

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
ГРАЖДАНСКОЙ АВИАЦИИ

---

А. В. Столяров

АРХИТЕКТУРА ЭВМ И  
СИСТЕМНОЕ ПРОГРАММНОЕ  
ОБЕСПЕЧЕНИЕ

ЗАДАНИЯ ЛАБОРАТОРНЫХ РАБОТ

*для студентов II и III курсов  
специальности 230401  
дневного обучения*

Москва – 2009

ББК 518  
С81

**Столяров А. В.**

**Архитектура ЭВМ и системное программное обеспечение: задания лабораторных работ — М.: 2009.— с.**  
С81

# Часть 1: III семестр

## Лабораторная работа 1.1: ознакомительная

**Задача работы:** ознакомление с операционной средой ОС Unix.

**Необходимо знать:** принципы взаимодействия с командной строкой ОС Unix (см. пособие [1], § 2).

**Задание:**

- Получить у преподавателя входное имя и пароль для входа в систему, если это не сделано раньше. Войти в систему, сменить пароль.
- Воспользовавшись командой `pwd`, узнать полное имя своей домашней директории.
- Пользуясь командами `cd` и `ls`, просмотреть содержимое корневой директории, директории `/etc` и директории `/usr/local`.
- Пользуясь командами `mkdir`, `touch`, `mv`, `rm` и `rmdir`, создать в домашней директории поддиректорию, войти в неё, создать в ней пустой файл, переименовать этот файл; убедиться с помощью команды `ls`, что все действия выполнены удачно; удалить файл, удалить поддиректорию.
- Пользуясь (по своему выбору) редактором текстов `joe`, `vim` или встроенным редактором оболочки Midnight Commander, создать текстовый файл, содержащий текст на английском языке (не менее 10 строк), например, какое-либо английское стихотворение, текст песни и т.п.
- Распечатать содержимое файла, пользуясь командой `cat`.
- Пользуясь командой `wc`, узнать количество строк, слов и символов в файле.
- Пользуясь перенаправлением вывода и командой `ls`, создать в домашней директории файл, содержащий список имён файлов, находящихся там же в домашней директории.
- Пользуясь командами `ls`, `wc` и конвейером, определить количество файлов в директории `/usr/bin`.
- Убедившись в том, что работа всех команд вам понятна, заявить о своей готовности и ответить на вопросы преподавателя.

## Лабораторная работа 1.2: ассемблер `nasm`

**Задача работы:** освоить процедуру компиляции и сборки программ, написанных на ассемблере `nasm`.

**Необходимые знания:** параметры вызова ассемблера `nasm` и системного компоновщика `ld`; пример программы на языке ассемблера, приведённый на лекции.

### Задание:

- Пользуясь любым (по своему выбору) редактором текстов, подходящим для написания программ (см. пособие [1], § 2.3), создать файл с суффиксом `.asm`, содержащий текст примера программы, приводившийся на соответствующей лекции. При написании текста необходимо строго соблюдать правила оформления программ на языке ассемблера:
  - директивы ассемблера и метки располагать в крайней левой позиции строки;
  - команды, псевдокоманды и макровыводы располагать в позиции первой табуляции, применяя для этого символ табуляции (никаких пробелов в начале строки, только табуляции!)
  - метки длиной шесть и более символов располагать на отдельных строках.
- Пользуясь конспектом лекций, повторить, что означают директивы и команды, составляющие программу; убедиться, что всё понятно.
- Пользуясь ассемблером `nasm`, оттранслировать файл. При необходимости — исправить ошибки. С помощью команды `ls` убедиться, что файл объектного модуля (файл с суффиксом `.o`) успешно создан.
- Пользуясь системным компоновщиком `ld`, создать исполняемый файл; убедиться, что он создан.
- Запустить исполняемый файл на выполнение. Убедиться, что результаты соответствуют ожидавшимся; в противном случае выяснить причину несоответствия.
- Внести в программу какие-либо изменения; повторить процедуру получения исполняемого файла и запуска его на выполнение. Убедиться, что результаты соответствуют ожидавшимся.

- Убедившись в том, что исходный текст примера, а также смысл действий, предпринимаемых для запуска программы, полностью понятен, заявить о своей готовности и ответить на вопросы преподавателя.

**Лабораторные работы 1.3–1.6 представляют собой единую серию заданий, каждое следующее из которых предполагает модификацию программы, написанной при выполнении предыдущего**

## **Лабораторная работа 1.3: чтение числа**

**Задача работы:** освоить макросы упрощенного ввода-вывода, регистры процессора, арифметические операции и команды условных переходов.

**Необходимые знания:** макросы `getchar`, `putchar` и `exitprogram`, команды `mov`, `add`, `mul`, `cmp`, `inc`, `dec`, `je`, `jne`, `jmp`, `loop`, псевдокоманды `db` и `dd`, директивы `%include` и `section` (см. конспекты соответствующих лекций)

**Задание:** написать на языке ассемблера `nasm` программу, читающую из потока стандартного ввода (с клавиатуры) целое число в десятичной системе счисления и по окончании ввода числа выдающую в стандартный поток вывода (на экран) столько символов `*`, какое число было введено.

**Указание:** Представление числа необходимо читать посимвольно. Коды цифр `'0'`, `'1'`, ..., `'9'` представляют собой последовательные числа, так что, чтобы получить из кода цифры её числовое значение, достаточно из прочитанного с клавиатуры кода вычесть код символа `'0'`, предварительно убедившись, что считанный код является кодом цифры (то есть больше либо равен коду символа `'0'` и меньше либо равен коду символа `'9'`). Вычисление числа, соответствующего введённому пользователем строковому представлению, производится по следующему алгоритму:

- Заводим счётчик, в котором будем накапливать число. Поскольку регистров не так много и они почти все используются командой `mul`, счётчик следует завести в памяти. Заносим в счётчик значение 0 (не путать с символом `'0'!`)

- При прочтении очередного символа, если этот символ **не** является цифрой, заканчиваем работу, считая, что прочитано число, которое к этому моменту находится в счётчике.
- Если же очередной символ является цифрой, то умножаем число, находящееся в счётчике, на 10, прибавляем к нему числовое значение прочитанной цифры и результат помещаем обратно в счётчик.

Например, если пользователь введёт число 275, то в счётчике сначала будет 0, затем  $0 \times 10 + 2 = 2$ , на следующем шаге  $2 \times 10 + 7 = 27$  и, наконец,  $27 \times 10 + 5 = 275$ , что и требовалось.

## Лабораторная работа 1.4: печать числа

**Задача работы:** освоить работу со строками и вывод чисел

**Необходимые знания:** команда `div`

**Задание:** воспользовавшись текстом программы, написанной в ходе предыдущей лабораторной работы, написать программу, которая читает из стандартного потока ввода (с клавиатуры) два числа, разделённых символом пробела, и, в зависимости от индивидуального варианта, выдаёт в стандартный поток вывода (на экран) их сумму, разность или произведение. Предусмотреть выдачу сообщений об ошибках в случае некорректного пользовательского ввода.

**Указание:** целое число, уместяющееся в 32-битный регистр, не может в десятичном представлении занимать больше десяти цифр. Таким образом, для временного хранения строкового представления результата нужен буфер, достаточная длина которого 11 байт.

Ввод чисел и перевод их из строкового представления в числовое производится так, как это описано в указаниях к предыдущей работе. Результат (сумму, произведение или разность, в зависимости от индивидуального варианта) следует занести в специально для этого отведённую память.

Перевод числа в строковое представление производится пошагово из конца в начало. На каждом шаге число делится (с остатком) на 10. Результат деления (частное) заносится обратно в память, хранящую число, что касается остатка, то он представляет собой числовое значение очередной цифры. Чтобы получить код цифры, необходимо к этому значению прибавить код символа '0'. Полученный код символа заносится в соответствующую позицию в строковом буфере: начать следует с последней позиции, затем уменьшать текущий адрес (например, хранящийся в регистре EDI) на каждом шаге на единицу.

Алгоритм перевода заканчивается, когда на очередном шаге частное оказывается равно нулю. После этого можно вывести содержимое буфера на печать.

## Лабораторная работа 1.5: подпрограммы и стек

**Задача работы:** освоить оформление подпрограмм, передачу параметров через стек, создание стековых фреймов и управление ими, трансляцию многомодульных программ.

**Необходимые знания:** общие принципы работы со стеком, команды `push`, `pop`, `call`, `ret`, структура и принципы организации стекового фрейма при вызове подпрограммы, конвенция вызовов подпрограмм языка Си, директивы `global` и `extern`, правила раздельной трансляции модулей и сборки программы, состоящей из нескольких модулей (см. конспекты соответствующих лекций).

**Задание:** оформить в виде подпрограмм фрагменты кода, переводящие число в строку и строку в число, написанные в ходе двух предыдущих лабораторных работ; все необходимые локальные переменные хранить в стековом фрейме. Переписать программу, написанную в ходе предыдущей работы, вынеся каждую из полученных подпрограмм в отдельный модуль (получив, таким образом, три модуля), оттранслировать всю программу в соответствии с правилами трансляции многомодульных программ.

## Лабораторная работа 1.6: системные вызовы

**Задача работы:** освоить прямую работу с системными вызовами ОС Unix (Linux).

**Необходимые знания:** понятие системного вызова, понятие программного прерывания, команда `int`, конвенция системных вызовов ОС Linux, системные вызовы `read`, `write` и `exit` (см. конспект соответствующих лекций).

**Задание:** переписать программу, написанную в ходе предыдущей работы, убрав директивы `%include` и все упоминания макросов `getchar`, `putchar`, `exitprogram`; для выполнения соответствующих действий использовать системные вызовы путём явной инициации программного прерывания.

## Лабораторная работа 1.7: макропроцессор

**Задача работы:** освоение возможностей макропроцессора

**Необходимые знания:** принципы работы макропроцессора, однострочные и многострочные макросы в ассемблере `nasm`, директивы условного ассемблирования и макроцикла.

**Задания** выдаются в индивидуальном порядке.

## Лабораторная работа 1.8: арифметический сопроцессор

**Задача работы:** освоение работы с арифметикой плавающей точки и арифметическим сопроцессором.

**Необходимые знания:** команды арифметического сопроцессора (F-команды).

**Задание:** написать на языке ассемблера `nasm` программу, считывающую из стандартного потока ввода три числа с плавающей точкой, задающие коэффициенты квадратного уравнения, и решающую квадратное уравнение в вещественных числах; если вещественных корней нет, выдать сообщение «no roots».

**Указания:** для перевода чисел из строкового представления в числовое и обратно используются уже известные алгоритмы (см. Л.Р. 1.3 и 1.4), с поправкой на то, что числа в нашем случае дробные. При переводе в строковое представление следует заранее задаться некоторой точностью (например, десять десятичных цифр после точки) и алгоритм перевода заканчивать, если соответствующая точность уже достигнута: в противном случае он может не кончиться никогда.

Объем задания этой лабораторной работы сравнительно велик, так что её следует частично или полностью выполнить дома или в ходе самостоятельной работы в классе.



## Часть 2: IV семестр

### Лабораторная работа 2.1: ознакомительная

**Задача работы:** освоить основные конструкции языка Си и работу с компилятором

**Необходимые знания:** операторы (управляющие конструкции) языка Си; выражения и операции языка Си; переменные, встроенные типы переменных, типы `int` и `double`; функции `printf` и `scanf`; командная строка компилятора `gcc` (см. кн. [3] и конспект первой лекции).

**Задание:**

- Воспользовавшись любым (по собственному выбору) редактором текстов, подходящим для программирования, набрать текст программы «Hello, world», откомпилировать её и запустить.
- Написать на языке Си программу, печатающую строку `Hello world` заданное число раз (например, 100 раз), воспользовавшись одной из конструкций цикла (`while`, `do-while` или `for`).
- Написать на языке Си программу для решения квадратных уравнений в вещественных числах с обязательным выделением вычисления дискриминанта в отдельную функцию; для ввода использовать функцию `scanf`.

**Указания:** при запуске компилятора *обязательно* указывайте флаги `-Wall` и `-g`, а также указывайте имя результирующего файла с помощью флага `-o`. После каждого действия убеждайтесь (с помощью команды `ls`), что ожидавшиеся файлы действительно созданы в вашей текущей директории.

### Лабораторная работа 2.2: посимвольный анализ текста

**Задача работы:** самостоятельное написание простых программ на языке Си

**Необходимые знания:** функции `getchar` и `putchar`, понятие ситуации конца файла.

**Задание** выдаётся индивидуально. Каждый из вариантов предполагает посимвольное чтение текста из стандартного потока ввода до возникновения ситуации «конец файла»; в зависимости от варианта,

требуется отдельный анализ каждой вводимой строки, либо анализ текста целиком.

Примеры индивидуальных вариантов задания:

1. Написать программу, которая читает из стандартного потока ввода (с клавиатуры) текст и после прочтения каждой очередной строки печатает количество пробелов, введённых в начале строки (то есть перед первым непробельным символом). Если строка начинается не с пробела, печатать ноль. При возникновении ситуации «конец файла» закончить работу.
2. Написать программу, которая читает из стандартного потока ввода (с клавиатуры) текст и для каждой прочитанной строки печатает количество встреченных в этой строке слов, заканчивающихся буквой 'z'. При возникновении ситуации «конец файла» закончить работу.
3. Написать программу, читающую из стандартного потока ввода (с клавиатуры) текст и для каждой строки печатающую количество слов, состоящих ровно из четырёх букв. При возникновении ситуации «конец файла» закончить работу.
4. Написать программу, читающую из стандартного потока ввода (с клавиатуры) текст и печатающую второе слово каждой строки. При возникновении ситуации «конец файла» закончить работу.

**Указания:** во всех вариантах этой работы хранение всего текста или строки целиком **не требуется**, так что нет необходимости в применении динамической памяти, как и во введении ограничений на длины строк, слов, текста целиком и т.п. Чтение следует выполнять посимвольно. Сколько бы ни было в вашей программе вызовов функции `getchar`, после каждого такого вызова необходимо проверять на возникновение ситуации «конец файла».

Обратите внимание, что функция `getchar` имеет тип возвращаемого значения `int`, а не `char`; присваивание возвращённого ею значения в переменную типа `char` является заведомой ошибкой, т.к. не позволяет отличить ситуацию «конец файла» от прочтения символа с кодом 255.

## Лабораторная работа 2.3: односвязные списки

**Задача работы:** освоение работы с односвязными списками

**Необходимые знания:** переменные типа «указатель», функции `malloc` и `free`, структурный тип данных, общие принципы работы с односвязными списками; техника, предполагающая передачу в функцию указателя на указатель (чтобы сделать возможными произвольную модификацию списка) и использование указателя на указатель в качестве переменной цикла (для внесения новых элементов и удаления элементов в произвольных позициях списка).

**Задание** выдаётся индивидуально. Примеры вариантов заданий:

- Написать программу, читающую (с помощью функции `scanf`) из стандартного потока ввода целые числа до возникновения ситуации «конец файла», сохраняя прочитанное в элементах списка; после возникновения ситуации «конец файла» распечатать введённые числа в обратном порядке, затем удалить из списка отрицательные числа и снова распечатать элементы списка.
- Написать программу, читающую (с помощью функции `scanf`) из стандартного потока ввода числа с плавающей точкой до возникновения ситуации «конец файла», сохраняя прочитанное в элементах списка; после возникновения ситуации «конец файла» вычислить среднее арифметическое введённых чисел, удалить из списка числа, меньшие, чем полученное среднее арифметическое, а оставшиеся числа распечатать в порядке, обратном их вводу.

**Указания:** удаление из списка элементов, удовлетворяющих условию, обязательно должно быть вынесено в отдельную функцию, получающую параметр типа «указатель на указатель». В процессе чтения добавлять новые элементы проще в начало списка, тогда для вывода «в обратном порядке» достаточно будет просмотреть список из начала в конец, поскольку числа в нём уже будут расположены в обратном порядке.

## Лабораторная работа 2.4: работа со строками

**Задача работы:** освоить работу со строками

**Необходимые знания:** понятие строки; параметры командной строки в языке Си.

**Задание** выдаётся индивидуально. Примеры вариантов заданий:

- Напечатать длину самого длинного из аргументов командной строки.

- Получить через аргументы командной строки букву и строку; удалить из заданной строки все вхождения заданной буквы, результат напечатать.
- \* Получить через аргументы командной строки слово и строку; сосчитать, сколько раз заданное слово встречается в заданной строке, результат напечатать.

**Указания:** использование библиотечных функций с префиксом `str` (таких как `strlen`, `strcmp`, `strstr`, `strdup` и т.д.) при выполнении данной работы **категорически запрещено**.

## Лабораторная работа 2.5: ввод строк неизвестной заранее длины

**Задача работы:** освоение работы со строками динамической длины.

Все **необходимые знания** у студентов к этому времени уже имеются, работа служит развитию и закреплению практических навыков.

**Задание** выдаётся индивидуально. Примеры задания:

- Написать программу, читающую из потока стандартного ввода фразы (по одной фразе в каждой строке) и в ответ на каждую прочитанную фразу (строку) выдающую строчку, составленную из первых букв слов фразы, потом из вторых и т.д., всего столько строк, какова длина самого длинного слова во фразе. Отсутствующие буквы заменяются пробелами; таким образом, каждое слово фразы оказывается напечатано вертикально сверху вниз. Например, в ответ на фразу «У попа была собака» должно быть напечатано

```

Упбс
оыо
плб
ааа
к
а

```

- Написать программу, читающую текст из стандартного потока ввода до получения ситуации «конец файла», а затем печатающую самое длинное из встреченных в тексте слов; если наибольшую длину имели несколько слов, разрешается напечатать любое из них.

- Написать программу, получающую через аргументы командной строки некоторое слово, затем читающую из потока стандартного ввода текст и печатающую только те строки, в которых данное слово встречается. Как слово, так и строки могут иметь произвольную длину, ограничения вводить запрещается. Получив на стандартном вводе ситуацию «конец файла», завершить работу.
- Написать программу, которая читает текст из потока стандартного ввода и печатает только те строки, в которых содержится не менее пяти слов (т.е. групп символов, разделённых пробелами). При возникновении ситуации «конец файла» завершить работу. Строки могут иметь произвольную длину, вводить ограничения на длину строки запрещается.
- Написать программу, которая читает текст из потока стандартного ввода, пока не будет получена ситуация «конец файла», затем печатает пять последних прочитанных строк и завершается; если всего введено было менее пяти строк, напечатать их все. Вводить ограничения на длину строки запрещается. Использовать внешние файлы для временного хранения информации запрещается. Хранить в памяти следует в каждый момент только последние пять строк, то есть не тратить память на строки, которые вам уже не понадобятся.

## Часть 3: V семестр

Лабораторные работы 3.1–3.5 представляют собой пять этапов создания одной компьютерной программы — модельного shell-интерпретатора

### Лабораторная работа 3.1: список слов

**Задача работы:** повторение пройденного материала, подготовка к написанию модельного командного интерпретатора.

**Необходимые знания:** язык Си, работа со строками, динамической памятью и списками, простейшие средства ввода-вывода.

**Задание:** написать программу, выполняющую в цикле чтение строк из потока стандартного ввода и разделяющую прочитанные строки на отдельные слова. Любое количество идущих подряд пробельных символов обрабатывается так же, как один пробел. Как строки, так и слова могут иметь произвольную длину, вводить явные ограничения запрещается.

В процессе прочтения очередной строки программа должна сформировать **список слов** этой строки. После завершения строки (т.е. при получении символа `'\n'`) слова из списка должны быть распечатаны, после чего список должен быть корректно уничтожен (т.е. должна быть освобождена вся занятая под него память), и программа должна приступить к чтению очередной строки; так работать до тех пор, пока не будет получена ситуация «конец файла».

**Указания:** Чтение каждого очередного слова удобнее всего вынести в отдельную функцию, которая в качестве внутренней структуры данных может использовать как список отдельных букв, так и массив символов изменяемой (динамической) длины.

### Лабораторная работа 3.2: запуск внешних программ

**Задача работы:** освоение базовых средств управления процессами в ОС Unix

**Необходимые знания:** понятие процесса и общие принципы управления процессами, системные вызовы `fork`, `execvp`, `exit`, `wait`, `chdir`, функция  `perror` (см. кн. [4], а также конспект соответствующей лекции).

**Задание:** модифицировать программу, написанную в ходе предыдущей работы, так, чтобы слова каждой введённой строки воспринимались как имя и аргументы для запуска внешней команды (программы),

т.е. в ответ на каждую введенную пользователем строку запускать соответствующую внешнюю программу с заданными аргументами, дожидаться её завершения, корректно уничтожить все созданные динамические данные и приступить к чтению следующей команды. Работать, как обычно, до возникновения ситуации «конец файла».

Дополнить программу «встроенной» командой `cd` для смены текущей директории; для этого проверять, не равно ли первое введенное пользователем слово строке "`cd`" и если равно, обрабатывать это как особый случай, не запуская никаких дочерних процессов, а вместо этого вызывая `chdir`.

**Указания:** Для вызова `execvp` необходимо сформировать *массив указателей* на элементы командной строки; эти элементы у вас уже есть в виде списка, так что достаточно сосчитать количество элементов списка и создать с помощью функции `malloc` массив размером на один указатель больше (т.е. если кол-во элементов списка равно `n`, то вызов `malloc` будет выглядеть так: `(char**)malloc(sizeof(char*)*(n+1))`).

Не забывайте о том, что **после вызова `execvp` строго обязательно проверка и завершение ненужного уже дочернего процесса, если вызов завершился ошибочно.**

## Лабораторная работа 3.3: фоновый режим

**Задача работы:** закрепление навыков управления процессами, корректное отрабатывание процессов-«зомби».

**Необходимые знания:** системный вызов `wait4` (либо `wait3`, либо `waitpid`, по выбору студента), макросы `WIFEXITED`, `WEXITSTATUS`, `WIFSIGNALED` и `WTERMSIG`.

**Задание:** Модифицировать подпрограмму чтения (см. ЛР №3.1) таким образом, чтобы она воспринимала символ `'&'` как разделительный символ (т.е. символ, который является отдельным словом сам по себе). Реализовать выполнение команд в фоновом режиме. Запускать в фоновом режиме команды, последним словом в котором является символ `'&'`. Если символ `'&'` встречен не в конце, выдавать сообщение об ошибке. По завершении команды, выполнявшейся в фоновом режиме, выдавать сообщение о ее завершении и код завершения. Программу написать так, чтобы процесс-«зомби» нельзя было увидеть во время выполнения обычных (нефоновых) команд.

**Указания:** Учтите, что при выполнении команды в **нефоновом** (обычном) режиме может получиться так, что до того, как она закончит работу, успеет завершиться кто-то из ранее запущенных фоновых

задач. Если при этом использовать обычный вызов `wait` для ожидания окончания нефоновой задачи, то он отреагирует на окончание другого (фонового) процесса; следовательно, необходимо проверять возвращаемое вызовом `wait` значение (подсказка: посмотрите в учебнике или в документации, какое конкретно значение возвращают вызовы семейства `wait`).

Для очистки от процессов-«зомби», оставшихся от ранее запущенных фоновых задач, необходимо периодически запускать цикл, снимающий всех имеющихся зомби, но при этом не блокирующийся в ожидании завершения процессов, которые пока не завершились. Для этого используется один из вызовов `wait4`, `wait3` или `waitpid` с опцией `WNOHANG`.

## Лабораторная работа 3.4: перенаправление ввода-вывода

**Задача работы:** освоение техники перенаправления стандартных потоков ввода-вывода в ОС Unix.

**Необходимые знания:** системные вызовы `open`, `close`, `read`, `write`, `dup` и `dup2`.

**Задание:** Модифицировать подпрограмму чтения (см. ЛР №3.1) таким образом, чтобы символы `'|'`, `'<'` и `'>'` воспринимались наряду с уже имеющимся с предыдущей работы символом `'&'` как разделительные (то есть всегда представляющие собой отдельное слово). Реализовать перенаправления стандартных потоков ввода и вывода (см. пособие [1], § 2.5).

## Лабораторная работа 3.5: построение конвейера

**Задача работы:** освоение взаимодействия процессов через именованные каналы.

**Необходимые знания:** вызов `pipe`.

**Задание:** Реализовать запуск команд конвейером. В минимальном варианте достаточно реализовать конвейер из двух команд, при наличии в командной строке более чем одного символа `'|'` выдавать сообщение об ошибке. В полном варианте ограничений на длину конвейера быть не должно.



## Лабораторная работа 3.6: сокет

**Задача работы:** освоение взаимодействия процессов по компьютерной сети с помощью сокетов потокового типа с использованием семейства адресации `AF_INET`.

**Необходимые знания:** системные вызовы `socket`, `connect`, `bind`, `listen`, `accept`, `read`, `write`, `shutdown`, `close`, особенности сокетов типа `SOCK_STREAM`, адресация в семействе `AF_INET` (стек протоколов `IPv4`); использование утилиты `telnet`.

**Задание:** написать программу-сервер, принимающую входящие соединения на TCP-сокет; сразу после принятия соединения отправлять клиенту строку "What is your name?", дожидаться ответа клиента, считая всё присланное им вплоть до символа конца строки «именем»; получив конец строки, отправлять клиенту сообщение вида "Pleased to meed you, dear NNNNNN", где NNNNNN — присланное клиентом «имя»; после этого разрывать соединение и принимать следующий запрос на соединение.

Для проверки работы сервера используйте программу `telnet`.

**Указания:** сервер в таком варианте оказывается «однопользовательским», то есть не может одновременно обслуживать больше одного клиента. Это не страшно; нашей задачей было освоение основных системных вызовов, связанных с сокетами. Решению проблемы многопользовательского доступа будут посвящены две следующие работы.

## Лабораторная работа 3.7: сервер с обслуживающими процессами

**Задача работы:** построение простейшего варианта многопользовательского TCP-сервера

Все **необходимые знания** у студентов уже имеются; рекомендуется повторить материал, связанный с системными вызовами `fork`, `exit`, `wait` и `wait4`.

**Задание:** Написать программу-сервер, работающую так же, как и написанная в ходе предыдущей работы, но способную обслуживать одновременно неограниченное количество клиентов (во всяком случае, не ограниченное ничем, кроме максимального числа одновременно открытых дескрипторов),

Для этого каждый раз сразу после установки соединения с очередным клиентом порождать дочерний процесс, предназначенный для обслуживания этого конкретного клиента. По окончании обслуживания

дочерний процесс должен завершаться.

**Указания:** не забывайте про процессы-«зомби», остающиеся от порождаемых вами процессов.

## Лабораторная работа 3.8: мультиплексирующий сервер

**Задача работы:** освоение техники событийно-ориентированного программирования.

**Необходимые знания:** системный вызов `select`.

**Задание:** написать программу-сервер, работающую так же, как и написанный в ходе предыдущей работы, но не порождающий никаких дополнительных процессов; для этого воспользоваться вызовом `select`.

## Литература

- [1] Столяров А. В. Архитектура ЭВМ и системное программное обеспечение: пособие по выполнению лабораторных работ на ЭВМ в среде ОС Unix. М.:РИО МГТУГА, 2009.
- [2] Баурн С. Операционная система UNIX. М.: Мир, 1986.
- [3] Керниган Б., Ритчи Д. Язык Си. СПб.: Невский диалект, 2001.
- [4] Робачевский А. М. Операционная система UNIX. СПб.: BHV, 1997.

# Содержание

<b>Часть 1: III семестр</b>	<b>3</b>
Лабораторная работа 1.1: ознакомительная . . . . .	3
Лабораторная работа 1.2: ассемблер <code>asm</code> . . . . .	4
Лабораторная работа 1.3: чтение числа . . . . .	5
Лабораторная работа 1.4: печать числа . . . . .	6
Лабораторная работа 1.5: подпрограммы и стек . . . . .	7
Лабораторная работа 1.6: системные вызовы . . . . .	7
Лабораторная работа 1.7: макропроцессор . . . . .	7
Лабораторная работа 1.8: арифметический сопроцессор . .	8
<b>Часть 2: IV семестр</b>	<b>9</b>
Лабораторная работа 2.1: ознакомительная . . . . .	9
Лабораторная работа 2.2: посимвольный анализ текста . .	9
Лабораторная работа 2.3: односвязные списки . . . . .	10
Лабораторная работа 2.4: работа со строками . . . . .	11
Лабораторная работа 2.5: ввод строк неизвестной заранее длинны . . . . .	12
<b>Часть 3: V семестр</b>	<b>14</b>
Лабораторная работа 3.1: список слов . . . . .	14
Лабораторная работа 3.2: запуск внешних программ . . . .	14
Лабораторная работа 3.3: фоновый режим . . . . .	15
Лабораторная работа 3.4: перенаправление ввода-вывода .	16
Лабораторная работа 3.5: построение конвейера . . . . .	16
Лабораторная работа 3.6: сокеты . . . . .	17
Лабораторная работа 3.7: сервер с обслуживающими про- цессами . . . . .	17
Лабораторная работа 3.8: мультиплексирующий сервер . .	18
<i>Литература</i>	<b>19</b>