким образом, строка

```
wc -l <file1 >file2 &
```

запустит программу *wc* для подсчета количества символов/слов/строк, которая сосчитает число строк (флаг *-l*) во входном файле *file1* и запишет результат в файл *file2*, но будет делать это в фоновом режиме. Как только команда будет введена пользователем, оболочка выведет символ приглашения к вводу и будет готова к обработке следующей команды.

Конвейеры также могут выполняться в фоновом режиме, например:

```
sort <file1 | head &
```

Можно одновременно запустить в фоновом режиме несколько конвейеров.

Список команд оболочки может быть помещен в файл, а затем можно будет запустить оболочку с этим файлом в качестве стандартного входа. В этом случае оболочка выполнит перечисленные в этом файле команды одну за другой – точно так же, как если бы эти команды вводились с клавиатуры. Файлы, содержащие команды оболочки, называются сценариями оболочки.

Сценарии оболочки могут присваивать значения переменным оболочки и затем считывать их. Они также могут иметь параметры и использовать конструкции *if*, *for*, *while* и *case*. Таким образом, сценарии оболочки представляют собой настоящие программы, написанные на языке оболочки.

### 2.1.5. Взаимодействие процессов в системе Linux

Ядро операционной системы Linux и ее системные приложения обычно имеют сложную многомодульную структуру типа клиент-сервер. Каждый модуль выполняет функции программного клиента и/или сервера и часто реализуется посредством процесса или программного потока (процесса/потока). Эти модули взаимодействуют между собой посредством объектов межпроцессного или межпоточкового взаимодействия, называемых в дальнейшем объектами IPC (IPC – Interprocess Communication).

IPC включает в себя следующие виды объектов:

- объекты для передачи сообщений (каналы, FIFO, очереди сообщений, сокет);

- объекты для синхронизации (взаимные исключения (мьютексы), условные переменные, блокировки чтения-записи, семафоры);
- разделяемая память (неименованная и именованная).

Взаимодействие процессов/потоков по механизмам реализации подразделяется на три основных шаблона (модели) взаимодействия:

1. *Сервер файлов*: клиент посылает серверу запрос с именем файла, а сервер возвращает клиенту его содержимое.
2. *Производитель-потребитель*: один или несколько процессов/потоков (производителей) помещают данные в буфер общего пользования, а другие процессы/потоки (потребители) производят с этими данными различные операции.
3. *Разделяемый индекс* (последовательное увеличение номера): один или несколько процессов/потоков увеличивают общий для всех индекс. Он может храниться в файле с общим доступом или в совместно разделяемой памяти.

Объекты IPC (очереди сообщений, семафоры, разделяемая или общая память) различаются по своей реализации и функциональным возможностям на объекты IPC стандарта System V и объекты IPC стандарта Posix. Интерфейс системных вызовов для каждого из стандартов приведен в табл. 3.1 и 3.2.

Таблица 3.1. Функции System V IPC

|                                   | <b>Очереди сообщений</b> | <b>Семафоры</b> | <b>Общая память</b> |
|-----------------------------------|--------------------------|-----------------|---------------------|
| Заголовочный файл                 | <sys/msg.h>              | <sys/sem.h>     | <sys/shm.h>         |
| Функции для создания или открытия | msgget                   | semget          | shmget              |
| Операции управления               | msgctl                   | semctl          | shmctl              |
| Операции IPC                      | msgsnd<br>msgrcv         | semop           | shmat<br>shmdt      |

Таблица 3.2. Функции Posix IPC

|                                           | <b>Очереди сообщений</b>           | <b>Семафоры</b>                                                | <b>Общая память</b>    |
|-------------------------------------------|------------------------------------|----------------------------------------------------------------|------------------------|
| Заголовочный файл                         | <mqueue.h>                         | <semaphore.h>                                                  | <sys/mman.h>           |
| Функции для создания, открытия и удаления | mq_open<br>mq_close<br>mq_unlink   | sem_open<br>sem_close<br>sem_unlink<br>sem_init<br>sem_destroy | shm_open<br>shm_unlink |
| Операции управления                       | mq_getattr<br>mq_setattr           |                                                                | ftruncate<br>fstat     |
| Операции IPC                              | mq_send<br>mq_receive<br>mq_notify | sem_wait<br>sem_trywait<br>sem_post<br>sem_getvalue            | mmap<br>munmap         |

## 2.2. Исходные данные к курсовой работе

Необходимо создать интерпретатор команд myshell в виде многомодульного приложения типа клиент-сервер, где каждый модуль выполняет функции

программного клиента и/или сервера. Модули взаимодействуют между собой через объекты IPC.

Предлагается следующая обобщенная структура (мета модель) интерпретатора команд, включающая в себя следующие модули-серверы, которые могут быть реализованы как процессы или потоки и могут быть интегрированы с другими модулями-серверами:

- сервер ввода командных строк из потока `stdin`;
- сервер исполнения интерактивной команды (сервер  $s_0$ );
- серверы исполнения неинтерактивных команд (серверы  $s_1, \dots, s_n$ );
- файловый сервер (сервер  $s_f$ );
- сервер истории, накапливающий список команд, полученных интерпретатором для выполнения в течение сеанса его работы (сервер  $s_h$ );
- сервер вывода в поток `stdout` результатов выполнения команд.

В контексте данной архитектуры интерпретатор команд функционирует следующим образом. Сервер ввода считывает очередную командную строку из потока `stdin` и разбивает ее на команды. Далее он производит анализ каждой команды: определяет тип команды - интерактивная или неинтерактивная, выделяет имя команды и ее аргументы и проверяет их правильность. Если в командной строке обнаружены ошибки, в сервер вывода через канал передается сообщение об ошибке. Если ошибок нет, то команды командной строки передаются на сервер  $s_h$  для добавления в список истории работы командного интерпретатора.

Команды, выделенные из командной строки, направляются на серверы исполнения команд: интерактивные команды – через объект `IPC1` на сервер  $s_0$  исполнения интерактивной команды, а неинтерактивные команды - через объект `IPC2` на серверы  $s_1, \dots, s_n$  исполнения неинтерактивных команд. Предполагается, что сервер  $s_0$  создается сразу же после запуска интерпретатора команд, а серверы  $s_1, \dots, s_n$  в зависимости от варианта задания создаются либо сразу же после запуска интерпретатора команд, либо динамически на время выполнения неинтерактивных команд.

Серверы исполнения команд при необходимости обращаются через объект(ы) `IPC3` к серверу файлов  $s_f$  для чтения или записи файлов. Объект(ы) `IPC3` должны обеспечивать двустороннюю связь между серверами  $s_0, s_1, \dots, s_n$  и файловым сервером  $s_f$ .

Результаты выполнения каждой команды через объект `IPC4` передаются на сервер вывода в поток `stdout`. В начале исполнения каждой неинтерактивной команды из серверов  $s_1, \dots, s_n$  на сервер вывода передается для вывода диспетчерский номер задания (неинтерактивного процесса), присвоенный интерпретатором (серверами  $s_1, \dots, s_n$ ), и идентификатор процесса или потока, присвоенный ядром операционной системы. Порядковый номер последнего из полученных заданий хранится в объекте `IPC5`.

При выполнении задания в соответствии с полученным вариантом на основе обобщенной структуры разрабатывается конкретная структура, например, структура, представленная на рис. 1. Здесь показаны способы реализации каж-

дого сервера (посредством процессов с помощью системных вызовов `fork()` и `exec()` либо посредством программных потоков с помощью функции `pthread_create()`) и приведены виды использованных объектов IPC (именованные каналы FIFO и неименованные каналы `pipe`, семафоры SEM).

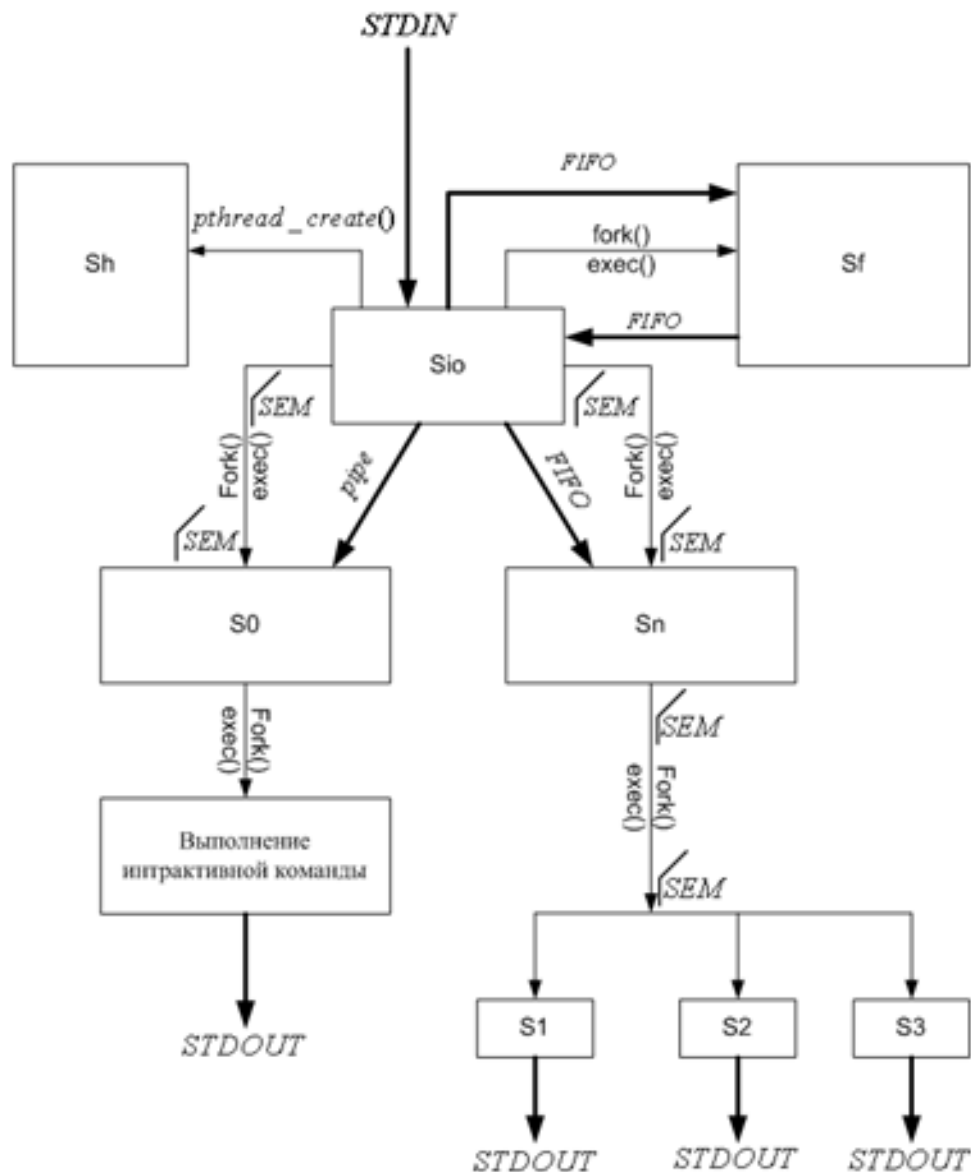


Рис. 1. Пример многомодульной структуры оболочки ядра

Реализовать оболочку ядра в таком виде довольно сложно. Поэтому предлагается интегрировать (объединять) некоторые серверы, уменьшая таким образом число создаваемых процессов/потоков и число используемых объектов IPC. Например, серверы ввода/вывода и сервер истории могут быть реализованы в одном процессе.

На рис. 2 приведен пример выполнения в оболочке `bash` системы `Ubuntu` командной строки, содержащих 5 интерактивных команд и одну фоновую (неинтерактивную) команду (с символом `&` в конце команды). Для записи неинтерактивных команд внутри командной строки обычно применяется синтаксис с использованием круглых скобок, например:

```
(ls >> f2 &) ; ls -l >> f2 &
```

```

user1@user1-VirtualBox:~/labs/lab3$ echo "Begin of commands line"; pwd >> f1 ; cat hello.c >> f1 ; ls; ls -a; ls -l >> f1 &
Begin of commands line
add.c add.cpp add.o a.out clg conf1 conf2 conf3 conf4 conf5 f1 hello.c prog
. .. add.c add.cpp add.o a.out clg conf1 conf2 conf3 conf4 conf5 f1 hello.c prog
[1] 13016
user1@user1-VirtualBox:~/labs/lab3$ cat f1
/home/user1/labs/lab3
// hello.c
#include <stdio.h>
int main ()
{
    printf ("Hello, world!\n");
    return 0;
}
итого 84
-rw-r--r-- 1 user1 user1 147 авг 21 17:45 add.c
-rw-r--r-- 1 user1 user1 154 авг 21 17:58 add.cpp
-rw-rw-r-- 1 user1 user1 3384 авг 21 17:58 add.o
-rwxrwxr-x 1 user1 user1 17504 авг 21 18:09 a.out
-rw-r--r-- 1 user1 user1 951 авг 20 15:03 clg
-rwxr-xr-x 1 user1 user1 73 авг 20 15:03 conf1
-rw-r--r-- 1 user1 user1 147 авг 20 15:04 conf2
-rw-r--r-- 1 user1 user1 133 авг 20 15:04 conf3
-rw-r--r-- 1 user1 user1 129 авг 20 15:04 conf4
-rw-r--r-- 1 user1 user1 176 авг 20 15:04 conf5
-rw-rw-r-- 1 user1 user1 111 сен 20 17:19 f1
-rw-r--r-- 1 user1 user1 89 авг 20 15:04 hello.c
-rwxrwxr-x 1 user1 user1 16688 авг 21 18:12 prog

```

Рис. 2. Пример выполнения командной строки

Если командную строку записать в файл f1 и передать его созданному интерпретатору команд myshell следующим образом:

```
myshell < f1 > f2
```

то это тестовое задание выполнится как скрипт, принимающий команды из f1 и записывающий результаты их выполнения в файл f2.

В таблице приведены типовые варианты заданий для разработки командной оболочки ядра операционной системы Linux.

Таблица 3.3.

Варианты реализации оболочки для ядра операционной системы Linux

| Вариант | Среда для виртуальных машин | Дистрибутив Linux | Среда рабочего стола | Реализация сервера $S_0$ | Реализация сервера $S_f$ | Серверы $S_1, \dots, S_n$ | Реализация перенаправления ввода/вывода | Момент создания серверов $S_1, \dots, S_n$ |
|---------|-----------------------------|-------------------|----------------------|--------------------------|--------------------------|---------------------------|-----------------------------------------|--------------------------------------------|
| 1       |                             |                   |                      | процесс                  | поток                    | $S_1$                     | >                                       | до разбора строки                          |
| 2       |                             |                   |                      | процесс                  | поток                    | $S_1, S_2$                | >>                                      | после разбора строки                       |
| 3       |                             |                   |                      | процесс                  | поток                    | $S_1, S_2, S_3$           | <                                       | до разбора строки                          |
| 4       |                             |                   |                      | процесс                  | процесс                  | $d$                       | >                                       | после разбора строки                       |
| 5       |                             |                   |                      | процесс                  | процесс                  | $S_1$                     | >>                                      | до разбора строки                          |
| 6       |                             |                   |                      | процесс                  | поток                    | $S_1, S_2$                | <                                       | после разбора строки                       |
| 7       |                             |                   |                      | процесс                  | поток                    | $S_1, S_2, S_3$           | >                                       | до разбора строки                          |
| 8       |                             |                   |                      | процесс                  | процесс                  | $d$                       | >>                                      | после разбора строки                       |
| 9       |                             |                   |                      | процесс                  | процесс                  | $S_1$                     | <                                       | до разбора строки                          |
| 10      |                             |                   |                      | процесс                  | процесс                  | $S_1, S_2$                | >                                       | после разбора строки                       |
| 11      |                             |                   |                      | поток                    | поток                    | $S_1, S_2, S_3$           | >>                                      | до разбора строки                          |
| 12      |                             |                   |                      | поток                    | поток                    | $d$                       | <                                       | после разбора строки                       |
| 13      |                             |                   |                      | поток                    | процесс                  | $S_1$                     | >                                       | до разбора строки                          |
| 14      |                             |                   |                      | поток                    | процесс                  | $S_1, S_2$                | >>                                      | после разбора строки                       |
| 15      |                             |                   |                      | поток                    | поток                    | $S_1, S_2, S_3$           | <                                       | до разбора строки                          |
| 16      |                             |                   |                      | поток                    | поток                    | $d$                       | >                                       | после разбора строки                       |
| 17      |                             |                   |                      | поток                    | процесс                  | $S_1$                     | >>                                      | до разбора строки                          |
| 18      |                             |                   |                      | поток                    | процесс                  | $S_1, S_2$                | <                                       | после разбора строки                       |
| 19      |                             |                   |                      | поток                    | процесс                  | $S_1, S_2, S_3$           | >                                       | до разбора строки                          |
| 20      |                             |                   |                      | поток                    | процесс                  | $d$                       | >>                                      | после разбора строки                       |

### **2.3. Задание на курсовую работу**

При выполнении курсовой работы «Разработка оболочки для ядра операционной системы» студент должен разработать интерпретатор команд типа клиент-сервер заданной архитектуры и реализовать набор встроенных команд интерпретатора.

Задание на проектирование выдается студенту в течение первых двух недель семестра. Задание оформляется на типовом бланке отдельно на каждый проект, включая индивидуальные задания. На бланке типового задания указывается тема работы "Разработка оболочки для ядра операционной системы" и исходные данные, определенные по номеру варианта, заданному преподавателем. В течение первых четырех недель с момента выдачи задания исходные данные могут быть откорректированы по согласованию с руководителем работы. Индивидуальное задание должно быть согласовано с ведущим лектором курса "Проектирование операционных систем".

### **2.4. Объем курсовой работы**

Курсовая работа «Разработка оболочки для ядра операционной системы» состоит из пояснительной записки и информационно-программных средств, реализующих задание на проектирование.

Пояснительная записка (ПЗ) оформляется согласно требованиям ЕСПД и должна содержать:

- титульный лист;
- бланк задания;
- введение;
- основное содержание;
- заключение;
- библиографический список;
- приложение, содержащее фрагменты программ и изображения входных и выходных документов, подтверждающие выполнение основных этапов курсовой работы и отражающие ее отличия от других вариантов заданий.

Работающие варианты программ вместе с исходными текстами предоставляются преподавателю в электронном виде.

### **2.5. Защита курсовой работы**

Выполненная и оформленная полностью курсовая работа предоставляется руководителю на проверку, который после проверки пояснительной записки и информационно-программного обеспечения подписывает ее к защите или возвращает студенту на доработку в зависимости от готовности работы.

Защищается курсовая работа перед комиссией из двух-трех преподавателей кафедры.

Защита работы заключается в кратком докладе студента об основных проблемах, возникших при разработке информационно-программного обеспечения, принятых им способах их решения и полученных результатах, и ответа им на вопросы членов комиссии.

Комиссия оценивает защиту как по качеству выполненной работы, так и по уровню знаний студента, проявленных им в процессе защиты.

В случае неудовлетворительной оценки студент получает новое задание и выполняет работу заново.

### **3. Методические указания к работе над курсовой работой**

#### **3.1. Основные этапы проектирования**

Выполнение курсовой работы включает в себя следующие этапы:

Э1. Ознакомление с заданием, создание, настройка и освоение соответствующей (заданию) операционной системы и среды разработки программ на компьютере.

Э2. Разработка сервера ввода командных строк из потока stdin, отладка его работы при разбиении командных строк на команды и анализе этих команд.

Э3. Реализация сервера вывода в поток stdout, отладка совместной работы серверов ввода и вывода.

Э4. Реализация сервера исполнения интерактивной команды и объектов IPC, отладка совместной работы трех серверов.

Э5. Реализация файлового сервера и объекта IPC3, отладка интерпретатора команд при выполнении заданий, содержащих только интерактивные команды.

Э6. Разработка серверов исполнения неинтерактивных команд и объекта IPC2, отладка интерпретатора команд при выполнении заданий, содержащих как интерактивные, так и неинтерактивные команды.

Э7. Разработка сервера истории и встроенных (реализованных в интерпретаторе) команд, комплексная отладка интерпретатора команд.

Э8. Комплексная отладка разработанного командного интерфейса операционной системы.

Э9. Оформление пояснительной записки курсовой работы в соответствии с требованиями ГОСТа.

#### **3.2. Методические указания к выполнению отдельных этапов проектирования**

Методические указания, необходимые для выполнения отдельных этапов проектирования, приведены в методических указаниях по выполнению лабораторных работ.

### 3.3. Содержание разделов пояснительной записки (текстовой части) к курсовой работе

В основное содержание ПЗ рекомендуется включить следующие разделы:

1. *Постановка задачи проектирования.* Приводится описание работы проектируемого интерпретатора и варианта задания.
2. *Операционная среда для разработки оболочки ядра.* Дается детальное описание среды виртуализации для создания виртуальной машины Linux, использованного дистрибутива системы Linux и среды рабочего стола операционной системы.
3. *Структура интерпретатора команд.* Приводятся вопросы проектирования серверов (выбор структур данных, декомпозиция серверов на процессы, потоки, функции) и их взаимодействия посредством объектов IPC, схемы порождения и взаимодействия процессов и потоков интерпретатора, основные фрагменты программ с краткими пояснениями.
4. *Программная реализация интерпретатора команд.* Приводится описание программного продукта в соответствии с требованиями ЕСПД. Должны присутствовать фрагменты текстовых документов:
  - описание программы;
  - руководство программиста;
  - руководство системного программиста (включая вопросы установки и настройки операционной системы);
  - руководство оператора (с примерами тестирования).
5. *Программная реализация встроенных команд.* Приводятся краткое описание программ и использованных структур данных.

Пояснительная записка выполняется на листах белой бумаги формата А4 (210х297 мм) машинописным или рукописным способом в соответствии с требованиями государственных стандартов [8,9].

### СПИСОК ЛИТЕРАТУРЫ

1. ГОСТ 19.701 - 90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
2. ГОСТ 19.101 - 77 ЕСПД. Виды программ и программных документов.
3. ГОСТ 19.106 - 78 ЕСПД. Требования к программным документам, выполненным печатным способом.
4. ГОСТ 19.401 - 78 ЕСПД. Текст программы. Требования к содержанию и оформлению.
5. ГОСТ 19.402 - 78 ЕСПД. Описание программы.
6. ГОСТ 19.404 - 79 ЕСПД. Пояснительная записка. Требования к содержанию и оформлению.
7. ГОСТ 2.105 – 95 «Общие требования к текстовым документам»

8. ГОСТ 34.601-90 «Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания»
9. ГОСТ 34.603-92 «Информационная технология. Виды испытаний автоматизированных систем»
10. ГОСТ 34.201-89\* . «Информационная технология. Комплекс стандартов на автоматизированные системы. Документация на АСУ. Виды, комплектность и обозначения документов при создании автоматизированных систем»
11. ГОСТ 51188-98. «Испытания программных средств на наличие компьютерных вирусов»
12. Компьютерное оформление отчетных документов / Составитель Т.И. Матикашвили, ТулГУ, Тула, 2000. -36 с.
13. Берсенева Г. Б. Системное проектирование программного обеспечения. Модели и методы принятия решений: Учеб. пособие. - Тул.гос.ун-т, Тула, 2001. - 115 с.
14. Современные операционные системы/ Э. С. Таненбаум, Х. Бос. – 4-е изд. – СПб. : Питер, 2017. – 1120 с.
15. Таненбаум Э. С. Современные операционные системы – 3-е изд. – СПб. : Питер, 2012. – 1120 с.
16. Лав Р. Разработка ядра Linux, 2-е изд. – М. : ООО «И.Д. Вильямс», 2006. – 448 с.
17. Колисниченко Д.Н. Linux. От новичка к профессионалу. – 5-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2016. – 608 с.
18. Кетов Д.В. Внутреннее устройство Linux. - СПб.: БХВ-Петербург, 2017. – 320 с.
19. Назаров С.В. Современные операционные системы: учебное пособие / С.В. Назаров, А.И. Широков. – 2-е изд. – М.: Национальный Открытый Университет «ИНТУИТ» : БИНОМ. Лаборатория знаний, 2013. – 367 с.
20. Операционные системы / Д. Бэкон, Т. Харрис. – СПб.: Питер; Киев: Издательская группа BHV, 2004. – 800 с.
21. Гома Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений: Пер. с англ. / Х. Гома. – М.: ДМК Пресс, 2002. -704 с.
22. Кертен Р. Введение в QNX Neutrino 2. Руководство по программированию приложений реального времени в QNX Realtime Platform / Р. Кертен. – СПб.: Петрополис, 2001. – 479 с.
23. Назаров С.В. Операционные системы специализированных информационно-вычислительных комплексов: Теория построения и системного проектирования / С.В. Назаров. - М.: Машиностроение, 1989. - 400 с.
24. Смит У. Эффективные решения: проектирование распределенных систем и систем реального времени / У. Смит. – СПб., "Вильямс", 2003. - 540 с.
25. Таненбаум Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван. Стеен. – СПб.: Питер, 2003. – 877 с.
26. QNX/UNIX: анатомия параллелизма / О. Цилюрик, Е. Горошко. – СПб.: Символ-Плюс, 2006. – 288 с.

27. Unix/Linux: теория и практика программирования. – М.: КУДИЦ-ОБРАЗ, 2004. – 576 с.
28. Стивенс У. UNIX: взаимодействие процессов. – СПб. : Питер, 2002. – 576 с.
29. Митчелл М., Оулдем Д., Самьюэл А. Программирование для Linux. Профессиональный подход. - М.: "Вильямс", 2002. - 288 с.
30. Робачевский, А.М. Операционная система UNIX : учеб.пособие / А.М.Робачевский, С.А.Немнюгин, О.Л.Стефик .— 2-е изд.,перераб.и доп. — СПб. : БХВ-Петербург, 2005 .— 635с.
31. Магда, Ю.С. UNIX для студента / Ю.С.Магда .— СПб. : БХВ-Петербург, 2007 .— 480с.
32. Рочкин, М.Д. Программирование для UNIX / М.Д.Рочкин;пер.с англ.под общ.ред.В.В.Вшивцева .— 2-е изд. — СПб. : БХВ-Петербург, 2005 .— 704с.
33. Галатенко, В.А. Программирование в стандарте POSIX : Курс лекций:Учеб.пособие / В.А.Галатенко; Под ред.В.Б.Бетелина .— М. : Интернет-Университет Информационных Технологий, 2004 .— 560с.
34. Хорвиц, Д. Unix-системы.От проектирования до сопровождения : пер.с англ. / Д.Хорвиц .— М. : DiaSoft, 2004 .— 608с.
35. Реймонд, Э.С. Искусство программирования для Unix / Э.С.Реймонд;пер.с англ.и ред.В.А.Швеца .— М.и др. : Вильямс, 2005 .— 543с.
36. Хейер Крис и др. Внутренний мир UNIX. – К.: ДиаСофт, 1998. – 832 с.
37. Немец Э., Снайдер Г., Сибасс С., Хейн Т. Р. UNIX: руководство системного администратора. Для профессионалов. – СПб.: Питер; К.: Издательская группа BHV, 2002. – 928 с.
38. Дансмур М., Дейвис Г. Операционная система UNIX и программирование на языке Си. – М.: "Радио и связь", 1989. – 192 с.
39. Богатырев А. Хрестоматия по программированию на Си в UNIX. Электронное издание, [www.opennet.ru](http://www.opennet.ru). – 1996 г.
40. Б.Керниган, Д.Ритчи, А.Фьюер. Язык программирования Си. Задачи по языку Си. - М.: Финансы и статистика, 1985.
41. М.Уэйт, С.Прата, Д.Мартин. Язык Си. Руководство для начинающих. - М.: Мир, 1988.
42. М.Болски. Язык программирования Си. Справочник. - М.: Радио и связь, 1988.
43. Л.Хэнкок, М.Кригер. Введение в программирование на языке Си. - М.: Радио и связь, 1986.
44. Р.Берри, Б.Микинз. Язык Си. Введение для программистов. - М.: Финансы и статистика, 1988.
45. М.Беляков, А.Ливеровский, В.Семика, В.Шяудкулис. Инструментальная мобильная операционная система ИНМОС. - М.: Финансы и статистика, 1985.
46. К.Кристиан. Введение в операционную систему UNIX. - М.: Финансы и статистика, 1985.

47. Р.Готье. Руководство по операционной системе UNIX. - М.: Финансы и статистика, 1986.
48. М.Банахан, Э.Раттер. Введение в операционную систему UNIX. - М.: Радио и связь, 1986.
49. С.Баурн. Операционная система UNIX. - М.: Мир, 1986.
50. П.Браун. Введение в операционную систему UNIX. - М.: Мир, 1987.
51. Берсенов Г. Б. Файлы и каталоги в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 6 с.
52. Берсенов Г. Б. Интерпретаторы команд и командные процедуры в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 10 с.
53. Берсенов Г. Б. Доступ к файловой системе через интерфейс системных вызовов. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 13 с.
54. Берсенов Г. Б. Работа с процессами в командной и программной средах операционной системы Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 16 с.
55. Берсенов Г. Б. Создание неименованных каналов и приложений типа клиент-сервер в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 11 с.
56. Берсенов Г. Б. Создание именованных каналов и приложений типа клиент-сервер в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 14 с.
57. Берсенов Г. Б. Создание и синхронизация потоков в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 17 с.
58. Берсенов Г. Б. Семафоры для синхронизации процессов и потоков в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 17 с.
59. Берсенов Г. Б. Взаимодействие процессов и потоков через очереди сообщений стандарта Posix в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 17 с.
60. Берсенов Г. Б. Взаимодействие процессов и потоков через очереди сообщений и разделяемую память стандарта System V в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 17 с.
61. Берсенов Г. Б. Последовательное увеличение индекса потоками и процессами в системе Linux. Методические указания по выполнению лабораторной работы. – Тула, ТулГУ (каф. ВТ), 2017. – 17 с.
62. Блог начинающего линуксоида.\_ Русский Linux\_ обзор Российских дистрибутивов общего назначения – URL: [http://linuxoidblog.blogspot.ru/2015/11/linux\\_20.html](http://linuxoidblog.blogspot.ru/2015/11/linux_20.html) . Дата последнего обращения: 29.08.17.