Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Максимов Алексей Борисович
Должность: директор департамента по образовательной политике
Дата подписания: 06.09.2023 12:40:01
Уникальный программный ключ:
8db180d1a3f02ac9e60521a5672742735c18b1d6

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

УТВЕРЖДАЮ
Декан факультета
«Информационные технологии»
_____/Д.Г.Демидов/
«12» мая 2022

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

«Операционные системы»

Направление подготовки

09.03.025_ «Информационные системы и технологии»

Профиль

«Программное обеспечение игровой компьютерной индустрии»

Квалификация (степень) выпускника
Бакалавр

Форма обучения
Очная

Москва 2022 г.

Программа составлена в соответствии с Федеральным государственным образовательным стандартом высшего образования по направлению подготовки бакалавров **09.03.02 «Информационные системы и технологии»**.

Программу составил:

профессор, к.т.н. _ /А.Ф. Иванько /

Программа утверждена на заседании кафедры **«Информатика и информационные технологии»** «30» августа 2022 г., протокол № 1.

Зав. кафедрой ИиИТ,
к.т.н. _____ /Е.В. Булатников/

Декан ФИТ,
к.т.н. _____ /Д.Г. Демидов/

1. **Цели и задачи дисциплины:**

Цели и задачи дисциплины: формирование у студентов теоретических знаний о современных информационных системах и технологиях, моделях, методах и средствах решения функциональных задач и организации информационных процессов, изучение организационной, функциональной и физической структуры Операционных систем, базовой информационной технологии и базовых информационных процессов, рассмотрение перспектив использования информационных технологий в условиях перехода к информационному обществу.

Частные цели – обучение практическим навыкам работы на персональных компьютерах с использованием современных Операционных систем (ОС) и информационных технологий при компьютерной обработке текстовой, графической и мультимедийной информации и последующему их использованию в издательском деле.

Основной задачей изучения дисциплины является овладение методами:

- изучения организационной, функциональной и физической структуры ОС, базовой информационной технологии и базовых информационных процессов в информационных системах и технологиях;
- анализа развития современных ОС и информационных технологий;
- решения функциональных задач ОС, информационных систем и технологий;
- организация информационных процессов при использовании информационных технологий в издательской деятельности.

2. Место дисциплины в структуре ОП

«Операционные системы» относится к обязательной части Блока 1 «Дисциплины (модули)» учебного плана программы бакалавриата по направлению 09.03.02 «Информационные системы и технологии».

Изучение данной дисциплины базируется на следующих дисциплинах:

– Информатика.

Основные положения дисциплины должны быть использованы в дальнейшем при изучении следующих за ней дисциплин, практик:

- Информационные технологии
- Инструментальные средства информационных систем
- Инфокоммуникационные системы и сети
- Технология кроссплатформенного программирования
- Администрирование информационных систем
- Администрирование компьютерных сетей
- Преддипломная практика
- Государственная итоговая аттестация (выполнение и защита ВКР)

3. Требования к результатам освоения дисциплины

В результате освоения ОП бакалавриата обучающийся должен овладеть следующими результатами обучения по дисциплине «Операционные системы»:

Код компетенции	**Результаты освоения ООП** ***Содержание компетенции****	**Перечень планируемых результатов обучения по дисциплине****
ОПК-7	Способен осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем	***Знать***- методы выбора платформ и инструментальных программно-аппаратных средств для реализации информационных систем; ***уметь***- осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем; ***владеть***- Способностью осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.
ПК-1	Способен разрабатывать требования и проектировать программное обеспечение	***Знать***-- требования и проектировать программное обеспечение; ***уметь***- разрабатывать требования и проектировать программное обеспечение; ***владеть***- Способностью разрабатывать требования и проектировать программное обеспечение.

4. Объем дисциплины и виды учебной работы

Общая трудоемкость дисциплины составляет 4 зачетные единицы.

Форма обучения	**курс**	**семестр**	**Трудоемкость дисциплины в часах**							**Форма итогового контроля**
			Всего час./ зач. ед	**Аудиторных часов**	**Лекции**	**Семинарские (практические) занятия**	**Лабораторные работы**	**Самостоятельная работа**	**Контроль (промежуточная аттестация)**	
Очная	**2**	**4**	**144/4**	**72**	**36**	**-**	**36**	**36**	**36**	**экзамен**

Объем дисциплины и виды учебной работы

Общая трудоемкость дисциплины составляет ____4_____ зачетных единиц.

Вид учебной работы	Всего часов	Семестры			
		4			
Контактная работа (всего)	72	72			
В том числе:	-	-	-	-	-
Лекции	36	36			
Практические занятия (ПЗ)					
Семинары (С)					
Лабораторные работы (ЛР)	36	36			
Самостоятельная работа (всего)	36	36			
В том числе:	-	-	-	-	-
Курсовой проект (работа)					
Расчетно-графические работы					
Реферат	36	36			
Эссе					
Контрольная работа					
Другие виды самостоятельной работы					
Вид промежуточной аттестации (экзамен)	36	36			
Общая трудоемкость час./ зач. ед	144/4	144/ 4			

5. Содержание дисциплины

5.1. Тематический план дисциплины

№	Наименование тем (разделов)	Всего часов	Контактная работа			Самостоя-тельная работа
			Лекции	Лабора-торные работы	Практи-ческие занятия, семинары	
1.	Основные сведения об операционных системах	4	1	1		2
2.	Классификация операционных систем.	10	1	1		8
3	Основные принципы построения ОС.	14	2	2		10

4	Состояния процесса. Виды, классификация и свойства процессов.	14	2	2		10
5	Подсистемы ОС. Средства защиты данных и администрирования. прерываний.	14	2	2		10
6	Программные прерывания. Д	14	2	2		10
7	Средства взаимодействия пользователя с компьютером в среде ОС.	14	2	2		10
8	Классическая архитектура ОС.	8	2	2		4
9	Планирование и диспетчеризация потоков.	8	2	2		4
10	Алгоритмы распределения памяти.	8	2	2		4

5.2. Содержание разделов дисциплины

№ раздела	**Наименование разделов**	**Разделы и их содержание**	**Форма текущего контроля успеваемости**
1	2	3	4

1.	Основные сведения об операционных системах	Основные сведения об операционных системах.Понятие ресурса ВС.	Письменные контрольные работы
2.	Классификация операционных систем.	Мультипрограммные режимы. Многозадачный режим. Классификация операционных систем.	Письменные контрольные работы
3.	Основные принципы построения ОС.	Основные принципы построения ОС. Требования , предъявляемые к современным ОС. Классификация и свойства ресурсов.	Письменные контрольные работы
4.	Состояния процесса. Виды, классификация и свойства процессов.	Состояния процесса. Виды, классификация и свойства процессов. Отношения между процессами. Поток.	Письменные контрольные работы
5.	Подсистемы ОС. Средства защиты данных и администрирования. прерываний.	Подсистемы ОС. Средства защиты данных и администрирования. Интерфейс прикладного программирования. Механизм прерываний.	Письменные контрольные работы
6.	Программные прерывания.	Программные прерывания. Концепция виртуализации. Дисциплины распределения ресурсов.	Письменные контрольные работы
7.	Средства взаимодействия пользователя с компьютером в среде ОС.	Средства взаимодействия пользователя с компьютером в среде ОС. Проблемы распределения ресурсов.	Письменные контрольные работы

8.	Классическая архитектура ОС.	Классическая архитектура ОС. Машинно-зависимые компоненты и переносимость ОС. Архитектура на основе микроядра.	Письменные контрольные работы
9.	Планирование и диспетчеризация потоков.	Планирование и диспетчеризация потоков. Алгоритмы планирования. Диспетчеризация приоритетов прерываний в ОС.	Письменные контрольные работы
10.	Алгоритмы распределения памяти.	Алгоритмы распределения памяти. Основные способы распределения памяти.	Письменные контрольные работы

5. Образовательные технологии.

Методика преподавания дисциплины «Операционные системы» и реализация компетентностного подхода в изложении и восприятии материала предусматривает использование следующих активных и интерактивных форм проведения групповых, индивидуальных, аудиторных занятий в сочетании с внеаудиторной работой с целью формирования и развития профессиональных навыков обучающихся:

- подготовка к выполнению лабораторных работ в лабораториях вуза;
- организация и проведение текущего контроля знаний студентов в форме устного опроса.

Удельный вес занятий, проводимых в интерактивных формах, определен главной целью образовательной программы, особенностью контингента обучающихся и содержанием дисциплины Операционные системы» и в целом по дисциплине составляет 50% аудиторных занятий. Занятия лекционного типа составляют 50% от объема аудиторных занятий.

6. Оценочные средства для текущего контроля успеваемости, промежуточной аттестации по итогам освоения дисциплины и учебно-методическое обеспечение самостоятельной работы студентов.

В процессе обучения используются следующие оценочные формы самостоятельной работы студентов, оценочные средства текущего контроля успеваемости и промежуточных аттестаций:

В четвертом семестре

- подготовка к выполнению лабораторных работ и их защита.

Оценочные средства текущего контроля успеваемости включают контрольные вопросы для контроля освоения обучающимися разделов дисциплины.

Образцы контрольных вопросов и заданий для проведения текущего контроля, экзаменационных билетов, приведены в приложении 2.

6.1. Фонд оценочных средств для проведения промежуточной аттестации обучающихся по дисциплине (модулю).

6.1.1. Перечень компетенций с указанием этапов их формирования в процессе освоения образовательной программы.

В результате освоения дисциплины (модуля) формируются следующие компетенции:

Код компетенции	В результате освоения образовательной программы обучающийся должен обладать
ПК-1	Способен разрабатывать требования и проектировать программное обеспечение
ОПК-7	Способен осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем

В процессе освоения образовательной программы данные компетенции, в том числе их отдельные компоненты, формируются поэтапно в ходе освоения обучающимися дисциплин (модулей), практик в соответствии с учебным планом и календарным графиком учебного процесса.

6.1.2. Описание показателей и критериев оценивания компетенций, формируемых по итогам освоения дисциплины (модуля), описание шкал оценивания.

Показателем оценивания компетенций на различных этапах их формирования является достижение обучающимися планируемых результатов обучения по дисциплине (модулю).

Показатель	Критерии оценивания			
	2	3	4	5
ПК-1_ Способен разрабатывать требования и проектировать программное обеспечение				

Знать: требования и проектировать программное обеспечение.	Обучающийся демонстрирует полное отсутствие или недостаточное соответствие следующих знаний: требования и проектировать программное обеспечение	Обучающийся демонстрирует неполное соответствие следующих знаний: разрабатывать требования и проектировать программное обеспечение. Допускаются значительные ошибки, проявляется недостаточность знаний, по ряду показателей, обучающийся испытывает значительные затруднения при оперировании знаниями при их переносе на новые ситуации.	Обучающийся демонстрирует частичное соответствие следующих знаний: разрабатывать требования и проектировать программное обеспечение, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях.	Обучающийся демонстрирует полное соответствие следующих знаний: разрабатывать требования и проектировать программное обеспечение, свободно оперирует приобретенными знаниями.
Уметь: разрабатывать требования и проектировать программное обеспечение.	Обучающийся не умеет или в недостаточной степени умеет разрабатывать требования и проектировать программное обеспечение	Обучающийся демонстрирует неполное соответствие следующих умений: разрабатывать требования и проектировать программное обеспечение. Допускаются значительные ошибки, проявляется недостаточность умений, по ряду показателей, обучающийся испытывает значительные затруднения при оперировании умениями при их переносе на новые ситуации.	Обучающийся демонстрирует частичное соответствие следующих умений: разрабатывать требования и проектировать программное обеспечение. Умения освоены, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях, переносе умений на новые, нестандартные ситуации.	Обучающийся демонстрирует полное соответствие следующих умений: разрабатывать требования и проектировать программное обеспечение. Свободно оперирует приобретенными умениями, применяет их в ситуациях повышенной сложности.

Владеть: Способностью разрабатывать требования и проектировать программное обеспечение	Обучающийся не владеет или в недостаточной степени владеет Способностью разрабатывать требования и проектировать программное обеспечение.	Обучающийся владеет Способностью разрабатывать требования и проектировать программное обеспечение. Допускаются значительные ошибки, проявляется недостаточность владения навыками по ряду показателей. Обучающийся испытывает значительные затруднения при применении навыков в новых ситуациях.	Обучающийся частично владеет Способностью разрабатывать требования и проектировать программное обеспечение. Навыки освоены, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях, переносе умений на новые, нестандартные ситуации.	Обучающийся в полном объеме владеет Способностью разрабатывать требования и проектировать программное обеспечение. Свободно применяет полученные навыки в ситуациях повышенной сложности.
ОПК-7_ Способен осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем				
Знать: как осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.	Обучающийся демонстрирует полное отсутствие или недостаточное соответствие следующих знаний: как осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.	Обучающийся демонстрирует неполное соответствие следующих знаний: как осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем. Допускаются значительные ошибки, проявляется недостаточность знаний, по ряду показателей, обучающийся испытывает значительные затруднения при оперировании знаниями при их переносе на новые ситуации.	Обучающийся демонстрирует частичное соответствие следующих знаний: как осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях.	Обучающийся демонстрирует полное соответствие следующих знаний: как осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем, свободно оперирует приобретенными знаниями.

Уметь: осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.	Обучающийся не умеет или в недостаточной степени умеет осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем	Обучающийся демонстрирует неполное соответствие следующих умений: осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем. Допускаются значительные ошибки, проявляется недостаточность умений, по ряду показателей, обучающийся испытывает значительные затруднения при оперировании умениями при их переносе на новые ситуации.	Обучающийся демонстрирует частичное соответствие следующих умений: Способен осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем. Умения освоены, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях, переносе умений на новые, нестандартные ситуации.	Обучающийся демонстрирует полное соответствие следующих умений: Способен осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем. Свободно оперирует приобретенными умениями, применяет их в ситуациях повышенной сложности.
Владеть: Способностью осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.	Обучающийся не владеет или в недостаточной степени владеет Способностью осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем..	Обучающийся владеет Способностью осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.. Допускаются значительные ошибки, проявляется недостаточность владения навыками по ряду показателей. Обучающийся испытывает значительные затруднения при применении навыков в новых ситуациях.	Обучающийся частично владеет Способностью осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.. Навыки освоены, но допускаются незначительные ошибки, неточности, затруднения при аналитических операциях, переносе умений на новые, нестандартные	Обучающийся в полном объеме владеет Способностью осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем.. Свободно применяет полученные навыки в ситуациях

			ситуации.	повышенной сложности.

Шкалы оценивания результатов промежуточной аттестации и их описание:

Форма промежуточной аттестации: зачёт.

Промежуточная аттестация обучающихся в форме зачёта проводится по результатам выполнения всех видов учебной работы, предусмотренных учебным планом по данной дисциплине (модулю), при этом учитываются результаты текущего контроля успеваемости в течение семестра. Оценка степени достижения обучающимися планируемых результатов обучения по дисциплине (модулю) проводится преподавателем, ведущим занятия по дисциплине (модулю) методом экспертной оценки. По итогам промежуточной аттестации по дисциплине (модулю) выставляется оценка «зачтено» или «не зачтено».

К промежуточной аттестации допускаются только студенты, выполнившие все виды учебной работы, предусмотренные рабочей программой по дисциплине «Операционные системы».

Шкала оценивания	**Описание**
Зачтено	Выполнены все виды учебной работы, предусмотренные учебным планом. Студент демонстрирует соответствие знаний, умений, навыков приведенных в таблицах показателей, оперирует приобретенными знаниями, умениями, навыками, применяет их в ситуациях различной сложности. При этом могут быть допущены незначительные ошибки, неточности, затруднения при аналитических операциях, переносе знаний и умений на новые, нестандартные ситуации.
Не зачтено	Не выполнен один или более видов учебной работы, предусмотренных учебным планом. Студент демонстрирует неполное соответствие знаний, умений, навыков приведенных в таблицах показателей, допускаются значительные ошибки, проявляется отсутствие знаний, умений, навыков по ряду показателей, студент испытывает значительные затруднения при оперировании знаниями и умениями при их переносе на новые ситуации.

Форма промежуточной аттестации: экзамен.

Промежуточная аттестация обучающихся в форме экзамена проводится по результатам выполнения всех видов учебной работы, предусмотренных учебным планом по данной дисциплине (модулю), при этом учитываются

результаты текущего контроля успеваемости в течение семестра. Оценка степени достижения обучающимися планируемых результатов обучения по дисциплине (модулю) проводится преподавателем, ведущим занятия по дисциплине (модулю) методом экспертной оценки. По итогам промежуточной аттестации по дисциплине (модулю) выставляется оценка «отлично», «хорошо», «удовлетворительно» или «неудовлетворительно».

К промежуточной аттестации допускаются только студенты, выполнившие все виды учебной работы, предусмотренные рабочей программой по дисциплине «Операционные системы».

Шкала оценивания	**Описание**
Отлично	Выполнены все виды учебной работы, предусмотренные учебным планом. Обучающийся демонстрирует полное соответствие знаний, умений, навыков приведенным в таблицах показателей. Свободно оперирует приобретенными умениями, применяет их в ситуациях повышенной сложности.
Хорошо	Выполнены все виды учебной работы, предусмотренные учебным планом. Обучающийся демонстрирует частичное соответствие знаний, умений, навыков приведенным в таблицах показателей, оперирует приобретенными знаниями, умениями, навыками, применяет их в ситуациях повышенной сложности. При этом могут быть допущены незначительные ошибки, неточности, затруднения при аналитических операциях, переносе знаний и умений на новые, нестандартные ситуации.
Удовлетворительно	Выполнены все виды учебной работы, предусмотренные учебным планом. Обучающийся демонстрирует неполное соответствие знаний, умений, навыков приведенным в таблицах показателей. Допускаются значительные ошибки, проявляется недостаточность знаний, умений, навыков, по ряду показателей, обучающийся испытывает значительные затруднения при

	оперировании знаниями, умениями, навыками при их переносе на новые ситуации.
Неудовлетворительно	Не выполнен один или более видов учебной работы, предусмотренных учебным планом. Обучающийся не владеет или в недостаточной степени освоил знания, умения, навыки, приведённые в таблицах показателей.

Фонды оценочных средств представлены в приложении 2 к рабочей программе.

7. Учебно-методическое и информационное обеспечение дисциплины.

а) основная литература:

1.Иванько А.Ф., Иванько М.А. Операционные системы: лабораторный практикум / А.Ф. Иванько, М.А. Иванько ; Моск. гос ун-т печати имени Ивана Федорова — М. : МГУП имени Ивана Федорова, 2016. — 218 с. [Электронный ресурс] URL: http://elib.mgup.ru/showBook.php?id=254 .
2.Назаров С. В., Широков А. И. Современные операционные системы: учебное пособие — Интернет-Университет Информационных Технологий, 2011 г. — 280 с. [Электронный ресурс] URL: http://biblioclub.ru/index.php?page=book_red&id=233197&sr=1 .
3.Сафонов В. О. Основы современных операционных систем: учебное пособие — Интернет-Университет Информационных Технологий 2011 г. — 584 с. [Электронный ресурс] URL: http://biblioclub.ru/index.php?page=book_red&id=233210&sr=1 .

б) дополнительная литература:

Курячий Г. В. Операционная система UNIX: методические рекомендации — Интернет-Университет Информационных Технологий 2004 г. — 288 с. [Электронный ресурс] URL: http://biblioclub.ru/index.php?page=book_red&id=233108&sr=1 .

в) программное обеспечение и интернет-ресурсы:

Для успешного освоения дисциплины, студент использует следующие программные средства:

- Microsoft Windows 7 (по программе бесплатного доступа Microsoft Imagine);
- Windows Server 2012 (по программе бесплатного доступа Microsoft Imagine);
- Ubuntu (свободное ПО GNU GPL);
- FreeBSD (свободное ПО BSD);
- VirtualBox (свободное ПО GNU GPL 2).

8. Материально-техническое обеспечение дисциплины.

- для проведения лекционных занятий используются компьютер и проектор для использования лекционного материала в форме презентационных слайдов,
- Компьютерный класс № 2 (ауд. 2554), г. Москва, ул. Прянишникова, д. 2а;

- Компьютерный класс № 2557, г. Москва, ул. Прянишникова, д. 2а;

-Столы, стулья, аудиторная доска, использование переносного мультимедийного комплекса (переносной проектор, персональный ноутбук). Персональные компьютеры, мониторы, мышки, клавиатуры. Рабочее место преподавателя: стол, стул.

9. Методические рекомендации для самостоятельной работы студентов.

Посещение лекционных занятий является обязательным. Пропуск лекционных занятий без уважительных причин и согласования с руководством в объеме более 40% от общего количества предусмотренных учебным планом на семестр лекций влечет за собой невозможность аттестации по дисциплине.

Допускается конспектирование лекционного материала письменным или компьютерным способом.

Регулярная проработка материала лекций по каждому разделу в рамках подготовки к промежуточным и итоговым формам аттестации, а также выполнение и подготовка к защите лабораторных работ по дисциплине является одним из важнейших видов самостоятельной работы обучающегося в течение семестра.

10. Методические рекомендации для преподавателя.

Изучение дисциплины «Операционные системы» обучающимися направления подготовки бакалавров 09.03.02 предусмотрено рабочим учебным планом во 2-ом семестре четвёртого года обучения.

Лекционные занятия проводятся в соответствии с содержанием настоящей рабочей программы.

Лабораторные работы по дисциплине Операционные системы» осуществляется в форме самостоятельной проработки теоретического материала обучающимися; выполнения практического задания; защиты преподавателю лабораторной работы (знание теоретического материала и выполнение практического задания).

При проведении контрольной точки обучающиеся не менее чем за неделю информируются об этом и им выдается список вопросов для подготовки к контрольной работе.

Приложение 1.

Структура и содержание дисциплины «Операционные системы»» по направлению подготовки 09.03.02 «Информационные системы и технологии»
(бакалавр)

n/n	Раздел	Семестр	Неделя семестра	Виды учебной работы, включая самостоятельную работу студентов, и трудоемкость в часах					Виды самостоятельной работы студентов					Формы аттестации	
				Л	П/С	Лаб	СРС	КСР	К.Р.	К.П.	РГР	Реферат	К/р	Э	З
1	**Четвертый семестр**														
2	Основные сведения об операционных системах	4	1	4			4					4			
3	Лабораторная работа № 1.	4	1-2			4									
4	Классификация операционных систем.	4	3	6			6					6			
5	Лабораторная работа № 2.	4	3			6									
6	Основные принципы построения ОС.	4		6			6					6			
7	Лабораторная работа № 3.	4	4			6									
8	Состояния процесса. Виды, классификация и свойства процессов.	4	4	4			4					4			
9	Лабораторная работа № 4.	4	5			4									
10	Подсистемы ОС. Средства защиты данных и администрирования. прерываний.	4	**5**	2			2					2			
11	Лабораторная работа № 4	4	**5**			2									
12	Программные прерывания.	4	6	2			2					2			

13	Лабораторная работа № 5.	4	6			2									
14	Средства взаимодействия пользователя с компьютером в среде ОС.	4	7	2			2					2			
15	Лабораторная работа № 5.	4				2									
16	Классическая архитектура ОС.	4	8	2			2					2			
17	Лабораторная работа № 6..	4				2									
18	Планирование и диспетчеризация потоков.	4	9	4			4					4			
19	Лабораторная работа № 7.	4	10			4									
20	Алгоритмы распределения памяти.	4	10-14	4			4					4			
21	Лабораторная работа № 8.	4	15-16			4									
	Контрольная работа	4	**17**											**Э**	
	Всего часов			36		36	36							**36**	

Приложение 2 к
рабочей программе

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)

Направление подготовки: 09.03.02 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

ОП (профиль): « Программное обеспечение игровой компьютерной индустрии»

Форма обучения: очная

Вид профессиональной деятельности: научно-исследовательская, производственно-технологическая, проектная

Кафедра: Информатика и информационные технологии

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

ПО ДИСЦИПЛИНЕ

«Операционные системы»

Составители:

Профессор Иванько А.Ф.

Москва, 2022 год

Таблица 1

ПОКАЗАТЕЛЬ УРОВНЯ СФОРМИРОВАННОСТИ КОМПЕТЕНЦИЙ

ОПЕРАЦИОННЫЕ СИСТЕМЫ					
ФГОС ВО **09.03.02 «Информационные системы и технологии»**					
В процессе освоения данной дисциплины студент формирует и демонстрирует следующие **профессиональные компетенции**:					
КОМПЕТЕНЦИИ		**Перечень компонентов**	**Технология формирования компетенций**	**Форма оценочного средства****	**Степени уровней освоения компетенций**
ИН-ДЕКС	**ФОРМУЛИРОВ-КА**				
ПК-1	Способен разрабатывать требования и проектировать программное обеспечение	**Знать: как** разрабатывать требования и проектировать программное обеспечение. **Уметь:** разрабатывать требования и проектировать программное обеспечение. **Владеть:** Способностью разрабатывать требования и проектировать программное обеспечение.	лекция, лабораторная работа, самостоятельная работа	К, УО, защита лабораторных работ, экзамен	**Базовый уровень** - воспроизводство полученных знаний в ходе текущего контроля **Повышенный уровень** - практическое применение полученных знаний в процессе подготовки, выполнения и защиты лабораторных работ - свободное использование приобретенных знаний, навыков, умений, применение их в ситуациях повышенной сложности

ОПК-7	Способен осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем	**Знать:** методы осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем. **Уметь:** осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем. **Владеть:** методами как осуществлять выбор платформ и инструментальных программно-аппаратных средств для реализации информационных систем	лекция, лабораторная работа, самостоятельная работа	К, УО, защита лабораторных работ, экзамен	**Базовый уровень** - воспроизводство полученных знаний в ходе текущего контроля **Повышенный уровень** - практическое применение полученных знаний в процессе подготовки, выполнения и защиты лабораторных работ - свободное использование приобретенных знаний, навыков, умений, применение их в ситуациях повышенной сложности

**- Сокращения форм оценочных средств см. в приложении 2 к РП.

Перечень оценочных средств по дисциплине «ПРОГРАММИРОВАНИЕ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ»

№ ОС	Наименование оценочного средства	Краткая характеристика оценочного средства	Представление оценочного средства в ФОС
1	Коллоквиум (К)	Средство контроля усвоения учебного материала темы, раздела или разделов дисциплины, организованное как учебное занятие в виде собеседования педагогического работника с обучающимися.	Вопросы по темам/разделам дисциплины
2	Устный опрос собеседование, (УО)	Средство контроля, организованное как специальная беседа педагогического работника с обучающимся на темы, связанные с изучаемой дисциплиной, и рассчитанное на выяснение объема знаний обучающегося по определенному разделу, теме, проблеме и т.п.	Вопросы по темам/разделам дисциплины

Тематика заданий текущего контроля

Контрольная работа №1:

1. Общая характеристика и классификация операционных систем. Категориальные понятия системного подхода. Формальные методы описания структуры системы. Понятие архитектуры
2. Категориальные понятия системного подхода.
3. Формальные методы описания структуры системы. Понятие архитектуры.
4. Модели функционирования операционных систем.
5. Технологии разработки операционных систем.
6. Особенности реализации операционных систем в различных предметных областях.

Контрольная работа №2:

1. Модели функционирования операционных систем.
2. Технологии разработки операционных систем.
3. Особенности реализации операционных систем в различных предметных областях.
4. Модели и структуры операционных систем.
5. Информационные ресурсы. Теоретические основы современных операционных систем.
6. Базовая эталонная модель Международной организации стандартов.
7. Компоненты операционных систем

Контрольная работа №3

1. Архитектура операционных систем в научных исследованиях.
2. Научные исследования, испытания и эксперименты как объект автоматизации.
3. Функциональные задачи автоматизированных систем научных исследований (АСНИ).
4. Классификация АСНИ, обеспечения АСНИ, функциональная и системная архитектуры.
5. Эталонные аппаратные платформы.
6. Типовые архитектурно-структурные решения, используемые при создании операционных систем.
7. Программное обеспечение операционных систем.

Вопросы для экзамена

Вопросы для оценки качества освоения дисциплины

3. Дайте определение ОС.
4. Каковы особенности алгоритмов управления ресурсами ЭВМ?
5. Назовите критерии эффективности ОС.
6. Перечислите этапы развития ОС компьютеров. Охарактеризуйте каждый из них.
7. Что такое привилегированный программный модуль? Почему нельзя создать мультипрограммную ОС, в которой бы не было привилегированных программных модулей?
8. Какие основные функции выполняют современные ОС?

9. Дайте определение мультипрограммирования.
10. Каковы возможности мультипрограммирования?
11. Перечислите особенности ОС, поддерживающих мультипрограммирование?

12. На что подразделяются ОС по числу одновременно выполняемых задач?
13. Какие средства управления включают в себя многозадачные и однозадачные ОС?
14. Каковы основные различия между вытесняющими и не вытесняющими алгоритмами многозадачности?
15. Дайте определение кластера? Какие требования предъявляются к ОС кластеров?
16. На какие типы в соответствии с использованными при их разработке критериями эффективности подразделяются многозадачные ОС? Дайте краткую характеристику каждого типа.
17. Какими бывают ОС по режиму обработки задач, по организации работы с вычислительной системой и по основному архитектурному принципу?

18. Перечислите основные принципы построения ОС и дайте их краткую характеристику.
19. Перечислите требования, предъявляемые к современным ОС.
20. Каковы тенденции развития ОС на современном этапе?
21. Дайте определение понятию ресурс?
22. Как классифицируются ресурсы?

23. Дайте определение понятий “вычислительный процесс” и “последовательный процесс”.
24. Сколько состояний имеет процесс, дайте их краткую характеристику?
25. Назовите виды процессов и охарактеризуйте их.
26. Опишите отношения между взаимосвязными процессами.
27. Дайте определение понятию поток.
28. Какими двумя способами может выполняться планирование?
29. Чем отличаются однопоточные процессы от многопоточных процессов?
30. Как ОС распределяет процессорное время между такими единицами работы как потоки?
31. Расскажите о реализации потоков в различных ОС.

32. Какие подсистемы ОС вы знаете, чем они отличаются друг от друга? Их основные функции.
33. Дайте краткую характеристику подсистемы управления процессами.
34. Сколько процессов одновременно может существовать в мультипрограммных ОС?
35. Дайте определение понятию процесс.
36. Дайте краткую характеристику подсистемы управления памятью.
37. Какие три типа адресов требуется на разных этапах жизненного цикла программ для предоставления переменных и кодов?
38. Каким может быть виртуальное адресное пространство и на какие две непрерывные части оно делится?
39. Дайте краткую характеристику подсистемы управления файлами и устройствами ввода- вывода.
40. Какие средства защиты данных и администрирования вы знаете?
41. На какие направления различают API(Application program interface) и сколько существует вариантов их реализации?

42. Перечислите основные функции прерываний**.**
43. Что такое графический интерфейс пользователя (ГИП, . *graphical user interface,* GUI) в вычислительной технике?
44. Чем поддерживаются механизмы прерываний? Какие способы выполнения прерываний вы знаете?

45. Что такое программные прерывания? Как они реализуются?
46. Что вы знаете о шлюзах прерываний? Для чего они используются?
47. Что такое виртуализация? Что дало ИТ-организациям её появление?
48. Расскажите о технология Intel Virtualization.
49. В чем заключается важнейшее преимущество виртуализации?
50. Перечислите основные задачи синхронизации.

51. Перечислите основные компоненты ОС.
52. Сколько режимов работы должна поддерживать аппаратура компьютера (как минимум)?
53. Что такое привилегированный режим работы ОС?
54. Какие функции выполняет ядро?
55. Назовите главное отличие пользовательского режима от привилегированного режима.
56. Из каких слоёв может состоять ядро ОС:
57. Что такое менеджер ресурсов?

58. На какие группы делятся модули ОС?
59. Какие функции выполняют модули ядра?
60. Дайте понятие API.
61. На какие группы подразделяются вспомогательные модули ядра?
62. Какие модули называются транзитными?
63. Перечислите типовые средства аппаратной поддержки ОС.
64. Расскажите про механизм работы системы прерываний.
65. Какую ОС называют переносимой?
66. Назовите основные правила для обеспечения свойства мобильности ОС.

67. Что является основным недостатком микроядерной архитектуры? И какие преимущества использования микроядерной архитектуры вы знаете?
68. Каким образом использование микроядерной модели повышает надежность ОС?
69. Почему ОС на основе микроядра менее производительная, чем ОС с классическим ядром?
70. Для чего имитируются обращения к библиотечным функциям?
71. Перечислите способы реализации прикладных программных сред.

72. Что такое планирование? Решение каких задач оно включает в себя?
73. Какой планировщик называется статическим?
74. Что такое диспетчеризация? В чем она заключается?
75. Дайте понятия вытесняющей и не вытесняющей многозадачности. В чем заключается основное различие между этими вариантами многозадачности?
76. Какие классы алгоритмов планирования вы знаете?
77. Как может быть организована очередь готовых процессов в алгоритмах планирования, основанных на квантовании?

78. Дайте определение приоритета применительно к алгоритмам планирования, основанным на приоритетах.

79. Что такое смешанные алгоритмы планирования?
80. В каких случаях приоритеты могут изменяться планировщиком?
81. Какие события, требующие перераспределения процессорного времени вы знаете?
82. Расскажите о диспетчеризации и приоритезации прерываний в ОС.
83. Какие требования предъявляются к реализации системных вызовов?
84. Расскажите о диспетчеризации прерываний на примере Windows NT.
85. Что такое DPC?

86. Что такое блокирующие переменные?
87. Дайте понятие семафора. Что необходимо для работы с семафорами?
88. Для чего используются семафоры?
89. Что такое гонка? Для чего она нужна?
90. Как семафоры используются для синхронизации потоков?
91. Приведите примеры таких синхронизирующих объектов ОС.
92. Возможно ли использование файлов, потоков и процессов в качестве синхронизирующих объектов. Приведите примеры.
93. Какие задачи выполняют сигналы?

94. Как делится виртуальное адресное пространство при страничном распределении памяти?
95. Что такое таблица страниц? Что в ней содержится?
96. Что такое принцип невыгружаемости?
97. Назовите основные критерии выбора страницы для выгрузки из оперативной памяти.
98. В виде чего может быть представлен виртуальный адрес при страничном распределении?
99. Какие действия выполняются при каждом обращении к оперативной памяти аппаратными средствами?
100. Чем отличается сегментное распределение памяти от страничного?
101. Что является недостатком сегментного распределения памяти?

102. Как организовано сегментно-страничное распределение памяти?
103. Дайте понятие разделяемой памяти. Приведите примеры применения разделяемой области памяти.
104. Какие способы создания разделяемого сегмента памяти вы знаете?
105. Какие поля содержит дескриптор сегмента, для того, чтобы отличать разделяемые сегменты памяти от индивидуальных?

ЭКЗАМЕНАЦИОННЫЕ БИЛЕТЫ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)
Высшая школа печати и медиаиндустрии

Институт принтмедиа и информационных технологий
Кафедра ИиИТ
Дисциплина **«ОПЕРАЦИОННЫЕ СИСТЕМЫ»**
Направление подготовки 09.03.02 «Информационные системы и технологии

Курс 2, группа 171-721;2;3;4;5, форма обучения очная

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 1.

1. Знать определение ОС.
2. Уметь определять особенности алгоритмов управления ресурсами ЭВМ.
3. Владеть знанием критериев эффективности ОС.

Утверждено на заседании кафедры «28»августа 2022 г., протокол № 8.

Заведующий кафедрой ИиИТ,
доцент, к.т.н.  / Д.Г. Демидов /

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)
Высшая школа печати и медиаиндустрии

Институт принтмедиа и информационных технологий
Кафедра ИиИТ
Дисциплина **«ОПЕРАЦИОННЫЕ СИСТЕМЫ»**
Направление подготовки 09.03.02 «Информационные системы и технологии

Курс 2, группа 171-721;2;3;4;5, форма обучения очная

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 2.

1. Знать определение мультипрограммирования.
2. Уметь перечислить этапы развития ОС компьютеров. Охарактеризовать каждый из них.
3. Владеть пониманием того, что такое привилегированный программный модуль.

Утверждено на заседании кафедры «28»августа 2022 г., протокол № 8.

Заведующий кафедрой ИиИТ,
доцент, к.т.н.  / Д.Г. Демидов /

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)
Высшая школа печати и медиаиндустрии

Институт принтмедиа и информационных технологий
Кафедра ИиИТ
Дисциплина **«ОПЕРАЦИОННЫЕ СИСТЕМЫ»**
Направление подготовки 09.03.02 «Информационные системы и технологии

Курс 2, группа 171-721;2;3;4;5, форма обучения очная

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 3.

1. Знать, какие основные функции выполняют современные ОС.
2. Уметь охарактеризовать возможности мультипрограммирования.
3. Владеть знанием основных характеристик ОС WINDOWS.

Утверждено на заседании кафедры «28»августа 2022 г., протокол № 8.

Заведующий кафедрой ИиИТ,
доцент, к.т.н. / Д.Г. Демидов /

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)
Высшая школа печати и медиаиндустрии

Институт принтмедиа и информационных технологий
Кафедра ИиИТ
Дисциплина **«ОПЕРАЦИОННЫЕ СИСТЕМЫ»**
Направление подготовки 09.03.02 «Информационные системы и технологии

Курс 2, группа 171-721;2;3;4;5, форма обучения очная

ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ № 4.

1. Знать, на что подразделяются ОС по числу одновременно выполняемых задач.
2. Уметь охарактеризовать, какие средства управления включают в себя многозадачные и однозадачные ОС.
3. Владеть знанием основных различий между вытесняющими и невытесняющими алгоритмами многозадачности.

Утверждено на заседании кафедры «28»августа 2022 г., протокол № 8.

Заведующий кафедрой ИиИТ,
доцент, к.т.н.  / Д.Г. Демидов /

ЛАБОРАТОРНЫЕ РАБОТЫ.

Лабораторная работа №1
Знакомство с ОС UBUNTU.

Изучение команд интерпритатора:

`date` - определение текущей даты и времени;

`env` - вывод значении переменных среды;

`who. Id` - идентификация пользователей;

`write, mesg` - команды обмена прямыми сообщениями;

`mail` - отправление и чтение почтовых сообщений;

`more` - постраничный вывод содержимого файла на экран.

РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЯ В СИСТЕМЕ

В отличие от персональной операционной системы, с многопользовательской ОС UBUNTU могут работать одновременно несколько пользователей. Каждый из пользователей перед началом работы должен быть зарегистрирован в системе, тем самым ему разрешается доступ к ресурсам системы. Процедура регистрации пользователя в системе называется авторизацией и выполняется администратором системы.

Информация о всех зарегистрированных пользователях содержится в файле cat /etc/passwd. При авторизации в файл добавляется строка, в общем случае содержащая имя пользователя, зашифрованный пароль длиной 13 символов, неотрицательные числовые идентификаторы пользователя и группы пользователей, к которой он отнесен, тип группы, полное имя личного каталога пользователя, имя программного файла для запуска родительского интерпретатора shell.

Например, для пользователя с входным именем lev регистрационная строка может иметь вид:

lev:код_пароля:210:14:USER:/home/user/lev:/usr/bin/ksh

Команда просмотра текущих идентификаторов:

id – выводит идентификаторы пользователя и его группы для данного сеанса связи:

$id

uid=303 (user3) gid=300 (class)

210(lev) 14(user)

$

Идентификаторы назначаются администратором системы по профессиональной принадлежности. Пользователь может быть включен в одну или несколько групп, к файлам которых он может иметь доступ.

Система работает только с числовыми идентификаторами, а символьные имена – только для удобства их восприятия пользователями.

В принципе идентификаторы могут быть изменены с целью, например, дополнительной защиты файлов. Пользовательские идентификаторы содержатся в файле /etc/passwd

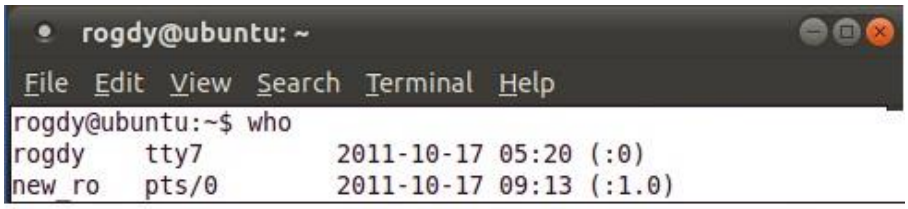
Идентификаторы групп – в файле /etc/group.

ГРУППА КОММУНИКАЦИОННЫХ КОМАНД

Рассматриваемая группа команд позволяет организовать взаимодействие между процессами нескольких пользователей. Обмен сообщениями между пользователями может быть начат по инициативе любого из пользователей.

Для уточнения реквизитов адресата можно воспользоваться командой:

`who - кто работает с системой ?`



Команда who в каждой строке своего сообщения выводит имя очередного пользователя, номер терминала, за которым он работает, дату и время начала работы этого пользователя.

```
rogdy@ubuntu:~$ who am i
rogdy    pts/0        2011-10-17 08:51 (:0.0)
```

Для посылки текстовых сообщений пользователю-адресату пользователь-отправитель может воспользоваться командой:

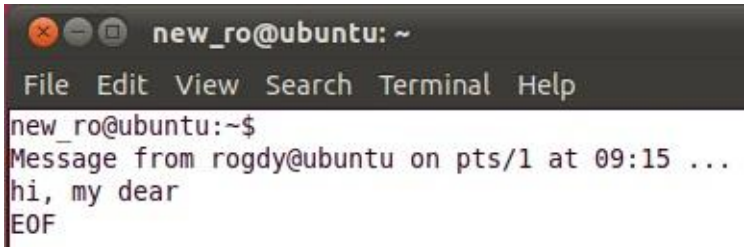
`write – послать абоненту сообщение`

```
rogdy@ubuntu:~$ write new_ro
write: new_ro is logged in more than once; writing to pts/0
hi, my dear
```

Ctrl+D

```
rogdy@ubuntu:~$
```

Результатом работы команды является "вторжение" на экран адресата (в данном случае, пользователя new_ro) приоритетного по отношению к текущей экранной выдаче приглашения к общению с отправителем и текст передаваемого сообщения с идентификатором конца сообщения <EOF>:



Если отправитель обращается к неработающему в настоящее

время пользователю-адресату, то работа команды write завершается выдачей системного сообщения об ошибке:

```
write ololo
write: ololo is not logged in
```

Блокирование выдачи на собственный экран несанкционированных сообщений других пользователей может осуществляться с помощью команды:

```
rogdy@ubuntu:~$ mesg n
rogdy@ubuntu:~$ mesf y
```

Многопользовательская ОС UBUNTU предоставляет возможности для организации электронной почты на базе команды:

mail - отправка или чтение почты

Команда mail имеет множество модификаций, задаваемых ключами и аргументами команды, предоставляя мощные средства поддержки разнообразных форм взаимодействия: от организации обмена сообщениями с их хранением в "почтовом ящике" до формирования и редактирования самих сообщений.

Роль почтового ящик выполняет специальный системный файл.

Каждое сообщение сопровождается заголовком, идентифицирующим пользователя-отправителя. Если при очередном входе пользователя в систему в почтовом ящике для него есть хотя бы одно сообщение, то систем выводит на экран:

```
You have mail
rogdy@ubuntu:~$
```

Дальнейшие действия пользователя подразумевают следующий диалог:

```
$ mail
""""""""""""""""""""""""""""""""""""""""""""""""""""""
From petr11 Thu Mar 11 19:10 MDT 2005
< Текст сообщения >
""""""""""""""""""""""""""""""""""""""""""""""""""""""
?
```

Последняя строка сообщения представляет собой приглашение системы mail '?' для ввода следующей команды, определяющей, что надо делать с прочитанными сообщениями. Возможны команды:

<+> - вывод на экран следующего сообщения;

<-> - вывод предыдущего;

<d> - удалить текущее сообщение,

<q> - сохранить в буфере,

<S [файл]> – сохранить в файле (по умолчанию – файл «mbox»)

<![команда]> - выполнение команды ОС не выходя из утилиты mail

После этого автоматически выводится очередное сообщение со своим заголовком и следующее за ним приглашение

'?'.

Если в "почтовом ящике" нет сообщений для Вас, то протокол работы с mail будет выглядеть так:

```
rogdy@ubuntu:~$ mail
No mail for rogdy
rogdy@ubuntu:~$
```

Для формирования и посылки Ваших посланий в качестве аргументов указываются входные имена пользователей-адресатов:

```
rogdy@ubuntu:~$ mail new_ro
Subject: hey! how are u, dear?
EOT
```

Текст сообщения может быть передан из текстового файл , также как и в команде write.

1.1 Методика выполнения.

1.войдите в систему с зарегистрированным администратором логическим именем и паролем. Проанализируйте сообщение системы. Чем заканчивается сообщение системы?

2. проанализируйте содержание системного файла cat/etc/passwd.найдите запись, относящуюся к Вам.

3. детально проанализируйте и объясните каждое поле записи, его значение.

4. выведите на экран значения переменных среды. Проанализируйте назначение переменных.

5. какая переменная определяет текст приглашения? Измените текст приглашения. Восстановите стандартное значение приглашения.

6. выведите текущую дату и время. Проанализируйте текст сообщения.

7. определите пользователей системы, работающих с системой параллельно с вами, их логические имена и номера терминалов.

8. договоритесь с соседним пользователем об организации обмена прямыми сообщениями. Обменяйтесь с ним сообщениями в режиме прямого диалога.

9. исследуйте возможности средств блокирования и разблокирования приема сообщений.

10. по договоренности с коллегами обменяйтесь несколькими почтовыми сообщениями.

11. проанализируйте возможности обработки поступивших почтовых сообщений.

12. определите числовые идентификаторы вас как пользователя и вашей группы.

13. проанализируйте с помощью команды history содержание лабораторной работы, продумайте ответы на нижеприведенные контрольные вопросы и сдайте выполненную работу

преподавателю. После получения зачета по работе – уничтожьте все созданные файлы и корректно выйдите из системы.

1.2 Контрольные вопросы.

1. Объясните назначение информации, запрашиваемой системой в начале работы.
2. В чем заключается процедура авторизации пользователя? Цель авторизации?
3. Объясните содержание и назначение каждого поля регистрационной записи.
4. Какая операционная система вас обслуживает и какой shell?
5. Что такое среда пользователя? Как она формируется?
6. В чем отличие в диалоге прямыми сообщениями и почтовыми?
7. Определение возможности электронной почты. Какие режимы работы электронной почты. Вы знаете?
8. Какое назначение числовых идентификаторов пользователей и групп в работе UBUNTU?

Каждая строка (учетная запись) в файле **/etc/passwd** описывает одного известного системе пользователя и имеет семь разделенных двоеточиями полей. Пример записи:

```
user_01:x:169:10:Student:/home/user_01:/bin/sh
```

Назначение полей этой записи представлено в следующей таблице.

Таблица 1. Поля файла /etc/passwd и их назначение

Поле	**Назначение**
Имя пользователя (регистрационное имя)	Содержит символьное имя пользователя, используемое при регистрации в системе. В пределах одной машины должно быть уникальным. Регистрационное имя должно состоять из алфавитно-цифровых символов (нижнего регистра), без пробелов, с максимальной длиной, определяемой конкретной ОС. Наиболее часто используется максимальная длина - восемь символов. Дублирование имен пользователей приводит к определенным осложнениям. Например, дубликаты появляются тогда, когда администратор использует в имени более 8 символов. Тогда для системы **jarmstrong** то же, что **jarmstroff**. Когда имя так продублировано, система использует первую найденную для него запись в файле **/etc/passwd** и игнорирует последующие.
Пароль	Поле хранит зашифрованный пароль. Допускается пустое поле. При использовании системы теневого хранения паролей, в этом поле находится только метка пароля (**x**), а зашифрованный пароль хранится в другом месте. Правила задания пароля обычно находятся в файле **/etc/default/passwd**, (например, директива **PASSLENGTH=число** в этом файле задает минимальное количество символов в пароле). Некоторые системы также учитывают регистр, а в некоторых предусматривается использование как минимум одного не алфавитно-цифрового символа.
Идентификатор пользователя	Поле хранит числовой идентификатор пользователя, который связан с его регистрационным именем. Любой созданный пользователем файл

	или запущенный процесс ассоциируется с его числовым идентификатором.
Идентификатор группы	Содержит числовой идентификатор группы. Любой созданный пользователем файл ассоциируется с его идентификатором группы. Указанная здесь группа является основной (первичной) для данного пользователя.
Комментарий	Содержит комментарий - любую алфавитно-цифровую строку. Предположительно это поле содержит информацию о реальном владельце регистрационного имени. ОС UBUNTU не задает его формат, так что подойдет любой. Некоторые программы печати и электронной почты используют это поле для вывода настоящего имени пользователя.
Начальный каталог	Определяет начальный каталог пользователя. Когда пользователь начинает сеанс работы, система помещает его в данный каталог. Пользователь должен иметь соответствующие права доступа к нему.
Начальная команда	Определяет командную среду пользователя (обычно запускается один из *командных интерпретаторов* UBUNTU, но, теоретически, можно указать любую команду). Это поле можно изменять.

Файл /etc/group

Этот файл соотносит числовые идентификаторы групп с символьными именами. Каждая строка файла /etc/group содержит четыре поля. Поля разделяются двоеточиями. Назначение полей этой записи представлено в табл. 2.

Таблица 2. Поля файла /etc/group и их назначение

Поле	**Назначение**
Имя группы	Содержит (уникальное) символьное имя группы.
Пароль группы	Группы могут иметь пароли, хотя использование паролей групп - явление редкое. В примере данное поле пустое - это значит, что пароль отсутствует.
Идентификатор группы	Содержит числовой идентификатор группы.
Список пользователей	Содержит список регистрационных имен пользователей данной группы. Имена в этом списке разделяются запятыми. Пользователи могут принадлежать к нескольким группам и, при необходимости, переключаться между ними с помощью команды newgrp.

Пример записи из файла /etc/group:

```
bin::2:root,bin,daemon
```

Лабораторная работа №2

Управление каталогами.

Эта работа посвящена изучению структуры файловой системы и возможностей командного языка UBUNTU по управлению каталогами.

Изучаются команды:

`mkdir, rmdir` - для создания и уничтожения каталогов;

`ls` - вывод листинга каталога. «что здесь есть?»;

`pwd` - вывод на экран полного имени текущего каталога, «где я нахожусь?»;

`cd` - смена текущего каталога, «перейти в»;

`find, grep` - поиск файлов в системе каталогов;

`>маршрутное` – имя - файла - создание пустого файла.

Эта работа посвящена изучению структуры файловой системы и возможностей командного языка Ubuntu по управлению каталогами. Для начала рассмотрим основные команды, используемые в ОС UBUNTU при работе с каталогами..

Разделителем элементов пути в системах UBUNTU служит символ V. В отличие от таких систем, как, например, MSDOS и Windows, в которых каждому тому соответствует отдельный корневой каталог, обозначаемый именем тома, в системе UBUNTU есть только один корневой каталог. Он обозначается ‘/’ - Все дополнительные тома, подключаются к основному дереву каталогов так, что корневой каталог каждого из этих томов становится просто одним из каталогов в файловой системе. Каталог для подключения может быть выбран произвольно. Операция подключения тома к файловой системе носит название монтирования, и может производиться в любой момент во время работы системы.

В системе обычно присутствуют следующие каталоги:

/ — корневой каталог;

`/bin`— каталог с пользовательскими программами;

`/sbin` — каталог программ для администрирования системы;

`/etc` — каталог с конфигурационными файлами программ;

`/home` — каталог, в котором создаются домашние каталоги пользователей;

`/lib` — каталог с динамическими и статическими библиотеками;

`/boot` — каталог, содержащий файлы системного загрузчика;

`/mnt` — каталог, в который, как правило, производится монтирование;

`/dev` — каталог, содержащий специальные файлы устройств;

`/opt` — каталог, в который устанавливается ПО сторонних производителей;

`/usr` — каталог, в котором хранятся в режиме доступа только для чтения разделяемые данные, такие как исполняемые файлы программ, документация, библиотеки и другие системные ресурсы;

/root— каталог, являющийся домашним для пользователя roo1;

/var — каталог, содержащий журналы, файлы баз данных, кеши разного рода; .

/tmp — каталог для хранения временных файлов.

Для печати текущего каталога:

pwd

Для смены каталога:

cd <путь к каталогу>

Пример: переход к каталогу var.

```
rogdy@ubuntu:~$ cd /var
rogdy@ubuntu:/var$ pwd
/var
rogdy@ubuntu:/var$ ls
backups  cache  crash  games  lib  local  lock  log  mail  opt  run  spool  tmp
rogdy@ubuntu:/var$
```

Попробуйте объяснить смысл каждой строки из данного примера. Какие операции запрашивает пользователь, и какие ответы даёт система?

Если команда запущена без указания каталога, то переход производится в домашний каталог пользователя. Вообще, для указания домашнего каталога пользователя можно использовать специальный символ ‘ ~’. Так, для перехода в папку tmp, находящуюся в домашнем каталоге можно воспользоваться следующей командой:

```
rogdy@ubuntu:/var$ cd ~ /tmp
```

Домашний каталог пользователя обычно располагается в каталоге /home и называется по имени пользователя. Например, для пользователя user1 домашний каталог будет таким: /home/user1.

Для создания каталога:

mkdir <список имен каталогов>

```
rogdy@ubuntu:~$ mkdir KaTaLoG
```

Если требуется создать сразу несколько вложенных друг в друга каталогов, можно воспользоваться ключом -p:

```
rogdy@ubuntu:~$ mkdir -p vheshny/vnutrenny
```

Для удаления каталога:

```
rogdy@ubuntu:~$ rmdir KaTaLoG
```

Пример: создание каталогов и работа с ними.



Попробуйте объяснить смысл каждый строки из данного примера. Какие операции запрашивает пользователь, и какие ответы даёт система?

Команда удаляет только пустые каталоги. Ключ -p подобен такому же ключу команды mkdir, и позволяет удалить сразу несколько каталогов, вложенных друг в друга, если все они пусты.

/ Корневой каталог. Это родительский каталог для всех каталогов и файлов в файловой системе UBUNTU.

/bin Каталог исполняемых модулей командной строки. Данный каталог содержит все исполняемые модули «родных» команд UBUNTU.

/dev Каталог устройств, содержащий специальные файлы для байт-ориентированных и блок-ориентированных устройств, таких как принтеры и клавиатуры. В данном каталоге существует файл под называнием null, который называется bit bucket и который может использоваться для перенаправления вывода в никуда.

/etc Файлы системной конфигурации и каталог исполняемых файлов. Большая часть административных файлов, а также файлов, связанных с командами, хранится здесь.

/lib В каталоге хранятся библиотеки компилятора С.

/lost+found Данный каталог содержит обрабатываемые файлы, если система отключилась ненормально. Система использует данный каталог для восстановления файлов. В каждом разделе диска есть только один каталог lost+found.

/usr Данный каталог имеет несколько подкаталогов, таких как adm, bin, etc и include. Например. /usr/include содержит файлы заголовков для компилятора С.

/home содержит домашние каталоги пользователей.

Для создания каталогов используется команда mkdir. Можно указывать как полный так и относительный путь. Поэтому можно создавать дерево каталогов: определить относительно или абсолютно корень, после чего создать относительно нового каталога новые поддирективы.

Команду ls (с ее многочисленными опциями) можно использовать для получения информации об одном или нескольких файлах или каталогах системы. Используйте ls для генерации списка файлов и каталогов в различном порядке, например по имени или по времени. Возможно распечатывать лишь отдельные детали о файлах и каталогах, например только имя файла.

2.1 Методика выполнения.

1. Определите уникальное имя вашего головного личного каталога. Объясните структуру полного маршрутного имени каталога.

2. Создайте два поддерева из одного и из двух каталогов.

3. С использованием команды ls проверьте факт построения дерева подкаталогов.

4. Посмотрите содержимое пустых подкаталогов, т.е. новых подкаталогов, не содержащих файлов. Объясните их содержание.

5. Сделайте текущим последний каталог меньшего дерева.

6. Определите полное маршрутное имя.

7. Смените текущий последний каталог на подкаталог большего дерева.

8. Определите его полное маршрутное имя.

9. Поместите в созданные подкаталоги по 2-3 пустых файла не выходя из текущего. Используйте при этом разные способы задания маршрутного имени подкаталогов.

10. Просмотрите содержимое каталогов. Объясните содержания каждого поля каталога.

11. Установите в качестве текущего HOME-каталога.

12. Найдите обычные файлы с определением их полных маршрутных имен. Выполните то же для различных комбинаций известных вам условий поиска файлов.

13. Проделайте предыдущие задания для файлов типа каталог.

14. Выведите на экран принадлежащую вам регистрационную запись с использованием команды grep.

15. Уничтожьте все построенные вами подкаталоги. Получите подтверждение выполнения команд по содержимому домашнего каталога.

16. Проанализируйте с использованием команды history содержание лабораторной работы, продумайте ответы на нижеприведенные контрольные вопросы и сдайте выполненную работу преподавателю. После получения зачета по работе – уничтожьте все созданные файлы и корректно выйдите из системы.

2.2 Контрольные вопросы.

1. Какие системные имена каталогов вам известны?

2. Каким образом можно построить отдельный каталог или цепочку каталогов?

3. Для чего и каким образом переопределяются текущие каталоги?

4. Как обратиться к файлам параллельных ветвей дерева каталогов? К вышележащему каталогу?

5. Какие условия поиска файлов вы знаете? Как комбинируются условия поиска? Как осуществляется поиск по дереву каталогов?

6. Какова последовательность действий при удалении одного каталога? Цепочки каталогов?

7. Объясните назначение и содержание каждого поля каталога.

8. Как отличить по содержимому каталога типы файлов, содержащихся в ваших каталогов.

9. Какую информацию содержит «пустой» вновь созданный каталог?

10. Как осуществить поиск файлов в системе каталогов по фрагментам текста файлов?

Лабораторная работа №3

Управление файлами.

Эта работа посвящена изучению приемов формирования и преобразования файлов в ОС UBUNTU.

Изучаются команды:

`cat, cp` - копирование файлов;

`mv` - перемещение и переименование файлов;

`ln` - организация ссылок на файл;

`sort` - сортировка файлов;

`wc` - определение числовых параметров файла;

`touch` - обновление временных характеристик файла.

Для подсчета строк, слов, или символов в файле:

`wc <ключ> <путь к файлу>`

С командой используются следующие ключи:

`-l` — для подсчета числа строк;

`-w` — для подсчета числа слов;

`-c` — для подсчета числа символов.

Для вывода содержимого файла на экран:

`cat <список путей к файлам>`

Если cat указано в списке более одного пути, то содержимое указанных файлов будет выведено последовательно. Таким образом, команду cat можно использовать для соединения нескольких файлов в один.

Для того чтобы записать результат соединения в целевой файл, следует воспользоваться приемом перенаправления вывода. Для указания целевого файла следует добавить в строку, запускающую команду на выполнение следующую конструкцию:

`> <путь к файлу результату>`

Например, для вывода сообщения «He11o, world!» в файл bloknot.txt, можно воспользоваться следующей командой:

```
rogdy@ubuntu:~$ echo Hello, world! > vheshny/bloknot.txt
```

Если требуется произвести запись в конец уже существующий файл с сохранением уже записанной туда информации, используется следующая конструкция:

Например:

```
rogdy@ubuntu:~$ echo how are you, world? >> vheshny/bloknot.txt
```

Подобным образом, можно осуществлять перенаправление не только вывода, но и ввода. В этом случае, если программа ожидает ввода пользователя, вместо него будет подставлено содержимое указанного файла.

< <путь к файлу-источнику>

В качестве примера, рассмотрим запуск команды cat без указания файла. В этом случае, программа будет ожидать от пользователя ввода данных, и выводит их.

```
rogdy@ubuntu:~$ cat > vheshny/bloknot.txt
hi!
```

^Z

Теперь, применим прием перенаправления ввода, указав cat, что ввод требуется осуществить из только что полученного файла.

```
rogdy@ubuntu:~$ cat < vheshny/bloknot.txt > bloknot2.txt
```

В случае, когда необходимо передать результаты, выводимые одной командой на вход другой команде можно воспользоваться конвейером. Конвейер строится при помощи символа ‘|’, которым разделяются команды. Например, следующую последовательность команд:

```
rogdy@ubuntu:~$ echo I love you > mybloknot.txt
rogdy@ubuntu:~$ wc -w < mybloknot.txt
3
```

Можно заменить таким конвейером:

```
rogdy@ubuntu:~$ echo I love you | wc -w
3
```

(функция wc –w считает количество введённых слов, разделённых пробелами)

Для постраничного просмотра файлов используется функция more:

```
rogdy@ubuntu:~$ more mybloknot.txt
I love you
```

Используйте клавишу Enter для перехода к следующей строке просматриваемого файла, клавишу пробела для перехода к следующей странице просматриваемого файла. Для выхода нажмите клавишу q.

Часто команда тоге используется для постраничного просмотра результата работы другой команды, для чего она объединяется с этой программой в конвейер. Например:

```
rogdy@ubuntu:~$ ls /bin | more
```

Для соединения файла, после разделения командой cut

```
rogdy@ubuntu:~$ paste mybloknot.txt bloknot2.txt
I love you      hi!
```

Для сортировки содержимого файла:

```
rogdy@ubuntu:~$ sort mybloknot.txt bloknot2.txt
hi!
I love you
```

Результат сортировки не записывается в исходные файлы, а выводится на экран. Команда имеет следующие ключи:

`-r` — производить сортировку по убыванию, вместо сортировки по возрастанию;

`-n` — рассматривать содержимое файлов как числа, и производить для сортировки числовое сравнение, а не лексикографическое. *Для создания нового пустого файла:*

`touch <список путей к файлам>`

Если файл уже существует, то у него будет изменено время последнего обращения.

Для копирования файла:

`cp <путь к исходному файлу> <путь к файлу назначения>`

`cp <список путей к исходным файлам> <путь к каталогу назначения>`

`cp -r <список путей к исходным файлам и/или каталогам> <путь к каталогу назначения>`

`Пример работы с файлами`

```
rogdy@ubuntu:~$ cd Documents
rogdy@ubuntu:~/Documents$ echo MGUP forever > mgup.txt
rogdy@ubuntu:~/Documents$ more mgup.txt
MGUP forever
rogdy@ubuntu:~/Documents$ rm mgup.txt
```

`Попробуйте объяснить смысл каждый строки из данного примера. Какие операции запрашивает пользователь, и какие ответы даёт система?`

Заметим, что для указания всех файлов в текущем каталоге в качестве списка путей в исходном файле, можно использовать специальный символ *, известный как `wildcard`. При этом команде `cp` символ * передан не будет. При обработке строки с командой и ее параметрами программа оболочка заменит его списком имен всех файлов, найденных в текущем каталоге. Вообще, кроме символа *, при задании шаблона для генерации списка файлов, могут использоваться также символы `?, [` и `]`. При этом они имеют следующее значение:

`*` — любая последовательность символов. Например, шаблон `/home/*a*` будет заменен списком всех файлов и каталогов в каталоге /home, содержащих символ a;

`?` — любой одиночный символ. Например, шаблон `/usr/???` будет заменен списком имен всех файлов и каталогов в каталоге `/usr`, имеющих длину 3;

`[ ]` — любой из символов, заданных в скобках. Например, шаблон `/lib/*[.0-9]*` будет заменен списком имен всех файлов и каталогов в каталоге /lib, имеющих в имени символ точки, или цифру;

`[^]` — любой из символов, кроме символов, заданных в скобках. Например, шаблон `/lib/[^aeiou] *` будет заменен списком имен всех файлов и каталогов в каталоге /bin, не начинающихся на гласную букву.

Так как обработка шаблонов генерации списков производится программой-оболочкой, использовать шаблоны для передачи списков файлов в качестве параметров любым программам, в том числе и не предназначенным для работы с файлами.

Следующий пример выведет список всех файлов и каталогов в корневом каталоге:

```
rogdy@ubuntu:~$ echo /*
/bin /boot /cdrom /dev /etc /home /initrd.img /lib /lost+found /media /mnt /opt
/proc /root /sbin /selinux /srv /sys /tmp /usr /var /vmlinuz
```

В случае если список, сгенерированный по шаблону оказывается пустым, то есть файлов, имена которых удовлетворяют критерию не обнаружено, шаблон передается программе, обрабатывающей команду в неизменном виде со всеми специальными символами. Вот, например, результат выполнения команды в пустом каталоге:

```
echo   *
*
```

Для перемещения файла:

mv `<путь к файлу> <новый путь для файла>`

`пример:`

```
rogdy@ubuntu:~$ mkdir old_adres
rogdy@ubuntu:~$ echo hi! > old_adres/1.txt
rogdy@ubuntu:~$ mkdir new_adres
rogdy@ubuntu:~$ mv old_adres/1.txt new_adres
```

Отметим, что команда перемещения файла может также быть использована для его переименования.

Для удаления файла:

```
rm <список путей к файлам>
rm -r <список путей к файлам и/или каталогам>
rogdy@ubuntu:~$ rm new_adres/1.txt
```

Для поиска файлов, удовлетворяющих критерию:

```
find <список путей для начала поиска> [опции] <список критериев поиска>
rogdy@ubuntu:~$ find new_adres -name bloknot.txt
new_adres/bloknot.txt
```

В случае отсутствия критериев поиска, результатом команды будут все файлы, расположенные во всех подкаталогах всех указанных для поиска каталогов. Причем, командой find наличие,

или отсутствие у файла в начале имени точки не учитывается, то есть в списки файлов попадут все файлы, включая скрытые. Перечислим некоторые критерии поиска:

`-name <шаблон>` — файл будет включен в результат только, если его имя (без учета пути к файлу) соответствует шаблону программы-оболочки, переданному а качестве параметра критерию;

`-type <тип>` — файл будет включен в результат только, если его тип соответствует указанному в критерии; среди всех возможных типов выделим следующие: `d` — для каталога, `f`— для файла.

В случае если требуется ограничить глубину поиска, следует воспользоваться опцией `-maxdepth`, которой в качестве параметра передается максимальная глубина спуска в подкаталоги. Например, чтобы производить поиск только среди файлов текущего каталога, имена которых начинаются не на точку, можно использовать такую команду:

```
find . -maxdepth l -name "[^.]*"
```

Заметим, что в предыдущем примере, шаблон генерации списка был заключен в кавычки, чтобы подавить генерацию списка программой-оболочкой при разборе параметров.

Для получения информации о размере файлов или каталогов:

```
du <список путей к файлам или каталогам>
```

Для каждого файла из списка параметров программа печатает его размер, а для содержащихся в списке параметров каталогов, и всех каталогов, содержащихся в них — суммарный объем хранящихся в них файлов.

3.1 Методика выполнения.

1. Выведите на экран содержимое вашего HOME-каталога.
2. Создайте 3-4 текстовых файла с частично совпадающими именами. Проанализируйте значение атрибутов ваших файлов.
3. Создайте еще один файл методом слияния из существующих. Как изменились атрибуты нового файла?
4. Создайте два новых параллельных подкаталога.
5. В один подкаталог скопируйте имеющиеся файлы HOME-каталога с изменением имен, а в другой – переместите. Проанализируйте как изменилось содержание всех трех каталогов и каковы атрибуты всех полученных файлов.
6. Просмотрите содержимое файлов с частично совпадающими именами с использованием механизма генерации имен файлов.
7. С использованием механизма генерации имен файлов слейте содержимое всех файлов в один и поместите его в HOME-каталог. Проанализируйте содержания файла.
8. Создайте ссылку с другим именем в одном из подкаталогов на один из файлов другого подкаталога. Проанализируйте и сравните все атрибуты связанных файлов. Объясните отличия.

9. Создайте еще одну ссылку с другим именем. Проанализируйте и сравните все атрибуты связанных файлов. Объясните отличия.

10. Создайте ссылки на несколько файлов, принадлежащих одному из каталогов, в другом каталоге одной командной строкой. Проанализируйте и сравните все атрибуты связанных файлов. Объясните отличия.

11. Изменение содержимое одного из связанных файлов. Проанализируйте содержимое других связанных с ним файлов (или файла). Объясните результат.

12. Уничтожьте один из связанных файлов. Проанализируйте и сравните все атрибуты связанных с ним файлов. Объясните изменения.

13. Упорядочите по алфавиту строки суммарного файла HOME-каталога.

14. Обновите временные характеристики одного из существующих файлов. Проанализируйте результат работы команды.

15. Обновите временные характеристики несуществующего файла. Проанализируйте результат работы команды.

16. Проанализируйте с использованием команды history содержание лабораторной работы, продумайте ответы на нижеприведенные контрольные вопросы и сдайте выполненную работу преподавателю. После получения зачета по работе – уничтожьте все созданные файлы и корректно выйдите из системы.

3.2 Контрольные вопросы.

1. Назовите известные вам способы создания пустых файлов.

2. Как создать текстовый файл?

3. Какие возможности сокращения записи имен файлов вы знаете с использованием механизма генерации имен файлов?

4. Какие три команды этой лабораторной работы можно использовать для переименования файлов? Как в этом случае надо использовать команды?

5. Какими способами можно объединить несколько текстовых файлов в один?

6. В чем разница работы команд cp и mv?

7. Сколько ссылок можно создать на единственный файл из разных каталогов?

8. Как создать несколько ссылок с совпадающими именами на несколько файлов в другом каталоге?

9. Какое соответствие атрибутов имеют связанные между собой файлы?

10. На какой атрибут и как влияет удаление одного из связанных файлов?

11. Как отражается на содержимом связанных файлов изменение содержания одного из них и почему?

12. Какими возможностями обладает команда sort?

Лабораторная работа №4

Управление правами доступа к файлам и каталогам.

Управление правами доступа к файлам и каталогам.

Эта работа посвящена изучению принципов защиты файлов и каталогов ОС UBUNTU от несанкционированного доступа. В UBUNTU эти средства являются встроенными и наиболее развиты по сравнению с другими операционными системами. Изучаются вопросы влияния задаваемых прав доступа к файлу на выполнение различных команд по обработке этих файлов.

Для управления правами доступа к файлам и каталогам используется команда: **chmod** – изменить права доступа к указанному файлу.

. Изучаются вопросы влияния задаваемых прав доступа к файлу на выполнение различных команд по обработке этих файлов.

Для модификации прав доступа к файлу:

`chmod <права доступа> <список путей к файлам и каталогам>`

Если в списке путей присутствуют каталоги, можно указанием ключа -R осуществить смену прав доступа для всех файлов и каталогов, вложенных в эти каталоги.

Права доступа указываются следующим образом: `< [ugoa ] > <+ | -| =>< [rwx] >,` где:

`u` — владелец файла (user);

`g` — группа пользователей, ассоциируемая с файлом (group);

`o` — прочие пользователи (other);

`a` — все пользователи системы указывают категорию пользователей, для которой производится изменение прав доступа (all);

`+` — добавление;

`-` — изъятие;

`=` — установка прав доступа;

`r`— права на чтение;

`w` — права на запись;

`x` — права на выполнение указывают виды добавляемые, изымаемые, или устанавливаемых прав доступа.

Например:

`a+r` — добавляет право чтения их файла для всех пользователей системы;

ug=rwx — устанавливает все права доступа для владельца файла и пользователей, состоящих в группе, ассоциированной с файлом;

og-wx — изымает права на запись и выполнение файла у всех пользователей системы, не являющихся владельцами файла, в том числе, и у пользователей, состоящих в группе, ассоциированной с файлом.

Для каталогов виды прав доступа имеют следующий смысл:

чтение — получение списка файлов каталога и их атрибутов;

запись — создание в каталоге файлов и каталогов, их удаление,

изменение;

выполнение — осуществление входа в каталог, или один из вложенных в него каталогов, например, командой сё.

Например, чтобы запретить всем остальным пользователям производить листинг своего домашнего каталога, пользователь rogdy мог бы ввести следующую команду:

```
rogdy@ubuntu:~$ chmod og-r /user/rogdy
```

Для того чтобы пользователь мог запустить файл сценария, необходимо, чтобы он в отношении этого файла обладал правами чтения и запуска. Например, чтобы добавить их всем пользователям системы, можно воспользоваться следующей командой:

```
chmod a+rx my_script.sh
```

Пример работы с правами доступа

```
rogdy@ubuntu:~$ cd new_adres
rogdy@ubuntu:~/new_adres$ chmod a-r bloknot.txt
rogdy@ubuntu:~/new_adres$ more bloknot.txt
bloknot.txt: Permission denied
rogdy@ubuntu:~/new_adres$ chmod a+r bloknot.txt
rogdy@ubuntu:~/new_adres$ more bloknot.txt
I love You
```

Попробуйте объяснить смысл каждый строки из данного примера. Какие операции запрашивает пользователь, и какие ответы даёт система?

4.1 Методика выполнения.

1. Создайте в вашем HOME-каталоге один текстовый файл, например с именем fl. Выведите на экран полный листинг каталога.

2. Проанализируйте и умейте объяснить какие права доступа к fl имеет владелец файла, его группа и остальные пользователи.

3. Проанализируйте права доступа к вашему головному каталогу. Есть ли ограничения на работу с файлами в этом каталоге?

4. Выведите на экран содержимое файла fl. Объясните почему операция выполнилась успешно.

5. Запретите права на чтение fl владельцу и групп попытайтесь вывести на экран текст файла. Объясните, почему операция не выполняется.

6. Удалите права на запись в файл. Попытайтесь добавить к файлу текст и удалить его. Объясните результат.

7. Удалите право на модификацию каталога. Повторите операцию удаления. Объясните результат.

8. Создайте подкаталог. Разместите в нем текстовый файл. Проанализируйте права доступа к подкаталогу и объясните возможности по использованию подкаталога.

9. Удалите право владельца на «выполнение» подкаталога.

10. Попытайтесь сделать подкаталог текущим. Объясните результат.

11. Просмотрите содержимое подкаталога. Объясните результат.

12. Попытайтесь вывести длинный листинг подкаталога только для одного из файлов (поиск файла по каталогу). Объясните результат.

4.2 Контрольные вопросы.

1. Как кодируются в атрибутах файла и каталога права доступа?
2. Кто может пользоваться и изменять права доступа к файлам?
3. Какие команды для изменения символьных кодов прав доступа вы знаете? Перечислите и расскажите о назначении каждой из команд.
4. В чем разница в применении команд chmod и umask?
5. Какие команды обработки файлов разрешают (или запрещают) права на чтение, модификацию и исполнение?
6. Какие команды обработки каталогов разрешают (или запрещают) эти же права?
7. Что означает право на выполнение, применительно к каталогу?
8. Какими правами надо обладать, чтобы удалить файл или каталог?
9. Какие команды для защиты файлов вы знаете?

Лабораторная работа №5
Текстовый редактор vi ОС UBUNTU.

Эта работа посвящена изучению основных возможностей встроенного текстового редактора `vi` – наиболее распространенного средства для построения текстовых файлов, исходных текстов программ и `shell`-процедур.

Изучаются команды:

- вход в редактор и выход, сохранение файлов;
- ввода текста;
- удаления фрагментов текста;
- копирования фрагментов текста в буферную

область памяти;
- вставки содержимого буфера в текст файла;
- редактирования (изменения) текста;
- поиска строк файлов по их фрагментам;
многострочных операций с файлом (префиксные
команды)

vi — текстовый редактор командной строки в Ubuntu и других ОС UBUNTU. При запуске vi не открывается новое графическое окно, просмотр и редактирование файла производится в текущем окне терминала. За счет того, что vi может работать и без Графического пользовательского интерфейса (GUI), он может использоваться для редактирования файлов.

Запуск редактора:

а) vi   myfile (одно или несколько имен файлов через пробелы для последовательного вызова их на редактирование).

Если такого файла нет, то появится начало пустого файла; курсор установится в начале первой строки.

б) vi + myfile

На экране конец файла; а курсор - в начале последней строки.

Пример:



в) vi +10 myfile

На экране - часть файла и строка 10 - в центре экрана;

курсор - в начале этой строки.

Режимы работы.

Редактор работает в нескольких режимах:

3. В режиме вставки (редактирования). Нажатие на клавишу приводит к вставке соответствующего символа в редактируемый текст.
4. В командном режиме
    5. нажатие на любую клавишу воспринимается как команда редактору, которая немедленно исполняется;

6. в этом режиме можно ввести команду с параметрами в командной строке.

Поэтому при работе с Vi пользователю всегда нужно обращать внимание на то, в каком режиме находится редактор.

Командный режим.

Редактор Vi всегда начинает работу в командном режиме. В этом режиме есть два способа отдавать команды редактору.

- нажатие практически на любую клавишу редактор воспринимает как определенную команду. Команды не отображаются, а сразу выполняются. Например, нажатие стрелок не перемещает курсор, а выводит на экран символы, соответствующие командам, расположенным по этим клавишам. Переключиться из режима вставки в командный режим - при помощи Esc.
- для ввода более сложных многословных команд используется командная строка, вызов которой осуществляется нажатием клавиши ":" .

Режим вставки (редактирования).

Для перехода к этому режиму следует использовать команду insert, выполнение которой осуществляется при нажатии на клавишу "i" в командном режиме. При переключении в этот режим внизу экрана появляется надпись Insert, можно вносить изменения в текст документа.

Выход из vi

а) с сохранением изменений: переключиться в командный режим (Esc), ввести команду

```
: w q
```

б) без сохранения изменений: переключиться в командный режим (Esc), ввести команду

```
: q !
```

Перемещение курсора

0 - в начало строки

$ - в конец строки

w - в начало слова

b - в начало предыдущего слова

Удаление

dw - удаление слова

d$ - удаление до конца строки

d0 - удаление до начала строки

d7w - удаление 7 слов

Изменение (замена)

c$ (или C)   <текст замены (может быть из нескольких строк)>

Esc - замена конца строки (от курсора);

c^ <текст замены> Esc - замена от начала строки до курсора

cc < текст > Esc - замена одной строки;

5cc < текст > Esc - замена пяти строк.

Создание новой строки

o - пустая строка после текущей строки;

O - пустая строка перед текущей строкой.

Использование буфера обмена

Занести в буфер:

yw - сохранить слово (курсор - в начале слова);

yy - сохранить одну строку ;

5yy - сохранить 5 строк; и т.п.

При выполнении команд ndd (где n - число) удаляемые n строк заносятся в буфер.

Вставка текста из буфера:

- p - после текущей строки;
- P - перед ней.

Пример работы с текстовым редактором

Vi + myfile.txt

```
i
helo!
How are you?
```

```
i
helo!
hOW ARE YOU?
C$ JJJ
```

```
i
helo!
hOW ?
C$ JJJ
```

Попробуйте объяснить смысл каждый строки из данного примера. Какие операции запрашивал пользователь? Какие запросы он вводил с клавиатуры, чтобы получить тот или иной результат?

5.1 Методика выполнения

1.Войдите в редактор с созданием нового пустого файла с произвольным именем и расширением - .1.

2. Поместите в созданный файл текст, включающий не менее четырех строк с несколькими словами в каждой.

3. Вставьте по одной пустой строке до и после одной из строк файла.

4. Заполните пустые строки произвольным текстом.

5.Вставьте еще по одной строке в середину текста файла без предварительного резервирования пустых строк.

6. Перейдите в режим редактирования и выполните произвольное редактирование отдельных слов и строк файла с использованием всех команд из групп «Команды изменения текста» и «Команды отмены произвольных изменений в текущей строке».

7. Выйдите из редактора с сохранением файла. Убедитесь в сохранении созданного файла.

8. Войдите опять в VI для редактирования созданного файла с использованием команд из групп "Команды копирования в буфер" и "Команды вставки буфера в текст".

9. Поменяйте местами несколько слов в строках файла.

10. Поменяйте местами несколько строк.

11. Поменяйте местами последовательно начало строки с её концом и наоборот.

12. Выполните операцию поиска строк файла по заданным их фрагментам с различными направлениями поиска (см. «Команды поиска строки файла по фрагменту её текста»).

13. Не выходя из vi, перепишите полученную в результате редактирования версию файла в файл с тем же именем, но с расширением .new.

14. Добавьте первые 3-и строки редактируемого файла к файлу .1.

16. Не выходя из vi, загрузите в буфер файл с расширением .1.

17. Последовательно удалите части строк и несколько строк с использованием команд из групп «Команды удаления текста».

18. Выйдите из редактора без сохранения файла.

19. Просмотрите и проанализируйте содержимое редактируемых файлов в текущем каталоге.

20. Уничтожьте созданные файлы

5.2 Контрольные вопросы

1. В чем особенности и преимущества встроенного редактора vi ОС UBUNTU?

2. Какие два основных режима работы использует редактор? Как осуществляется переключение режимов?

3. Какую структуру имеет экран при редактировании файла? Назначение полей экрана?

4. Как в vi организованно редактирование открытого в нем файла?

5. Как организованна работа с клавишными командами редактора?

6. Что такое «префиксные команды» и их назначение? Как организована работа с командами этого вида?

7. Какие функции редактора вы использовали при выполнении лабораторной работы?

Лабораторная работа №6

Введение в shell-программирование.

Эта работа посвящена ознакомлению со средствами языка shell, для создания процедур обработки данных. Изучаются вопросы оформления shell-процедур.

Изучение команд:

`Set` – присваивание значения параметрам, передаваемым процедурам;

`Echo, read, banner` – вспомогательные команды для ввода и вывода информации;

`If, then, else` – команды проверки условий и ветвления вычисления в процедуре;

`Test` – проверка файлов, числовых величин, строк символов;

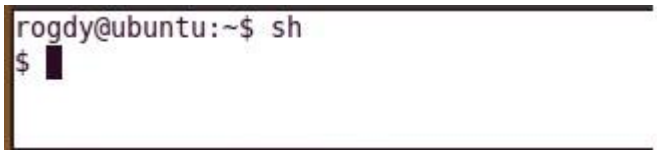
`While, until, for` – команды построения циклических процедур;

- правила построения и постановки значений переменных;
- вычисление арифметических выражений;
- обработка символьных строк.

Командный язык shell (в переводе - раковина, скорлупа) фактически есть язык программирования очень высокого уровня. На этом языке пользователь осуществляет управление компьютером. Обычно, после входа в систему вы начинаете взаимодействовать с командной оболочкой.

Программирование на языке Shell происходит в окне терминала, а сами операции выполняются сразу, как только нажимается клавиша Enter после введения текста функции. (В отличие, скажем, от языка C++, где для того, чтобы компьютер выполнил введённые вами функции, необходимо запускать написанную программу нажатием клавиши F9. В Shell такого нет, а значит, нужно быть внимательным при вводе функций и команд, т.к. в случае ошибки придётся переписывать всю программу заново).

Для того, чтобы, работая в терминале, переключиться в среду Shell, необходимо ввести команду sh. Признаком того, что оболочка (shell) готова к приему команд служит выдаваемый ею на экран промптер("$").



Имя shell-переменной - это начинающаяся с буквы последовательность букв, цифр и подчеркиваний.

Значение shell-переменной - строка символов.

Для присваивания значений переменным может использоваться оператор присваивания "=".

var_1=13 - "13" - это не число, а строка из двух цифр. (аналог из C++: char var_1[3] = "13")

var_2="ОС UBUNTU" - здесь двойные кавычки (" ") необходимы, так как в строке есть пробел.

ВАЖНО: Обратим внимание на то, что, как переменная, так и ее значение должны быть записаны без пробелов относительно символа "=".

Возможны и иные способы присваивания значений shell-переменным. Так например запись,

```
$ DAT=`date`
```

приводит к тому, что сначала выполняется команда "date" (обратные кавычки говорят о том, что сначала должна быть выполнена заключенная в них команда), а результат ее выполнения, вместо выдачи на стандартный выход, приписывается в качестве значения переменной, в данном случае "DAT".

```
$ echo $DAT
Mon Oct 17 16:46:28 MSD 2011
```

Можно присвоить значение переменной и с помощью команды "read", которая обеспечивает прием значения переменной с (клавиатуры) дисплея в диалоговом режиме (аналог из C++: scanf или cin). Обычно команде "read" в командном файле предшествует команда "echo", которая позволяет предварительно выдать какое-то сообщение на экран. Например:

```
$ echo -n "vvedite znachenie x"
vvedite znachenie x$ read x
5
$ echo $x
5
```

При выполнении этого фрагмента командного файла, после вывода на экран сообщения

```
Введите трехзначное число:
```

интерпретатор остановится и будет ждать ввода значения с клавиатуры. То число, которое пользователь введёт с клавиатуры, станет значением переменной "x".

Одна команда "read" может прочитать (присвоить) значения сразу для нескольких переменных. Если переменных в "read" больше, чем их введено (через пробелы), оставшимся присваивается пустая строка. Если передаваемых значений больше, чем переменных в команде "read", то лишние игнорируются.

ПРЕДУПРЕЖДЕНИЕ. На самом деле интерпретатор для продолжения работы ждет лишь нажатия клавиши. Введенное вами число воспринимается им не как число, а как последовательность символов(!). Интерпретатор не проверяет, что вы ввели. Поэтому в качестве значения переменной может оказаться любая введенная абракадабра или просто нажатие , как значение пустой строки. (Для обеспечения проверки формата ввода следует написать свою команду).

При обращении к shell-переменной необходимо перед именем ставить символ "$" для получения Значения этой переменной. Сравните две команды, по-разному набранные с клавиатуры и заметьте разницу в результатах, распечатываемых программой на экране.

```
$ echo x
x
$ echo $x
5
```

Запись echo x печатает исключительно те буквы, которые введены после команды echo. (Аналог в C: printf(“x”) и printf(“%d”, x)).

И еще один пример. Фрагмент командного файла:

```
$ echo x = $x
```

выдаст на экран

```
x = 5
```

Что наглядно показывает, что команда echo может за один раз выводить на экран несколько переменных.

Команда "set" устанавливает значения параметров. Это бывает очень удобно. Например, команда "date" выдает на экран текущую дату, скажем, "Mon Oct 13 16:46:28 2011", состоящую из пяти слов, тогда

```
$ set `date`
$ echo $1 $3
```

выдаст на экран

```
Mon 17
```

Команда "set" позволяет также осуществлять контроль выполнения программы, например:

set -v на терминал выводятся строки, читаемые shell.

Set +v отменяет предыдущий режим.

Set -x на терминал выводятся команды перед выполнением.

Set +x отменяет предыдущий режим.

Команда "set" без параметров выводит на терминал состояние программной среды

Как во всяком языке программирования в тексте на языке shell могут быть комментарии. Для этого используется символ "#". Все, что находится в строке (в командном файле) левее этого символа, воспринимается интерпретатором как комментарий. Например,

```
$ # ja kommentariy!
$ ## ja tozhe
$ ### i ja =P
```

Как во всяком процедурном языке программирования в языке shell есть операторы. Ряд операторов позволяет управлять последовательностью выполнения команд. В таких операторах часто необходима проверка условия, которая и определяет направление продолжения вычислений.

Команда test ("[ ]")

Команда test проверяет выполнение некоторого условия. С использованием этой (встроенной) команды формируются операторы выбора и цикла языка shell.

Два возможных формата команды:

```
test условие
```

или

```
[ условие ]
```

Пробелы должны быть и между значениями и символом сравнения или операции (как, кстати, и в команде "expr") . Про последнюю следует сказать подробнее. Именно команда expr даёт программе возможность воспринимать ту или иную переменную именно как число. (Иными

словами, если сравнивать два ЯП, Shell и C, запись echo `expr $x` будет аналогична записи printf("%d", x). Разница в том, что в языке Си тип данных (char, int и пр.) определяется и устанавливается сразу при объявлении переменной, а в shell тип данных определяется после того, как значение было записано в переменную и может меняться в теле программы).

Пример работы с командами и операторами языка shell.

```
$ echo date: `date`
date: Mon Oct 17 17:27:46 MSD 2011
$ s=`expr $6 - $3`
$ echo $s
1994
```

Попробуйте объяснить смысл каждой строки из данного примера. Какие операции запрашивает пользователь, и какие ответы даёт система?

В shell используются условия различных "типов".

Условный оператор "if"

В общем случае оператор "if" имеет структуру

```
if  [условие] ; echo $0
  then  список
     [elif  условие
      then  список]
  [else  список]
Fi
```

Здесь "elif" сокращенный вариант от "else if" может быть использован наряду с полным, т.е. допускается вложение произвольного числа операторов "if" (как и других операторов). Разумеется "список" в каждом случае должен быть осмысленный и допустимый в данном контексте.

Конструкции

```
[elif  условие
 then  список]
```

и

```
[else  список]
```

не являются обязательными (в данном случае для указания на необязательность конструкций использованы квадратные скобки - не путать с квадратными скобками команды "test"!).

пример использования оператора if:

```
$ z=7
$ if [  expr $z > 0`] ; echo $0
> then "POLOZITELNO"
> [ elif `expr &z<0` then echo "otrizatelno"]
> [else echo "null"]
> fi
sh
sh: POLOZITELNO:
```

Самая усеченная структура этого оператора

```
if   условие
  then  список
fi
```

Обратите внимание, что структура обязательно завершается служебным словом "fi". Число "fi", естественно, всегда должно соответствовать числу "if".

Примеры.

Пусть написан расчет "if-1"

```
$ if [ $1 -gt $2]
> then pwd
> else echo $0: Hello!
> fi
```

Тогда вызов расчета

```
if-1 12 11
```

даст

```
/home/sae/STUDY/SHELL,
```

а

```
if-1 12 13
```

даст

```
if-1 : Hello!
```

Оператор вызова ("case")

Оператор выбора "case" имеет структуру:

```
case   строка  in
   шаблон)  список команд;;
   шаблон)  список команд;;
        ...
esac
```

Здесь "case" "in" и "esac" - служебные слова. "Строка" (это может быть и один символ) сравнивается с "шаблоном". Затем выполняется "список команд" выбранной строки. Непривычным будет служебное слово "esac", но оно необходимо для завершения структуры.

Пример.

```
###
# case-1: Структура "case".
#         Уже рассматривавшийся в связи со
#         структурой "if" пример проще и
#         нагляднее можно реализовать с
#         помощью структуры "case".
echo -n " А какую оценку получил на экзамене?: "
read z
case $z in
   5) echo Молодец !            ;;
   4) echo Все равно молодец !  ;;
   3) echo Все равно !          ;;
   2) echo Все !                ;;
   *) echo  !                   ;;
```

```
esac
```

Непривычно выглядят в конце строк выбора ";;", но написать здесь ";" было бы ошибкой. Для каждой альтернативы может быть выполнено несколько команд. Если эти команды будут записаны в одну строку, то символ ";" будет использоваться как разделитель команд.

Обычно последняя строка выбора имеет шаблон "*", что в структуре "case" означает "любое значение". Эта строка выбирается, если не произошло совпадение значения переменной (здесь $z) ни с одним из ранее записанных шаблонов, ограниченных скобкой ")". Значения просматриваются в порядке записи.

```
###
# case-2:  Справочник.
#          Для различных фирм по имени выдается
#          название холдинга, в который она входит
case $1 in
        ONE|TWO|THREE) echo Холдинг: ZERO    ;;
             MMM|WWW) echo Холдинг: Not-Net ;;
    Hi|Hello|Howdoing) echo Холдинг: Привет! ;;
                  *) echo Нет такой фирмы  ;;
esac
```

Оператор цикла с перечислением ("for")

Оператор цикла "for" имеет структуру:

```
for имя in список значений
    do
        список команд
    done
```

где "for" - служебное слово определяющее тип цикла, "do" и "done" - служебные слова, выделяющие тело цикла. Не забывайте про "done"! Фрагмент "in список значений" может отсутствовать.

Пример работы с оператором done:

```
$ a=1
$ b=2
$ c=3
$ for i in $a $b $c
> do
> echo "I love you!"
> done
I love you!
I love you!
I love you!
```

Т.о. мы видим, что переменная i в цикле пробегает по всем значением, заданным в условии (т.е. по очереди принимает значения всех переменных, указанных в условии), и для каждого из них выполняет ту операцию, которая заказана в теле цикла.

Оператор цикла с истинным условием ("while")

Структура "while", также обеспечивающая выполнение расчетов, предпочтительнее тогда, когда неизвестен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле.

Оператор цикла "while" имеет структуру:

```
while условие
    do
        список команд
    done
```

где "while" - служебное слово определяющее тип цикла с истинным условием. Список команд в теле цикла (между "do" и "done") повторяется до тех пор, пока сохраняется истинность условия (т.е. код завершения последней команды в теле цикла равен "0") или цикл не будет прерван изнутри специальными командами ("break", "continue" или "exit"). При первом входе в цикл условие должно выполняться.

```
$ a=1
$ b=5
$ while [ $a -lt $b]
> do
> a=`expr $a +1`
> echo $a
> done
```

```
a=`expr $a + 1`
```

т.е. при каждой итерации значение "a" увеличивается на 1.

Оператор цикла с ложным условием ("until")

Оператор цикла "until" имеет структуру:

```
until условие
    do
        список команд
    done
```

где "until" - служебное слово определяющее тип цикла с ложным условием. Список команд в теле цикла (между "do" и "done") повторяется до тех пор, пока сохраняется ложность условия или цикл не будет прерван изнутри специальными командами ("break", "continue" или "exit"). При первом входе в цикл условие не должно выполняться.

Отличие от оператора "while" состоит в том, что условие цикла проверяется на ложность (на ненулевой код завершения последней команды тела цикла) проверяется ПОСЛЕ каждого (в том числе и первого!) выполнения команд тела цикла.

Примеры.

```
$ until false
> do
> read x
> if [$x=5]; echo $0
> then echo enough; break
> else echo some more
> fi
> done
```

Здесь программа с бесконечным циклом ждет ввода слов (повторяя на экране фразу "some more"), пока не будет введено "5". После этого выдается "enough" и команда "break" прекращает выполнение цикла.

6.1 Методика выполнения.

1. Разработайте текст процедуры с использованием по заданию (см. ниже), вариант задания назначается преподавателем.
2. Отладьте, при необходимости отредактируйте и выполните процедуру.
3. Оформите процедуру с использованием вспомогательных команд и комментариев так, чтобы она легко читалась и чтобы результаты её работы легко анализировались.

6.2 Контрольные вопросы.

1. Что такое shell-процедура? Назначение?
2. Какого типа команды могут быть включены в тело процедуры?
3. Чем отличается обработка процедуры при выполнении от обработки программы на языке высокого уровня?
4. Что такое параметры? Для каких целей они используются? Какое число параметров может быть передано процедуре?
5. Какие вспомогательные команды вы использовали при оформлении процедуры?
6. Какого вида значения и как могут быть присвоены переменным языка shell?
7. Что такое локальные переменные и для каких целей их надо экспортировать в среду?
8. Как осуществляется ветвление вычислительного процесса процедуры?
9. Какого типа цикла в процедурах могут быть построены средствами языка shell?
10. Какие способы вызова процедур на исполнение вы знаете?

6.3 Варианты заданий к лабораторной работе № 6 «Введение в shell-программирование»

Разработать shell-процедуру с комментариями, выполняющую ниже перечисленные функции.

1. Вводит последовательность из N слов и подсчитывает в каждом введенном слове число символов. Если число символов больше M, то слово выводится на экран. Значения N и M передаются в качестве параметров.
2. Вводит строку из заданного числа слов. Выделяет слова, начинающиеся на указанную параметром букву, подсчитывает число таких слов.
3. Вводит строку N слов, анализирует длину каждого слова, упорядочивает слова по их алфавиту и выводит список на экран. Значение N задается параметром.
4. Вводит заданное параметром число слов и выводит каждое слово на печать, сопровождая его порядковым номером.
5. Вводит произвольное число коротких символьных параметров, подсчитывает длину каждого из них и выводит на экран список значений длин и общее число введенных параметров.
6. Вводит несколько коротких чисел в виде параметров, подсчитывает их сумму и результат выводит на экран.
7. Запрашивает последовательно ввод нескольких чисел со знаками и выводит на экран два списка чисел – положительных и отрицательных.
8. Запрашивает ввод строки символов, разделенных пробелами и заданной параметром длины, разбивает символы на пересекающиеся пары и выводит их на экран.
9. Ищет в личном головном каталоге пользователя созданные им файлы, выводит список их имен и распечатывается текст файла, заданного пользователем.
10. Создает новый подкаталог и помещает туда новые файлы, создаваемые пользователем по запросам процедуры. Имена новых файлов указываются параметрами.

11. Создает новый подкаталог и копирует туда из родительского каталога файлы заданного параметром типа.
12. Анализирует указанный параметром каталога и выводит на экран число файлов различного типа (обычные, директория, скрытые). Тип задается параметром.

Лабораторная работа №7.
Управление процессами.

Работа преследует цель - закрепить представление о возможностях командного языка UBUNTU по управлению процессами, которым выделяются все необходимые ресурсы вычислительной системы.

Изучаются команды:

`ps` - запрос информации о процессах текущего терминала;

`&`- запуск фонового процесса;

`fg, bg` – переводит процесс в активный или фоновый режим;

`jobs` - запрос листинга списка заданий;

`nohup` - защита фоновых процессов от прерывания выполнения при выходе из сеанса работы с системой;

`nice` - понижение приоритета процесса;

`kill` - прекращение выполнения процесса.

Процессом называется программа в стадии выполнения. Управление процессами — задача ядра операционной системы. Для поддержания информации о процессах ядро использует специальную структуру — таблицу процессов. Каждому процессу назначается уникальное число-идентификатор процесса, которое используется ядром для однозначного определения того, какому процессу соответствует запись в таблице процессов.

В `ОС UBUNTU` между процессами поддерживаются отношения "родитель"-"потомок". Каждый процесс может породить произвольное число процессов-потомков, но каждый процесс имеет только одного родителя, и нет ни одного процесса, который не имел бы родителя. Самый первый процесс, запускаемый ядром в ходе раскрутки системы, в системе называется init и служит для управления процессом раскрутки системы, в том числе, запуска процессов-демонов. Демоны, также известные как системные сервисы, — это процессы, производящие неинтерактивную обработку данных пользователя, то есть процесс-демон не подключен к терминальному устройству. Примером процессов-демонов могут служить серверы протоколов HTTP и FTP `httpd` и `ftpd`, соответственно, или сервер системного журнала `syslogd`. Часто имена исполняемых файлов программ-демонов заканчиваются буквой d. Процессу init назначается идентификатор 1,

первому порожденному init процессу, 2, и так далее. В ОС UBUNTU для идентификатора процесса общеупотребительным является сокращение PID.

После того, как процесс порожден, то есть ему отведена запись в системной таблице процессов, он находится в состоянии готовности, и может быть выбран планировщиком процессов операционной системы для выполнения. При выполнении операций ввода вывода, и в ряде других случаев, процесс может перейти в состояние ожидания некоторого события; тогда до наступления этого события, или поступления в процесс некоторого прерывания, он не будет выбираться для выполнения. Кроме процессов, находящихся в состояниях готовности и ожидания, в UBUNTU могут присутствовать остановленные процессы, и процессы-зомби. Останов и возобновление выполнения процессов мы обсудим позднее в ходе сегодняшнего занятия, а процессов-зомби коснемся в ходе рассмотрения системных вызовов ОС UBUNTU.

Для вывода списка запущенных процессов:

s

Команда ps принимает следующие ключи:

-a — вывод информации о процессах, запущенных от имени всех пользователей, а не только текущего;

-x — вывод информации о процессах, не подключенных к терминальным устройствам, например, о процессах демонах;

-u — вывод информации о ресурсах, используемых процессами;

-m — сортировка списка по объему используемой процессам оперативной памяти;

-r — сортировка списка по используемому процессом процессорному времени;

-U <имя пользователя> — вывод процессов, запущенных от имени указанного пользователя;

-p <список идентификаторов процессов> — вывод информации только об указанных процессах; идентификаторы в списке разделяются запятой.

Выводимый список включает несколько столбцов; эти столбцы содержат следующую информацию:

USER — имя пользователя, запустившего процесс;

PID — идентификатор процесса;

%CPU — процент процессорного времени, потребляемый процессом;

%MEM — процент объема памяти, занятой процессом;

COMMAND — команда, использованная для запуска процесса;

TT — терминальное устройство, к которому подключен процесс;

VSZ — объем виртуальной памяти, затребованный процессом, в блоках по 1024 байта;

RSZ — реальный объем памяти, используемый процессом, в блоках по 1024 байта;

START — время запуска процесса.

STAT — информация о состоянии процесса; первая буква означает состояние процесса:

S — процесс находится в состоянии ожидания менее 20 секунд;

I — процесс находится в состоянии ожидания 20 секунд, или более; К — процесс находится в состоянии готовности;

T — процесс остановлен;

Z —процесс-зомби.

За первой буквой может также быть указана дополнительная информация.

Например, команда:

```
ps  -u  -r  -x
```

может быть использована для вывода подробной информации обо всех процессах, запущенных от имени текущего пользователя, причем список процессов будет упорядочен по степени использования процессорного времени.

Для наблюдения за процессами системы:

```
top
```

Программа top выводит таблицу, в которой каждому процессу соответствует одна строка. Сортировкой строк можно управлять, *а по умолчанию критерием для сортировки является идентификатор процесса.* В отличие от программ, с которыми мы имели дело до этого, top не возвращает управление пользователю сразу после печати результата. Вместо этого, через некоторые заданные промежутки времени программа собирает информацию о процессах системы и снова выводит ее на экран. Для выхода из программы следует нажать на клавишу [Q].

Программа принимает следующие параметры:

-o<параметр> — указывает показатель для сортировки строк с информацией о процессах; по умолчанию сортировка производится по убыванию; для того, чтобы сортировка проводилась по возрастанию, перед названием показателя следует поместить знак ’+ ‘ не отделяя его пробелами ни от ключа, ни от названия показателя; среди показателей наиболее интересны следующие:

command — команда, которой был запущен процесс;

cpu — степень использования процессорного времени;

pid — идентификатор;

time — время выполнения процесса;

rsize — реальный используемый процессом объем памяти;

vsize — объем виртуальной памяти, используемый процессом;

-s <число> — указывает задержку в секундах, между обновлением информации в таблице; по умолчанию 1 секунда;

-n <число> — указывает число процессов, о которых будет выведена информация.

Например, следующая команда:

```
top  -n  10  -s  3  -o  cpu
```

выведет таблицу из не более, чем 10 строк, в которой процессы будут отсортированы по убыванию используемого ими процессорного времени, причем обновляться таблица будет каждые 3 секунды.

Заголовок таблицы содержит следующие обозначения столбцов для показателей:

`PID` — идентификатор процесса;

`TIME` — процессорное время, потребленное процессом с момента запуска;

`%CPU` — процент процессорного времени, потребляемого процессом;

`USERNAME` — пользователь, от имени которого выполняется процесс.

Одним из способов взаимодействия между процессами в UBUNTU являются сигналы; сигналы — это сообщения, которые могут быть посланы процессу при помощи системного вызова. Каждому виду сигналов в системе соответствует название и номер. Многие сигналы имеют специальный смысл. В частности, сигнал `SIGTERM` уведомляет процесс о необходимости завершения, сигнал `SIGTERM` используется для безусловного завершения процесса, сигналы `SIGSTOP` и `SIGCONT`, соответственно, останавливают и возобновляют выполнение процесса, и так далее.

Для того чтобы послать сигнал процессу из программы-оболочки:

```
kill [-s <название сигнала>] <список идентификаторов  процессов>
```

Например, чтобы остановить выполнение некоторого процесса, с идентификатором 123, можно воспользоваться следующей командой:

```
rogdy@ubuntu:~$ kill -s SIGSTOP 123
```

А для его возобновления, можно применить такую команду:

```
rogdy@ubuntu:~$ kill -s SIGCONT 123
```

Если при вызове kill не указать название сигнала, то будет послан сигнал `SIGTERM`. Так, для завершения процесса с идентификатором 123, можно использовать команду:

```
rogdy@ubuntu:~$ kill 123
```

Для вывода всех сигналов, поддерживаемых системой, служит ключ `-l`:

```
rogdy@ubuntu:~$ kill -1
```

Для ожидания в течение указанного числа секунд:

```
sleep <время ожидания>
```

Рассмотрим следующий пример файла сценария:

```
rogdy@ubuntu:~$ while  [ [ -n "true" ] ]
> do
> i=1
> while [ [ "$i" -lt 100000 ] ]
> do
> let i=$i+1
> done
> #sleep for 3 seconds
> sleep 3
> done
```

Приведенный процесс, порожденный приведенным примером, будет попеременно находиться в

состоянии готовности и выполнения — последовательное изменение значения переменной i с 1 до 9999, — и состоянии ожидания — вызов `sleep` с параметром 3. Процесс никогда не завершится самостоятельно, так как условие внешнего цикла `while` всегда истинно — строка `"true"` не пуста.

Далее будем считать, что сценарий сохранен в файле `loop.sh`.

Программа-оболочка позволяет запускать программы в фоновом режиме. Сразу после запуска программы, указанной в команде запуска в фоновом режиме, пользователь снова получает возможность вводить новые команды в программе-оболочке, в то время как запущенная программа продолжает выполняться.

Для запуска программы в фоновом режиме команда завершается символом амперсанда: '&'. Например, команда:

```
. /loop.sh  &
```

запустит на выполнение вышеприведенный файл сценария `loop.sh` в фоновом режиме.

А следующая последовательность команд:

```
./ loop.sh  &
ps
```

выведет информацию о запущенных процессах сразу после запуска сценария `loop.sh`, не дожидаясь завершения его выполнения.

После запуска процесса, программа-оболочка напечатает его PID. Другим способом получить PID последнего процесса, запущенного в фоновом режиме, обращение к специальной переменной окружения! Этот способ удобно применять в файлах сценариев программы-оболочки.

Следующий файл сценария запускает `loop.sh`, выводит информацию об этом процессе, а затем завершает его.

```
#!/bin/sh
./ loop.sh   &
loop_pid=$ !
echo  started  loop.sh
ps  -u  -p  $loop_pid
sleep  3
echo  terminating loop.sh
ps –u –p $loop_pid
kill $ loop_pid
echo  terminated loop.sh
ps –u –p $loop_pid
```

Обратите внимание, что последний вызов ps не выводит на экран информацию ни об одном процессе, так как к этому моменту, процесс, выполняющий сценарий `loop.sh`, уже завершен.

Протяженность времени, отводимого планировщиком операционной системы процессу, зависит от приоритета процесса. Чем выше приоритет процесса, тем большую часть процессорного времени он будет получать во время выполнения. Приоритет процесса выражается числом. Чем меньше число, тем выше приоритет. По умолчанию, процессы запускаются с приоритетом породившего их процесса.

Для запуска процесса со значением приоритета, отличного от приоритета процесса-предка:

```
nice [-n <число> ] <команда>
```

При запуске процесса, значение приоритета будет откорректировано, исходя из переданного числа. Если ключ -n не был введен в команде, то используется число 10. Чем выше число, тем меньший приоритет получит процесс. Число, передаваемое в составе ключа -n может лежать в пределах от —20 до 20, но пользователям системы, не обладающим администраторскими полномочиями, как правило, не позволяется задавать отрицательные значения. То есть, рядовой пользователь не может повысить приоритет процесса.

Для изменения приоритета после запуска процесса:

```
renice -n <число> <список идентификаторов процессов>
```

Программа `renice` работает аналогично `nice`, с той разницей, что изменяется не приоритет создаваемого процесса, а приоритеты всех запущенных процессов, идентификаторы которых входят в разделенный пробелами список, передаваемый команде в качестве параметра. Число, в составе ключа -n прибавляется к текущему приоритету процесса. Например, в следующем примере:

```
rogdy@ubuntu:~$ nice -n 5 ./loop.sh &
[1] 4549
renice -n 5 $!
```

приоритет процесса, выполняющего сценарий `loop.sh` устанавливается сначала 5, а затем 10.

При вводе команды в программе-оболочке, вне зависимости о того, запускается ли программа в фоновом режиме, или нет, программа-оболочка порождает дочерний процесс. Для исполнения программы самим процессом программы-оболочки:

```
exec <команда>
```

Рассмотрим следующий сценарий:

```
#!/bin/sh
ps
exec ./loop.sp
```

Пусть, он сохранен под именем `exec.sh`.

```
./exec.sh &
ps
```

Первая команда запускает на выполнение сценарий `exec. sh`, что приводит к порождению нового процесса. В этом можно убедиться, изучив результат вызова `ps`, который производится

этим сценарий. После этого сценарий запускает уже рассмотренный нами сценарий `loop.sh`. Вторая команда из приведенной выше пары вызывает `ps` для выдачи списка запущенных процессов. Обратите внимание на то, что нового процесса для выполнения сценария `loop.sh` создано не было — его выполняет тот же процесс, который был порожден для выполнения сценария `exec.sh`.

В качестве еще одного примера, рассмотрим следующую команду:

```
exec  ls
```

Результат ее выполнения — завершение работы программы-оболочки: так как процесс, выполнявший код программы-оболочки получил инструкцию выполнять код программы `ls`, то завершение кода программы `ls`, наступающее после вывода листинга текущего каталога, приведет к завершению выполняющего ее процесса. Таким образом, процесс, изначально выполнявший код программы-оболочки, будет завершен.

7.1 Методика выполнения.

1. Вывести на экран листинг характеристик (в длинном и коротком форматах) процессов, инициализированных с вашего терминала. Проанализировать и объяснить содержание каждого поля сообщения.
2. Разработать и запустить простейшую процедуру в фоновом режиме с бесконечным циклом выполнения, предусматривающую, например, перенаправление вывода каких то сообщений в файл или в фиктивный файл, и использующую команду sleep для сокращения частоты циклов процедур.
3. Выполнить п.1. Объяснить измерения в листинге характеристик процессов. Объясните содержание PID и PPID.
4. Понизите значение приоритета процедуры. На что и как повлияет эта операция при управлении вычислительным процессом системы? Как отразятся её результаты в описателях процессов?
5. Проанализируйте листинг процессов. Какой процесс является родительским для процедуры.
6. Выйдите из системы и войдите заново. Проанализируйте листинг процессов. Объясните изменения в системе.
7. Запустите процедуру в фоновом режиме, но предусмотрите её защиту от прерывания при выходе из системы.
8. Выполните п.6. Объясните изменения PPID процедуры.
9. Завершите выполнение процесса процедуры.
10. Запустите процедуру в оперативном режиме с перенаправлением вывода в соответствующий файл.
11. Переведите задание с процедурой в фоновом режиме и проанализируйте сообщение на экран.
12. Переведите задание с процедурой в оперативном режиме и проанализируете сообщение на экран.

13. Завершите выполнение процедуры и проанализируйте сообщение на экран.

14. Проанализируйте с использованием команды history содержание лабораторной работы, продумайте ответы на нижеприведенные контрольные вопросы и сдайте выполненную работу преподавателю. После получения зачета по работе – уничтожьте все созданные файлы и корректно выйдите из системы.

7.2. Контрольные вопросы

1. Объясните понятия процесса и ресурса. Какое их значение в организации вычислительного процесса в ОС UBUNTU?
2. Какая информация содержится в описателях процессов? Как просмотреть их содержание в процессе работы с системой?
3. Какими способами можно организовать выполнение программ в фоновом режиме?
4. Какие особенности выполнения программ в фоновом режиме? Как избежать вывода фоновых сообщений на экран и прерывания выполнения фоновых программ при прекращении сеанса работы с системой?
5. Как пользователь может повлиять на распределение ресурсов между активными процессами?
6. Как можно прервать выполнение активных процессов? Какая информация для этого необходима и откуда она извлекается?

Лабораторная работа №8
Программирование shell-процедур.

Работа предусматривается выполнение индивидуального задания повышенной сложности. Работа предусматривает несколько выходов на машину для отладки процедуры.

Отчетом по работе является работающая процедура, продемонстрированная преподавателю с объяснениями её текста и алгоритма работы. Выполнение лабораторной работы в полном объеме является обязательным условием для получения по курсу в целом экзаменационной оценки – «отлично».

Индивидуальные задания к лабораторной работе

Вариант 1.

Написать shell-процедуру, которая:

- вводит передаваемое в качестве 1-го параметра количество символьных строк;

- в каждой введенной строке ищет подстроку, передаваемую в качестве второго параметра;
- заменяет каждую найденную подстроку на строку, передаваемую в качестве третьего параметра;
- выводит на экран каждую введенную строку и соответствующую ей новую строку.

Вариант 2.

Написать shell-процедуру, которая:

- вводит 2 символьные строки;
- в каждой введенной строке ищет подстроку, передаваемую в качестве параметра;
- заменяет каждую найденную подстроку на пробел;
- образует из полученных строк третью строку так, чтобы в ней чередовались слова из первой и второй строк;
- выводит на экран введенные строки и новую строку.

Вариант 3.

Написать shell-процедуру, которая:

- вводит символьную строку;
- во введенной строке ищет подстроку, передаваемую в качестве первого параметра;
- вставляет после каждой найденной подстроки символ, передаваемый в качестве второго параметра;
- удаляет из полученной строки символ, передаваемый в качестве третьего параметра;
- выводит на экран введенную и новую строку.

Вариант 4.

Написать shell-процедуру, которая:

- вводит символьную строку;
- проверяет введенную строку на совпадение со строкой, переданной в качестве 1-го параметра;
- если строки совпадают, то выдает на экран приглашение повторить ввод;
- если не совпадают, то сравнивает длину введенной строки с длиной 2-го параметра, и, в случае их равенства, выводит на экран введенную строку в обратном порядке составляющих ее символов;

Вариант 5.

Написать shell-процедуру, которая:

- вводит символьную строку;
- проверяет введенную строку на совпадение со строками, содержащимися в файле, имя которого передается в качестве 1-го параметра;
- для всех найденных совпадений заменяет соответствующие строки в файле на строку, переданную в качестве 2-го параметра;
- выводит на экран старое и новое содержимое файла, а также число найденных совпадений;

Вариант 6.

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую маршрутное имя некоторого файла; проверяет введенное маршрутное имя, если оно начинается с символа /, на совпадение его первой части с маршрутным именем домашнего каталога пользователя;
- если введенное маршрутное имя содержит маршрутное имя домашнего каталога или является относительным, то проверяет существование указанного файла, в противном случае выводит на экран сообщение об ошибке;
- если файл существует, то выводит на экран его содержимое;
- если файл не существует, то создает его и записывает в него строку, передаваемую в качестве параметра;

Вариант 7.

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую имя некоторого файла;
- проверяет наличие файла в домашнем каталоге или в одном из подкаталогов;
- если файл существует, то выводит на экран его содержимое;
- если файл не существует, то создает его и записывает в него с консоли некоторый текст;
- устанавливает для файла права доступа, передаваемые в качестве параметра.

Вариант 8.

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую текст некоторого сообщения;
- проверяет наличие в своем почтовом ящике такого же сообщения;
- если в почтовом ящике имеется введенное сообщение, то выводит его на экран и посылает на терминалы всем пользователям, в данный момент работающим в системе из числа тех, чьи имена передаются в качестве параметров;
- всем остальным пользователям, чьи имена передаются в качестве параметров, рассылает введенное сообщение по почте;

Вариант 9.

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую текст некоторого сообщения;
- проверяет регистрацию в системе пользователей, чьи имена переданы вторым и последующими параметрами;
- всем пользователям, чьи имена передаются в качестве второго и следующих параметров и работающим в системе в течение заданного первым параметром времени, рассылает введенное сообщение по почте;
- всем остальным пользователям, работающим в данный момент в системе, рассылает прямые сообщения, содержащие введенную символьную строку.

Вариант 10.

Написать shell-процедуру, которая:

- проверяет свой почтовый ящик на наличие в нем сообщений;
- находит в почтовом ящике одинаковые по тексту сообщения;
- всем пользователям, приславшим более одного сообщения с одинаковым текстом, рассылает по почте сообщения, текст которого содержится в файле, имя которого передается в качестве параметра.