

МОСКОВСКИЙ ТЕХНИКУМ КОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

УТВЕРЖДАЮ
Заместитель директора
по учебной работе
Давыдова А. А.

*РАЗРАБОТКА ПРОГРАММНОГО
ОБЕСПЕЧЕНИЯ ДЛЯ АВТОМАТИЧЕСКОГО
УПРАВЛЕНИЯ ЭЛЕКТРОПИТАНИЕМ
АКТИВНОГО СЕТЕВОГО ОБОРУДОВАНИЯ*

Пояснительная записка

МТКП.340015.000 81

МП41-06

Листов 111

Старший консультант

Сидорова Н. А.

Руководитель разработки

Муковников Ю. А.

Консультант по экономической части

Кардаш Г. Г.

Рецензент

Храпова Л. А.

Председатель предметной комиссии

Жилкина Н. А.

Разработал

Муковников М. Ю.

2010

Содержание

1	Введение	4
2	Специальная часть	7
2.1	Постановка задачи	7
2.1.1	Назначение	7
2.1.2	Технико-математическое описание задачи	8
2.1.3	Требования к программному обеспечению	8
2.1.4	Описание формата обмена	10
2.1.5	Интерфейс взаимодействия с ЭВМ	11
2.1.5.1	Функция CreateFile	11
2.1.5.2	Функция CloseHandle	16
2.1.5.3	Функция ReadFile	16
2.1.5.4	Функция WriteFile	19
2.1.5.5	Функция SetCommState	21
2.1.5.6	Функция GetCommState	23
2.1.5.7	Структура DCB	23
2.1.5.8	Функция SetCommTimeouts	29
2.1.5.9	Структура COMMTIMEOUTS	30
2.2	Схемы алгоритмов программ	32
2.2.1	Схема алгоритма программы keusbd	32
2.2.2	Схема алгоритма программы keusb	33
2.3	Отладка программ	34
2.4	Инструкция по эксплуатации	36
2.4.1	Назначение программ	36
2.4.2	Условия выполнения	36
2.4.3	Подготовка к запуску	36
2.4.4	Настройка программы	37
2.4.5	Запуск программы	39
2.4.6	Сообщения и ошибки	41
2.4.7	Завершение работы программы	44
3	Охрана труда	45
3.1	Введение	45
3.2	Основные понятия гигиены, физиологии и психологии труда	47

3.3	Техника безопасности при работе на компьютере	49
4	Экономическая часть	52
4.1	Технико-экономическое обоснование	52
4.2	Расчёт трудоёмкости	53
4.3	Расчёт себестоимости	59
4.3.1	Расчёт себестоимости на этапе машинно-ручных работ . .	60
4.3.2	Расчёт себестоимости на этапах от $C_{и}$ до $C_{д}$	63
4.3.3	Расчёт полной себестоимости	66
4.3.4	Анализ структуры себестоимости	66
4.4	Графическая часть	68
4.5	Эффективность от внедрения	71
5	Заключение	72
	Список используемой литературы	73
	Приложение А. Листинг программы	74
	Приложение Б. Результаты выполнения программы	107

1 ВВЕДЕНИЕ

В настоящее время уже ни у кого не вызывает удивления повсеместное использование компьютеров: в офисах крупных и мелких компаний, в высших и средних учебных заведениях, дома; работу многих отраслей уже практически невозможно представить без использования компьютерной техники. Существующих мощностей вычислительных систем не хватает разве что лишь на моделирование нейронных сетей, по масштабу структуры сопоставимых с мозгом животных, а огромное количество выпускаемых устройств на базе микропроцессоров способствует снижению их, и без того низкой, себестоимости. Сама идея построения любой сложной системы без применения микропроцессоров (а, скажем, на транзисторах) уже является абсурдной по причине сложности, низких качественных показателей и абсолютной нерентабельности. Можно привести образное сравнение — если бы Боинг 747 прогрессировал с такой же скоростью, с какой прогрессирует твердотельная электроника, то он умещался бы в спичечном коробке и облетал бы без дозаправки земной шар 40 раз.

Между тем, компьютер, как самостоятельная единица, не может похвастать высокой надёжностью, гибкостью и возможностью неограниченного увеличения производительности без существенного усложнения архитектуры. После появления возможности объединения компьютеров в одну сеть данная проблема сошла на нет — объёмы накопителей информации нескольких узлов могут быть объединены, а алгоритмы решения задач могут быть распараллелены, что с увеличением вычислительных узлов в сети даст близкое к линейному ускорение решения задачи. Существующая гибкость сетей даёт возможности для горячей замены узлов, выполняющих схожую или одинаковую работу без необходимости остановки всего процесса. Данная гибкость многократно увеличивает как надёжность системы в целом, так и возможность масштабирования. Всё это свело на нет необходимость существования, так называемых, суперкомпьютеров — судите сами, существующие системы распределённых вычислений состоят из нескольких миллионов узлов, а их производительность достигает десятка петафлопс (10 квадриллионов или 10 000 000 000 000 000 элементарных операций в секунду), когда самый мощный суперкомпьютер может похвастать производительностью “лишь” 1.8 петафлопс. Самая масштабная сеть на планете, Интернет, без которой уже просто не может представить своей жизни каждый современный человек, является самоупорядо-

ченной децентрализованной структурой примерно из половины миллиарда узлов, распределённых по всей планете. Она ежедневно обеспечивает передачу нескольких петабайт информации и способствует движению денежных средств на десятки миллионов рублей.

Однако необходимо не забывать, что в процессе масштабирования происходит логарифмическое увеличение сетевых устройств — оборудования, необходимого для функционирования сети. Активное сетевое оборудование — это сложный аппаратно-программный комплекс, построенный на основе микроконтроллеров и высокоспециализированного программного обеспечения, дающего возможности применения высокоточных математических методов, необходимых для расчётов маршрутов и нагрузки сетевых каналов, лёгкого масштабирования и связи разнотипных устройств друг с другом. К сожалению, существует тот факт, что при всей сложности, существующая система является довольно чувствительной к воздействию с внешней стороны, будь это сильные электромагнитные помехи или перепады в сети электропитания. Это создаёт весьма острую проблему необходимости обеспечения постоянной работоспособности всей сетевой инфраструктуры, с целью локального решения которой и выполнена данная работа.

В задачу данной работы входит циклическая проверка доступности заданных частей сети, путём отправки служебных ICMP-запросов. В случае недоступности, в сетевом оборудовании, ответственном за связь с данной подсетью, на несколько секунд разрывается электрическая цепь, осуществляя тем самым жёсткий сброс виновного устройства. Вновь получив питание, устройство заново проводит инициализацию, самотестирование и восстанавливает соединение. Коммутирование цепей питания осуществляется через специальное устройство Ke-USB24R, подключённое к шине USB и программно управляемое через интерфейс виртуального порта RS-232C. Будучи основанным на микроконтроллере, данное устройство предоставляет специальный протокол, посредством которого возможно управление состоянием четырёх высоковольтных реле, обеспечивающих коммутацию цепей питания подключённых устройств. Общая схема данной системы изображена на рисунке 1.1.

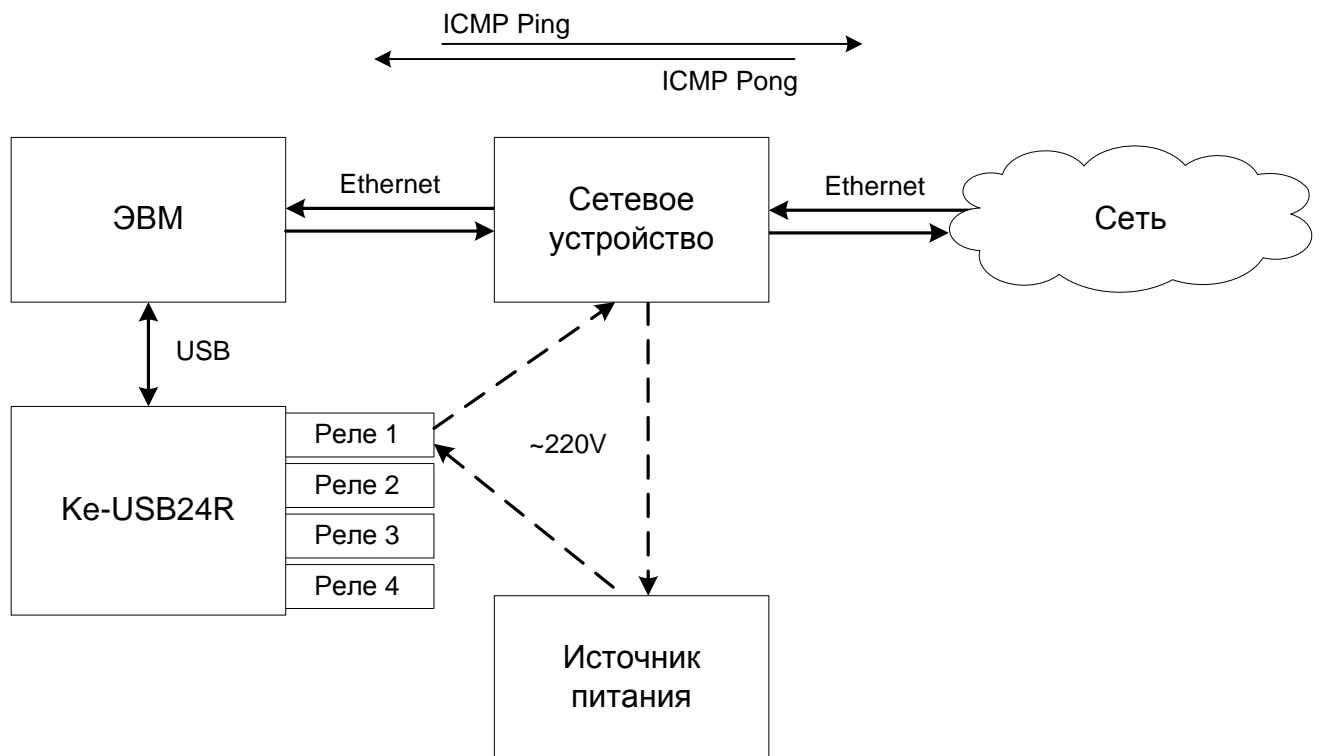


Рисунок 1.1 – Схема взаимодействия компонент системы

Целью данной дипломной работы является разработка программного обеспечения для поддержания функционирования сетевой инфраструктуры путём автоматического управления электропитанием активного сетевого оборудования.

2 СПЕЦИАЛЬНАЯ ЧАСТЬ

2.1 Постановка задачи

Темой данной дипломной работы является “Разработка программного обеспечения для автоматического управления электропитанием активного сетевого оборудования”.

2.1.1 Назначение

Данное программное обеспечение должно выполнять следующие операции:

а) считывать настройки из конфигурационного файла, включающие в себя:

1) список действий и дерево закреплённых за ними правил, каждый элемент которого может из себя представлять:

- доменное имя или IP-адрес хоста;
- операцию логического “И”;
- операцию логического “ИЛИ”;
- операцию логического исключающего “ИЛИ”;
- операцию логического “НЕ”;

2) частоту проверки состояния сети и количество посылаемых пакетов;

б) циклически обрабатывать деревья, путём отправки ICMP-запросов заданным узлам и применение логических операций к результату;

в) в случае получения логического нуля продолжить обработку следующего дерева, иначе:

1) подключиться к указанному устройству;

2) послать устройству команду, сформированную на основе заданного действия, среди которых могут быть:

- отключение электропитания заданной линии;

- включение электропитания заданной линии;
- переключение состояния электропитания на заданной линии;
- сброс на заданное количество секунд электропитания заданной линии;
- получение текущего состояния реле на заданной линии;
- жёсткий сброс устройства с восстановлением настроек по умолчанию;

3) проанализировать возвращённые устройством данные и отключиться от устройства;

г) выждать заданное количество времени и перейти к пункту б).

В результате работы данного программного обеспечения будет гарантирована поддержка работоспособности критических частей сети с учётом всех возможных случаев (например, если в тестируемой подсети один узел вышел из строя, но доступен другой, то перезагружать оборудование не имеет смысла).

2.1.2 Техничко-математическое описание задачи

Описание формата обмена данными с устройством приведено в пункте 2.1.4. Описание интерфейсных функций операционной системы, необходимых для работы данного программного обеспечения приведено в пункте 2.1.5.

2.1.3 Требования к программному обеспечению

Данное программное обеспечение должно отвечать следующим требованиям:

- простота в использовании;
- повышенная надёжность, предполагающая непрерывную работу программы в течении нескольких месяцев;

- возможность запуска на серверах, где отсутствует поддержка графического интерфейса;
- хранение настроек в отдельном файле в понятном для человека виде с возможностью изменения любым текстовым редактором;
- запись всех событий, предупреждений и ошибок в отдельный, легко читаемый человеком, файл;
- предоставление возможностей управления электропитанием другим программам.

Данное программное обеспечение должно быть представлено в виде двух частей: демон¹⁾, названный `keusbd`, и драйвер устройства, названный `keusb`.

Драйвер устройства должен работать в пользовательском режиме (третье кольцо защиты), а контакт с устройством производить через специальные интерфейсы, предоставляемые операционной системой. Далее, в его обязанности входит поиск устройства по адресу или серийному номеру (если устройство в системе в одном экземпляре, оно должно быть выбрано по умолчанию без указания адреса), формирование команд на любое действие (будь то включение, выключение, переключение или сброс линии) и обработка возвращённых данных.

Демон при запуске должен считывать конфигурационный файл, хранящий основные параметры работы (наборы правил опроса устройств и все временные интервалы) и, в случае корректности оных, уйти в фоновый режим, начав анализирование сети.

Обе программы должны выполняться на любой платформе из `x86`, `amd64`, `sparc`, `ppc64`, `mips`, `ia64` или `arm` под управлением любой POSIX-совместимой операционной системой с установленным пакетом `iputils` (среди них `BeOS`, `Mac OS X`, `OpenSolaris`, `OpenVMS`, `QNX`, `BeOS`, `FreeBSD` и `GNU/Linux`) или под управлением операционной системой `Windows`.

Для разработки необходимо использовать стандартизированный процедурный язык программирования `ANSI C`. Выбор языка был продиктован следующими соображениями:

- программы, написанные на нём могут быть легко перенесены на множество других платформ;

¹⁾ Здесь и далее: демон (англ. *daemon*) — программа, работающая в фоновом режиме без прямого общения с пользователем

- небольшие размеры стандартной библиотеки делают возможным статическую компиляцию с целью запуска из специального образа initrd при отсутствии доступа к файловым системам;
- небольшой уровень абстракций в решаемой задаче не требует применения объектно-ориентированных или функциональных языков программирования;
- простота связывания с системными библиотеками с целью прямого использования системных интерфейсов.

2.1.4 Описание формата обмена

Для управления модулем KE-USB24R предназначен ряд команд в текстовом формате, называемых KE командами. Любая KE команда, отсылаемая модулю, должна начинаться с символов “\$KE”. Также все команды должны заканчиваться символом возврата каретки <CR> и символом перехода на новую строку <LF> (в шестнадцатеричном формате эти символы имеют коды 0x0D и 0x0A соответственно).

Ответы модуля на команды, а также отдельные информационные блоки, выдаваемые модулем, всегда начинаются с символа “#” (шестнадцатеричный код 0x23) и заканчиваются символами возврата каретки <CR> и перехода на новую строку <LF>. Далее по тексту документа символы <CR><LF>, которыми должна заканчиваться любая команда модулю и любой ответ, выдаваемый модулем, опускаются.

В том случае, если, синтаксис команды, отправленной модулю, не является верным, модуль выдает сообщение об ошибке: “#ERR”.

Команда тестовой проверки модуля \$KE

Команда проверки работоспособности модуля. Это простая тестовая команда, на которую модуль должен ответить “#OK”.

Команда получения серийного номера \$KE,SER

Чтение серийного номера модуля. Каждый модуль имеет свой собственный уникальный серийный номер. В качестве ответа должно выступить сообщение вида “#SER,<Серийный номер>”.

Команда жёсткого сброса \$KE,RST

Сброс всех настроек модуля в значение по умолчанию. Ответ на запрос: “#RST,OK”.

Команда управления реле \$KE,REL,<Номер Реле>,<Состояние>

Команда предназначена для управления реле модуля. Номер реле может быть в пределах от 1 до 4 включительно, а состояние принимать значение 0 или 1. Если состояние равно 0, то контакты реле 1 и 2 замкнуты, а 2 и 3 разомкнуты, иначе контакты реле 1 и 2 разомкнуты, 2 и 3 замкнуты (реле включено). Ответ на запрос: “#REL,OK”.

Команда получения состояния реле \$KE,RDR,<Номер реле>

Команда позволяет определить, в каком сейчас состоянии находится выбранное реле — включено оно или выключено. Номер реле может быть в пределах от 1 до 4 включительно. Ответ на запрос: “#RDR,<Номер реле>,<Состояние>”. Если состояние равно 0, то реле выключено, если же состояние равно 1, то, соответственно, реле включено.

Команда получения состояния всех реле \$KE,RDR,ALL

Команда позволяет получить состояние всех четырех реле за один запрос. Ответ на запрос: “#RDR,ALL,<Состояние 1>,<Состояние 2>,<Состояние 3>,<Состояние 4>”. Если состояние равно 0, то соответствующее реле выключено, если же состояние равно 1, то реле включено.

2.1.5 Интерфейс взаимодействия с ЭВМ

2.1.5.1 Функция CreateFile

Функция CreateFile создает или открывает каталог, физический диск, том, буфер консоли, устройство на магнитной ленте, коммуникационный ресурс, почтовый слот или именованный канал. Функция возвращает дескриптор, который может быть использован для доступа к объекту.

Синтаксис

```
HANDLE CreateFile(  
    LPCTSTR lpFileName,  
    DWORD   dwDesiredAccess,  
    DWORD   dwShareMode,  
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    DWORD   dwCreationDisposition,  
    DWORD   dwFlagsAndAttributes,  
    HANDLE  hTemplateFile  
);
```

Параметры

а) `lpFileName` — указатель на символьную строку с нулем в конце, устанавливающую имя объекта, который создается или открываться.

б) `dwDesiredAccess` — тип доступа к объекту (чтение, запись или то и другое). Вы не можете требовать режима доступа, который находится в противоречии с режимом совместного использования, заданным в предыдущем запросе открытия объекта, дескриптор которого является все еще открытым. Если этот параметр равняется нулю, приложение может сделать запрос атрибутов устройства, не обращаясь к диску. Если этот параметр равняется нулю, приложение может запросить атрибуты файла и диска, не обращаясь к устройству. Это применяется в том случае, если приложение хочет выяснить размер дискеты флоппи-диска и форматов, которые он поддерживает, не требуя наличия гибкого диска в диске. Это может также использоваться и для того, чтобы проверить существование файла или каталога, не открывая его доступа для чтения или записи.

в) `dwShareMode` — режим совместного доступа (чтение, запись, оба или никакого действия). Вы не можете требовать режим совместного доступа, который находится в противоречии с режимом доступа, заданным в предыдущем запросе открытия объекта, дескриптор которого является все еще открытым. Сделав так, вы в результате получите ошибку совместного доступа (`ERROR_SHARING_VIOLATION`). Если этот параметр равняется нулю, а `CreateFile` завершается успешно, объект не может совместно использоваться и не может быть открыт снова до тех

пор, пока дескриптор не закроется. Чтобы дать возможность другим процессам совместно использовать объект, в то время, когда ваш процесс открыт, используйте комбинацию из одного или нескольких нижеследующих значений, чтобы определить режим доступа, который они могут запросить, когда открывают объект. Эти параметры совместного использования остаются в силе до тех пор, пока Вы не закроете дескриптор объекта. Значение может состоять из любой комбинации следующих флагов:

1) FILE_SHARE_DELETE — разрешает последующие операции открытия объекта, которые требуют доступа к его удалению. В противном случае, другие процессы не смогут открыть объект, если они потребуют доступа к удалению. Если этот флажок не определяется, но объект уже был открыт с доступом для удаления, то функция завершается ошибкой.

2) FILE_SHARE_READ — разрешает последующие операции открытия объекта, которые требуют доступа для чтения. В противном случае, другие процессы не смогут открыть объект, если они потребуют доступа для чтения. Если этот флажок не определяется, но объект уже был открыт с доступом для чтения, то функция завершается ошибкой.

3) FILE_SHARE_WRITE — разрешает последующие операции открытия объекта, которые требуют доступа для записи. В противном случае, другие процессы не смогут открыть объект, если они потребуют доступа для записи. Если этот флажок не определяется, но объект уже был открыт с доступом для записи, то функция завершается ошибкой.

г) lpSecurityAttributes — указатель на структуру SECURITY_ATTRIBUTES, которая устанавливает может ли возвращенный дескриптор быть унаследован дочерними процессами. Если lpSecurityAttributes имеет значение NULL, дескриптор не может быть унаследован.

д) dwCreationDisposition — выполняемые действия с файлами, которые существуют и выполняемые действия с файлами, которые не существуют. Этот параметр должен быть одним из следующих значений:

1) CREATE_NEW — создаёт новый файл. Функция завершается ошибкой, если заданный файл уже существует.

2) `CREATE_ALWAYS` — создаёт новый файл. Если файл существует, функция переписывает файл, сбрасывает существующие атрибуты и объединяет, заданные параметром `dwFlagsAndAttributes` атрибуты файла и флажки, с `FILE_ATTRIBUTE_ARCHIVE`, но не устанавливает дескриптор безопасности заданный структурой `SECURITY_ATTRIBUTES`.

3) `OPEN_EXISTING` — открывает файл. Функция завершается ошибкой, если файл не существует.

4) `OPEN_ALWAYS` — открывает файл, если таковой существует. Если файл не существует, функция создает файл, как будто бы `dwCreationDisposition` имел значение `CREATE_NEW`.

5) `TRUNCATE_EXISTING` — открывает файл и обрезает его так, чтобы его размер равнялся нулю байтов. Вызывающий процесс должен открыть файл с правом доступа `GENERIC_WRITE`. Функция завершается ошибкой, если файл не существует.

е) `dwFlagsAndAttributes` — атрибуты и флажки файла. Ниже следуют атрибуты и флажки файла, которые используются только для объектов файла, а не для других объектов, создаваемых функцией `CreateFile`. Когда `CreateFile` открывает существующий файл, он объединяет флажки файла с существующими его атрибутами, но игнорирует любые предоставляемые атрибуты файла. Этот параметр может включать в себя любую комбинацию следующих атрибутов файла (заметьте, что все другие атрибуты файла не принимают во внимание атрибут `FILE_ATTRIBUTE_NORMAL`):

1) `FILE_ATTRIBUTE_ARCHIVE` — файл должен быть архивирован. Приложения используют этот атрибут, чтобы отметить файлы для резервного копирования или перемещения.

2) `FILE_ATTRIBUTE_ENCRYPTED` — файл или каталог шифруются. Для файла, это означает, что все данные в файле зашифрованы. Для каталога, это означает, что шифрование — это значение по умолчанию для недавно созданных файлов и подкаталогов. Этот флажок не действует, если также установлен и флажок `FILE_ATTRIBUTE_SYSTEM`.

3) `FILE_ATTRIBUTE_HIDDEN` — файл скрытый. Он не должен включаться в обычный перечень файлов каталога.

4) FILE_ATTRIBUTE_NORMAL — у файла нет других установленных атрибутов. Этот атрибут допустим только в том случае, если он используется один.

5) FILE_ATTRIBUTE_NOT_CONTENT_INDEXED — файл не будет индексироваться службой индексации содержания.

6) FILE_ATTRIBUTE_OFFLINE — данные файла доступны не сразу. Этот атрибут указывает, что данные файла были физически перемещены на автономное хранилище данных. Этот атрибут используется Удаленным хранилищем, программой Иерархического управления памятью. Приложения произвольно не должно изменять этот атрибут.

7) FILE_ATTRIBUTE_READONLY — файл доступен только для чтения. Приложения могут читать файл, но не могут записать в него или удалить его.

8) FILE_ATTRIBUTE_SYSTEM — файл является частью или используется исключительно операционной системой.

9) FILE_ATTRIBUTE_TEMPORARY — файловые системы избегают писать данные обратно на запоминающее устройство большой ёмкости, если доступна достаточная кэш-память, потому что приложение часто удаляет временный файл вскоре после того, как дескриптор закрывается. В этом случае, система может полностью отменить запись данных. В противном случае, данные должны быть записаны после закрытия дескриптора.

ж) hTemplateFile — дескриптор файла шаблона с правом доступа GENERIC_READ. Файл шаблона предоставляет атрибуты файла и дополнительные атрибуты для создающегося файла. Этот параметр должен быть NULL. При открытии существующего файла, CreateFile игнорирует файл шаблона.

Возвращаемые значения

Если функция завершается успешно, возвращаемое значение — открытый дескриптор заданного файла. Если заданный файл существовал до вызова функции, а параметр dwCreationDisposition установлен в CREATE_ALWAYS или OPEN_ALWAYS, вызов GetLastError возвращает значение ERROR_ALREADY_EXISTS

(даже при том, что функция завершилась успешно). Если файл не существовал перед вызовом функции, функция GetLastError возвращает ноль. Если функция завершается с ошибкой, возвращаемое значение — INVALID_HANDLE_VALUE. Чтобы получать дополнительные сведения об ошибке, вызовите GetLastError.

2.1.5.2 Функция CloseHandle

Функция CloseHandle закрывает дескриптор открытого объекта.

Синтаксис

```
BOOL CloseHandle(  
    HANDLE hObject  
);
```

Параметры

а) hObject — дескриптор открытого объекта.

Возвращаемые значения

Если функция завершается успешно, величина возвращаемого значения — не ноль. Если функция завершается с ошибкой, величина возвращаемого значения — ноль. Чтобы получить дополнительные данные об ошибке, вызовите GetLastError.

2.1.5.3 Функция ReadFile

Функция ReadFile читает данные из файла, начиная с позиции, обозначенной указателем файла. После того, как операция чтения была закончена, указатель

файла перемещается на число действительно прочитанных байтов, если дескриптор файла не создан с атрибутом асинхронной операции. Если дескриптор файла создается для асинхронного ввода-вывода, приложение должно переместить позицию указателя файла после операции чтения. Эта функция предназначена и для синхронной и асинхронной операции. Функция ReadFileEx предназначена исключительно для асинхронной операции. Это дает возможность приложению выполнять другие действия в ходе операции чтения файла.

Синтаксис

```
BOOL ReadFile(  
    HANDLE    hFile,  
    LPVOID    lpBuffer,  
    DWORD     nNumberOfBytesToRead,  
    LPDWORD   lpNumberOfBytesRead,  
    LPOVERLAPPED lpOverlapped  
);
```

Параметры

а) hFile — дескриптор файла, который читается. Дескриптор файла должен быть, создан с правом доступа GENERIC_READ.

б) lpBuffer — указатель на буфер, который принимает прочитанные данные из файла.

в) nNumberOfBytesToRead — число байтов, которые читаются из файла.

г) lpNumberOfBytesRead — указатель на переменную, которая получает число прочитанных байтов. Функция ReadFile устанавливает это значение в ноль перед началом любой работы или проверкой ошибок. Если этот параметр равняется нулю, когда ReadFile возвращает значение TRUE для именованного канала, другой конец канала в режиме передачи сообщений вызывает функцию WriteFile с параметром nNumberOfBytesToWrite установленным в ноль. Если используются порты завершения ввода-вывода, а Вы используете процедуру повторного вызова, чтобы освободить занимаемую память структурой OVERLAPPED, на которую указывает

параметр `lpOverlapped`, задайте `NULL`, как значение этого параметра, чтобы избежать проблемы искажения данных в памяти в ходе ее освобождения. Эта проблема искажения данных в памяти становится причиной возвращения в этом параметре неверного числа байтов.

д) `lpOverlapped` — указатель на структуру `OVERLAPPED`. Эта структура требуется тогда, если параметр `hFile` создавался с флажком `FILE_FLAG_OVERLAPPED`. Если `hFile` был открыт с флажком `FILE_FLAG_OVERLAPPED`, у параметра `lpOverlapped` не должно быть значения `NULL`. Он должен указать на правильную структуру `OVERLAPPED`. Если `hFile` создавался с флажком `FILE_FLAG_OVERLAPPED`, а `lpOverlapped` имеет значение `NULL`, функция может неправильно сообщить о завершении операций чтения. Если `hFile` был открыт с флажком `FILE_FLAG_OVERLAPPED`, а `lpOverlapped` имеет значение не `NULL`, операция чтения начинается при смещении, заданном в структуре `OVERLAPPED`, и `ReadFile` может вернуть значение прежде, чем операция чтения будет закончена. В этом случае, `ReadFile` возвращает значение `FALSE`, а функция `GetLastError` возвращает значение `ERROR_IO_PENDING`. Это дает возможность вызывающему процессу продолжиться, в то время как операция чтения заканчивается. Событие, определяемое в структуре `OVERLAPPED` устанавливается в сигнальное состояние после завершения операции чтения. Вызывающая программа должна корректировать местоположение указателя позиции в файле после завершения работы. Функция `ReadFile` сбрасывает событие, указанное членом `hEvent` структуры `OVERLAPPED` в несигнальное состояние, когда она начинает операцию ввода-вывода. Поэтому, нет необходимости для вызывающей программы, чтобы делать так. Если `hFile` не открывался с флажком `FILE_FLAG_OVERLAPPED`, а `lpOverlapped` — значение `NULL`, операции чтения начинается в текущей позиции файла и `ReadFile` не возвращает значения до тех пор, пока операция не будет закончена. Система модернизирует указатель позиции в файле после завершения работы. Если `hFile` не открывался с `FILE_FLAG_OVERLAPPED`, а `lpOverlapped` — не `NULL`, операция чтения стартует при смещении, указанном в структуре `OVERLAPPED`. `ReadFile` не возвращает значение до тех пор, пока операция чтения не завершилась. Система модернизирует позицию указателя в файле после завершения работы.

Возвращаемые значения

Функция ReadFile возвращает значение тогда, когда выполнено одно из ниже перечисленных условий:

- а) операция записи завершается на записывающем конце канала;
- б) затребованное число байтов прочитано;
- в) происходит ошибка.

Если функция завершается успешно, величина возвращаемого значения — не ноль. Если функция завершается с ошибкой, величина возвращаемого значения — ноль. Чтобы получить дополнительные сведения об ошибке, вызовите GetLastError.

Если величина возвращаемого значения — не ноль, а число прочитанных байтов равняется нулю, указатель файла был за пределами текущего конца файла на момент операции чтения. Однако, если файл был открыт с флажком FILE_FLAG_OVERLAPPED, и lpOverlapped имеет значение не NULL, величина возвращаемого значения — ноль, а GetLastError возвращает ошибку ERROR_HANDLE_EOF, когда указатель файла проходит вне текущего конца файла.

2.1.5.4 Функция WriteFile

Функция WriteFile пишет данные в файл с места, обозначенного указателем позиции в файле. Эта функция предназначена и для синхронной, и для асинхронной операции. Функция WriteFileEx предназначена исключительно для асинхронной операции.

Синтаксис

```
BOOL WriteFile(  
    HANDLE    hFile,  
    LPCVOID   lpBuffer,  
    DWORD     nNumberOfBytesToWrite,
```

```
LPDWORD lpNumberOfBytesWritten,  
LPOVERLAPPED lpOverlapped  
);
```

Параметры

а) `hFile` — дескриптор файла. Дескриптор файла, должен быть создан с правом доступа `GENERIC_WRITE`. Для получения дополнительной информации, см. статью *Защита файла и права доступа*.

б) `lpBuffer` — указатель на буфер, содержащий данные, которые будут записаны в файл.

в) `nNumberOfBytesToWrite` — число байтов, которые будут записаны в файл. Значение нуля определяет пустую операцию записи. Поведение пустой операции записи зависит от лежащей в основе файловой системы. Чтобы сократить или продлить файл, используйте функцию `SetEndOfFile`. Операции записи в именованном канале по всей сети ограничены 65 535 байтами.

г) `lpNumberOfBytesWritten` — указатель на переменную, которая получает число записанных байтов. Функция `WriteFile` устанавливает это значение в ноль перед выполнением какой-либо работы или выявлением ошибок. Если порты завершения ввода-вывода используются, а Вы используете процедуру обратного вызова, чтобы освободить занимаемую память структурой `OVERLAPPED`, на которую указывает параметр `lpOverlapped`, установите `NULL` как значение этого параметра, чтобы избежать проблемы порчи данных в памяти в ходе освобождения ресурса. Эта проблема порчи данных в памяти становится причиной неправильного числа байтов, которые возвращаются в этом параметре.

д) `lpOverlapped` — указатель на структуру `OVERLAPPED`. Эта структура требуется тогда, если параметр `hFile` создавался с флажком `FILE_FLAG_OVERLAPPED`. Если `hFile` был открыт с флажком `FILE_FLAG_OVERLAPPED`, у параметра `lpOverlapped` не должно быть значения `NULL`. Он должен указывать на правильную структуру `OVERLAPPED`. Если `hFile` был открыт с флажком `FILE_FLAG_OVERLAPPED`, а `lpOverlapped` имеет значение `NULL`, функция может неправильно сообщить о завершении операции записи. Если `hFile` был открыт с флажком

FILE_FLAG_OVERLAPPED, а lpOverlapped имеет значение не NULL, операция записи начинается при смещении, заданном в структуре OVERLAPPED, а WriteFile может вернуть значение прежде, чем операция записи будет закончена. В этом случае, WriteFile возвращает значение FALSE, а функция GetLastError возвращает значение ERROR_IO_PENDING. Это дает возможность вызывающему процессу продолжать работу до тех пор, пока операция записи не закончится. После завершения операции записи, событие, определяемое в структуре OVERLAPPED устанавливается в сигнальное состояние. Вызывающая программа должна корректировать позицию указателя позиции в файле после завершения операции. Если hFile не открывался с флажком FILE_FLAG_OVERLAPPED, а lpOverlapped — значение NULL, операция записи начинается с текущей позиции в файле и WriteFile не возвращает значения до тех пор, пока операция не будет закончена. Система после завершения операции модернизирует указатель позиции в файле. Функция WriteFile сбрасывает событие, заданное членом hEvent структуры OVERLAPPED в несигнальное состояние, когда она начинает операцию ввода-вывода. Поэтому, нет какой-либо необходимости вызывающей программе проделывать эту процедуру.

Возвращаемые значения

Если функция завершается успешно, величина возвращаемого значения — не ноль. Если функция завершается с ошибкой, величина возвращаемого значения — ноль. Чтобы получить дополнительные сведения об ошибке, вызовите GetLastError.

2.1.5.5 Функция SetCommState

Функция SetCommState конфигурирует коммуникационное устройство согласно определениям в управляющем устройством блоке (структура DCB). Функция повторно инициализирует все аппаратные и управляющие настройки, но не опорожняет очереди вывода или ввода данных.

Синтаксис

```
BOOL SetCommState(  
    HANDLE hFile,  
    LPDCB lpDCB  
);
```

Параметры

а) `hFile` — дескриптор коммуникационного устройства. Функцией `CreateFile` возвращается этот дескриптор.

б) `lpDCB` — указатель на структуру `DCB`, которая содержит информацию о конфигурации заданного коммуникационного устройства.

Возвращаемые значения

Если функция завершается успешно, возвращаемое значение не ноль. Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке, вызовите `GetLastError`.

Замечания

Функция `SetCommState` использует структуру `DCB`, чтобы установить требуемую конфигурацию. Функцией `GetCommState` возвращается текущая конфигурация. Чтобы установить только несколько членов структуры `DCB`, вам следует изменить структуру `DCB`, которая заполнялась вызовом функции `GetCommState`. Это гарантирует то, что остальные члены структуры `DCB` имеют соответствующие значения. Функция `SetCommState` завершается ошибкой, если член `XonChar` структуры `DCB` равен `XoffChar` члену этой же структуры. Когда используется функция `SetCommState`, чтобы конфигурировать 8250, к значениям членов `ByteSize` и `StopBits` структуры `DCB` применяют ниже перечисленные ограничения: Число битов данных должно быть 5–8 битов.

2.1.5.6 Функция *GetCommState*

Функция *GetCommState* извлекает данные о текущих настройках управляющих сигналов для указанного коммуникационного устройства.

Синтаксис

```
BOOL GetCommState(  
    HANDLE hFile,  
    LPDCB lpDCB  
);
```

Параметры

а) *hFile* — дескриптор коммуникационного устройства. Функцией *CreateFile* возвращается этот дескриптор.

б) *lpDCB* — указатель на структуру *DCB*, которая получает информацию о настройках управляющих сигналов.

Возвращаемые значения

Если функция завершается успешно, возвращаемое значение не ноль. Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке, вызовите *GetLastError*.

2.1.5.7 Структура *DCB*

Структура *DCB* определяет настройки управления последовательным коммуникационным устройством.

Структура данных

```
typedef struct _DCB {  
    DWORD   DCBlength;  
    DWORD   BaudRate;  
    DWORD   fBinary:1;  
    DWORD   fParity:1;  
    DWORD   fOutxCtsFlow:1;  
    DWORD   fOutxDsrFlow:1;  
    DWORD   fDtrControl:2;  
    DWORD   fDsrSensitivity:1;  
    DWORD   fTXContinueOnXoff:1;  
    DWORD   fOutX:1;  
    DWORD   fInX:1;  
    DWORD   fErrorChar:1;  
    DWORD   fNull:1;  
    DWORD   fRtsControl:2;  
    DWORD   fAbortOnError:1;  
    DWORD   fDummy2:17;  
    WORD    wReserved;  
    WORD    XonLim;  
    WORD    XoffLim;  
    BYTE    ByteSize;  
    BYTE    Parity;  
    BYTE    StopBits;  
    char    XonChar;  
    char    XoffChar;  
    char    ErrorChar;  
    char    EofChar;  
    char    EvtChar;  
    WORD    wReserved1;  
} DCB;
```

Члены структуры

- 1) DCBlength — длина структуры, в байтах.
- 2) BaudRate — скорость передачи данных, в бодах, с которой работает коммуникационное устройство. Этот член структуры может быть фактическим зна-

чением скорости передачи данных в бодах, или одним из ниже перечисленных индексов:

- CBR_110;
- CBR_19200;
- CBR_300;
- CBR_38400;
- CBR_600;
- CBR_56000;
- CBR_1200;
- CBR_57600;
- CBR_2400;
- CBR_115200;
- CBR_4800;
- CBR_128000;
- CBR_9600;
- CBR_256000;
- CBR_14400.

3) fBinary — если этот член структуры — TRUE, включается двоичный режим. Windows не поддерживает недвоичный режим передачи, так что этот член структуры должен быть TRUE.

4) fParity — если этот член структуры — TRUE, выполняется проверка четности и сообщается об ошибках.

5) fOutxCtsFlow — если этот член структуры — TRUE, то проверяется сигнал готовности к приему (CTS) для управления потоком вывода данных. Если этот член структуры — TRUE, а сигнал готовности к приему (CTS) выключен, вывод данных приостанавливается до тех пор, пока сигнал готовности к приёму (CTS) не отправляется снова.

6) fOutxDsrFlow — если этот член структуры — TRUE, то проверяется сигнал готовности модема (DSR) для управления потоком вывода данных. Если этот член структуры — TRUE, а сигнал готовности модема (DSR) отключается, вывод данных приостанавливается до тех пор, пока сигнал готовности модема (DSR) не отправляется снова.

7) fDtrControl — сигнал DTR (готовности терминала к передаче данных) управления потоком данных. Этот член структуры может быть одним из следующих значений:

- DTR_CONTROL_DISABLE — отключает линию DTR, когда устройство открывается и оставляет ее заблокированной;
- DTR_CONTROL_ENABLE — включает линию DTR, когда устройство открывается и оставляет ее включенной;
- DTR_CONTROL_HANDSHAKE — включает процедуру установления связи DTR. Если процедура установления связи включена, она является ошибкой для приложения, которое корректировать линию, используя функцию EscapeCommFunction.

8) fDsrSensitivity — если этот член структуры — TRUE, коммуникационный драйвер чувствителен к состоянию сигнала готовности модема (DSR). Драйвер игнорирует любые принимаемые байты, если сигнал DSR модемной линии ввода данных не высокий.

9) fTXContinueOnXoff — если этот член структуры — TRUE, то передача продолжается и после того, как байты заполнения буфера ввода данных достигают XoffLim, а драйвер передал символ члена XoffChar, чтобы остановить прием байтов. Если этот член структуры — FALSE, передача не продолжается до тех пор, пока выгружаемые байты буфера ввода данных не достигнут XonLim, а драйвер не передаст символ члена структуры XonChar, чтобы возобновить приём.

10) fOutX — указывает, используется ли XON/XOFF управление потоком данных в ходе передачи. Если этот член структуры — TRUE, передача останавливается, когда принимается символ члена структуры XoffChar и начинается снова, когда принят символ члена XonChar.

11) fInX — указывает, используется ли XON/XOFF управление потоком данных в ходе приема. Если этот член структуры — TRUE, символ члена структуры

XoffChar отправляется тогда, когда заполняемые байты буфера ввода данных достигают величины XoffLim, а символ члена XonChar отправляется тогда, когда выгружаемые байты буфера ввода данных находятся в пределах величины XonLim.

12) fErrorChar — указывает, заменяются ли байты, принятые с ошибками четности символом, определенным членом структуры ErrorChar. Если этот член структуры — TRUE, и член структуры fParity — TRUE, замена происходит.

13) fNull — если этот член структуры — TRUE, при приеме пустые байты сбрасываются.

14) fRtsControl — сигнал RTS (готовности к передаче) управления потоком данных. Этот член структуры может быть одним из следующих значений:

- RTS_CONTROL_DISABLE — отключает линию RTS, когда устройство открывается и оставляет ее отключенной;

- RTS_CONTROL_ENABLE — включает в работу линию RTS, когда устройство открывается и оставляет ее включенной;

- RTS_CONTROL_HANDSHAKE — включает процедуру установления связи RTS. Драйвер поднимает линию RTS, когда (входной) буфер "опережающего ввода с клавиатуры" заполнен меньше, чем на половину и понижает линию RTS, когда буфер заполнен больше, чем на три четверти. Если процедура установления связи разрешается, то это — ошибка для прикладной программы, которая корректирует линию, используя функцию EscapeCommFunction;

- RTS_CONTROL_TOGGLE — определяет, что линия RTS должна быть поднята, если байты доступны для передачи. После того, как все буферизированные байты отправлены, линия RTS должна быть опущена.

15) fAbortOnError — если этот член структуры — TRUE, драйвер завершает все операции чтения и записи с состоянием ошибки, если происходит ошибка. Драйвер не будет допускать любую дальнейшую коммуникационную операцию до тех пор, пока приложение не подтвердит ошибку при помощи вызова функции ClearCommError.

16) fDummy2 — зарезервированный; не используется.

17) wReserved — зарезервированный; должен быть ноль.

18) XonLim — минимальное число байтов, которое допустимо в буфере ввода данных перед активизацией управления потоком данных, когда их задерживает отправитель. Обратите внимание на то, что то, что отправитель может передать символы после того, как стал активным сигнал управления потоком данных, так что это значение никогда не должно равняться нулю. Это предполагает, что или XON/XOFF, RTS, или DTR сигнал управления потоком данных устанавливается в членах структуры fInX, fRtsControl или fDtrControl.

19) XoffLim — максимальное число байтов, допустимое в буфере ввода данных перед активизацией управления потоком данных, чтобы дать возможность осуществить передачу отправителю. Это предполагает, что или XON/XOFF, RTS, или DTR сигнал управления потоком данных устанавливаются в членах структуры fInX, fRtsControl или fDtrControl. Максимальное допустимое число байтов рассчитывается, путем вычитания этого значения из размера буфера ввода данных, в байтах.

20) ByteSize — число переданных и принятых битов, в байтах.

21) Parity — используемая схема четности. Этот член структуры может быть одним из следующих значений:

- EVENPARITY — проверка по четности;
- MARKPARITY — проверка четности по метке;
- NOPARITY — без проверки четности;
- ODDPARITY — проверка по нечетности;
- SPACEPARITY — проверка четности по паузе.

22) StopBits — число используемых стоповых битов. Этот член структуры может быть одним из следующих значений:

- ONESTOPBIT — 1 стоповый бит;
- ONE5STOPBITS — 1.5 стоповых бита;
- TWOSTOPBITS — 2 стоповых бита.

23) XonChar — значение символа XON и для передачи и для приема.

24) XoffChar — величина символа XOFF и для передачи и для приема.

25) ErrorChar — значение символа, используемого для замены байтов, принятых с ошибкой четности.

26) EofChar — значение символа, используемого для сигнала о конце данных.

27) EvtChar — значение символа, используемого, чтобы предупредить о событии.

28) wReserved1 — зарезервирован; не используется.

Замечания

Когда используется структура DCB, чтобы конфигурировать 8250, применяют ниже перечисленные ограничения к значениям, указанным в членах ByteSize и StopBits: число битов данных должно быть от 5 до 8 битов.

2.1.5.8 Функция SetCommTimeouts

Функция SetCommTimeouts устанавливает параметры простоя для всех операций чтения и записи для заданного коммуникационного устройства.

Синтаксис

```
BOOL SetCommTimeouts(  
    HANDLE hFile,  
    LPCOMMTIMEOUTS lpCommTimeouts  
);
```

Параметры

а) hFile — дескриптор коммуникационного устройства. Функция CreateFile возвращает этот дескриптор.

б) lpCommTimeouts — указатель на структуру COMMTIMEOUTS, которая содержит новые значения о простое.

Возвращаемые значения

Если функция завершается успешно, возвращаемое значение не ноль. Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке, вызовите GetLastError.

2.1.5.9 Структура COMMTIMEOUTS

Структура COMMTIMEOUTS используется в функциях SetCommTimeouts и GetCommTimeouts, чтобы установить и сделать запрос параметров интервала простоя по времени для коммуникационного устройства. Параметры устанавливают характер работы функций ReadFile, WriteFile, ReadFileEx и WriteFileEx на устройстве.

Структура данных:

```
typedef struct _COMMTIMEOUTS {  
    DWORD ReadIntervalTimeout;  
    DWORD ReadTotalTimeoutMultiplier;  
    DWORD ReadTotalTimeoutConstant;  
    DWORD WriteTotalTimeoutMultiplier;  
    DWORD WriteTotalTimeoutConstant;  
} COMMTIMEOUTS,  
*LPCOMMTIMEOUTS;
```

Члены структуры

1) ReadIntervalTimeout — максимальное время, которое допускается для интервала между поступлением двух символов в коммуникационную линию, в миллисекундах. В ходе операции ReadFile, период времени начинается, когда получен первый символ. Если интервал между поступлением любых двух символов будет больше этой величины, операция ReadFile завершается и любые буферизированные данные возвращаются. Значение нуля указывает, что интервал простоя по по

времени не используются. Значение MAXDWORD, объединенное с нулевыми значениями и для члена ReadTotalTimeoutConstant, и для члена ReadTotalTimeoutMultiplier определяет, что операция чтения должна немедленно вернуть значение с символами, которые были уже получены, даже если никаких символов не было принято.

2) ReadTotalTimeoutMultiplier — множитель, используемый для вычисления полного периода времени простоя для операций чтения, в миллисекундах. Для каждой операции чтения, это значение умножается на затребованное число байтов, которые читаются.

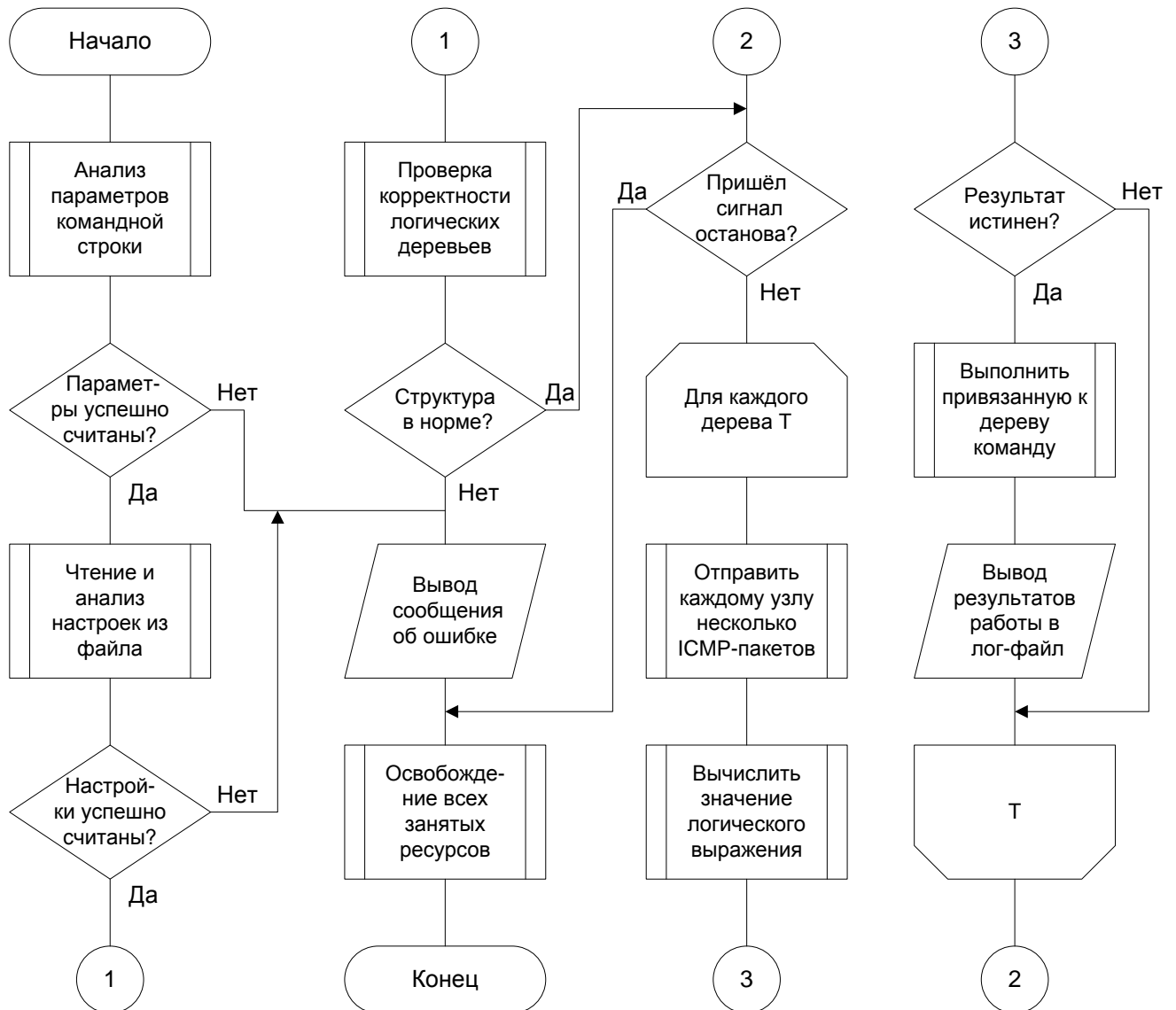
3) ReadTotalTimeoutConstant — константа, используемая, чтобы вычислить полный период времени простоя для операций чтения, в миллисекундах. Для каждой операции чтения, это значение добавляется к произведению члена структуры ReadTotalTimeoutMultiplier и прочитанного числа байтов. Значение нуля и для члена ReadTotalTimeoutMultiplier, и для члена ReadTotalTimeoutConstant указывает, что полное время простоя не используется для операций чтения.

4) WriteTotalTimeoutMultiplier — множитель, используемый для вычисления полного периода времени простоя для операций записи, в миллисекундах. Для каждой операции записи, это значение умножается на число записываемых байтов.

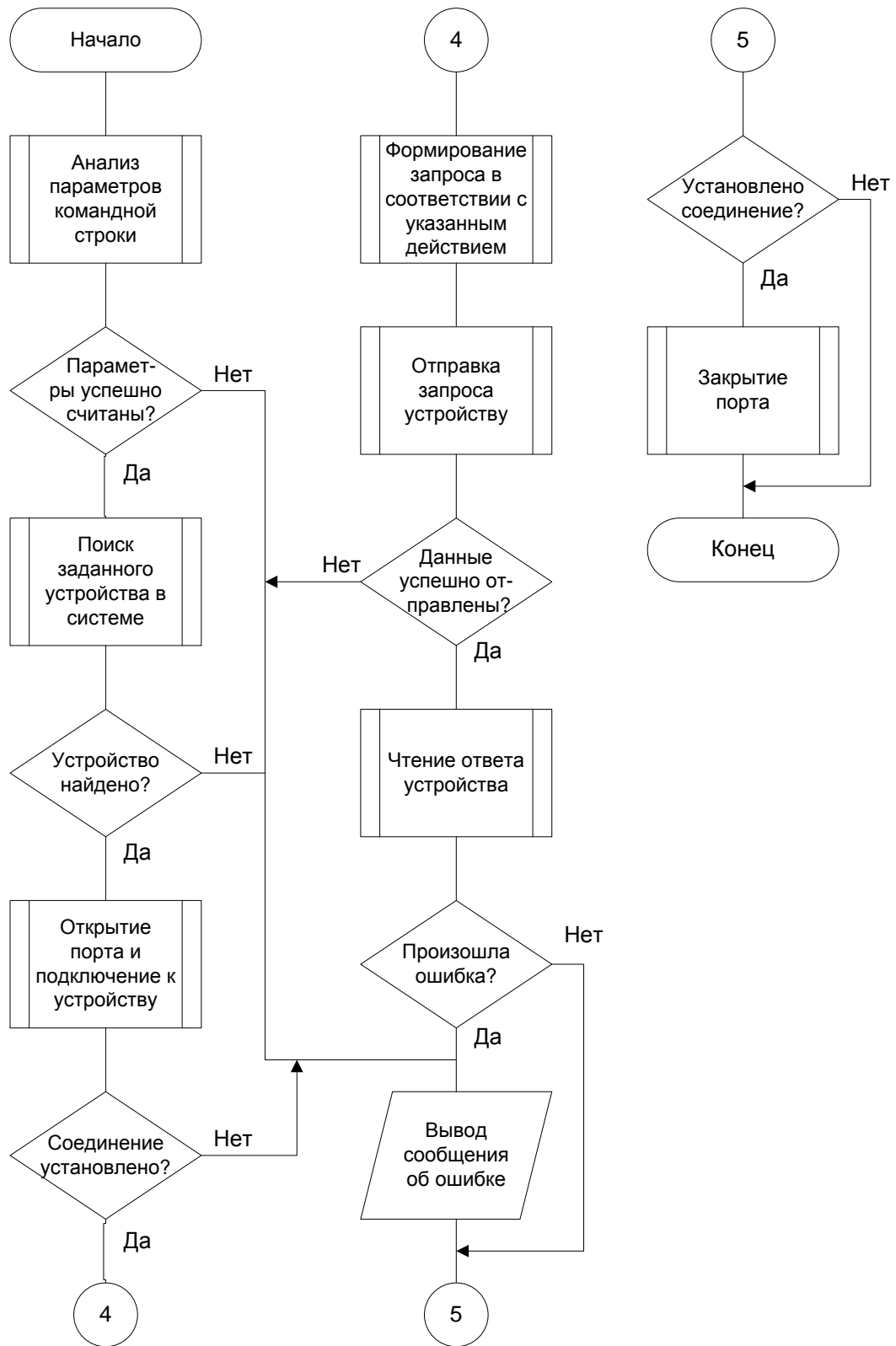
5) WriteTotalTimeoutConstant — константа, используемая, чтобы вычислить полный период времени простоя для операций записи, в миллисекундах. Для каждой операции чтения, это значение добавляется к произведению члена структуры WriteTotalTimeoutMultiplier и записанного числа байтов. Значение нуля и для члена WriteTotalTimeoutMultiplier, и для члена WriteTotalTimeoutConstant указывает, что полное время простоя не используется для операций записи.

2.2 Схемы алгоритмов программ

2.2.1 Схема алгоритма программы keusbd



2.2.2 Схема алгоритма программы keusb



2.3 Отладка программ

Отладка — процесс поиска и устранения ошибок в программе. Она занимает значительную часть рабочего времени программиста, нередко большую, чем составление самой программы.

Ошибки программного обеспечения делятся на:

а) технологические ошибки — ошибки документации и фиксирования программ в памяти ЭВМ. Они составляют 5–10% от общего числа ошибок, обнаруженных при отладке;

б) программные ошибки:

1) синтаксические — состоят в нарушении формальных правил написания программы и появляются в результате недостаточного знания пользователем языка программирования и невнимательности при технической подготовке программы к обработке в ЭВМ. Эти ошибки обычно выявляются во время компиляции. Большинство из них можно выловить во время проверки, когда программист тщательно проверяет листинг своей программы, выявляя орфографические ошибки, недопустимые форматы команд, неопределённые переменные и так далее;

2) семантические (логические) — причинами таких ошибок являются несоответствие алгоритма поставленной задаче, неправильное понимание программистом смысла (семантики) операторов языка программирования, нарушение допустимых пределов и правил представления данных. Эти ошибки устраняются обычно посредством выполнения программы с тщательно подобранными проверочными данными, для которых известен правильный ответ, полученный либо с помощью ручного подсчёта, либо с помощью досконально проверенных вычислений на машине. Для сложных программ необходимо применять тестирование. Если программа состоит из нескольких независимых модулей, то каждый модуль должен быть независимо проверен для всех возможных комбинаций;

3) алгоритмические ошибки — ошибки, обусловленные некорректной постановкой задачи, такие ошибки составляют 6–8% от общего числа;

4) системные ошибки — возникают из-за неполной информации о реальных процессах, происходящих в источниках и потребителях информации. Обычно в начале отладки доля этих ошибок около 10%, но она существенно возрастает (35–40%) на завершающих этапах;

В процессе разработки возникали ошибки синтаксического характера, такие как отсутствие точки с запятой. В процессе тестирования оборудования была обнаружена ошибка в документации к устройству: в примере использования команды \$KE,RDR в документации к устройству неправильно описан формат возвращаемых значений, что вызвало некоторые трудности при составлении алгоритма обработки результата:

Ошибочные данные: #RID,<LineNumber>,<Value>.

Исправленные данные: #RDR,<LineNumber>,<Value>.

В результате предварительных испытаний в виде месяца непрерывной работы с устройством Ke-USB24R, программа показала очень стабильную работу на протяжении всего времени испытаний.

Листинг программы приведён в приложении А, а результаты выполнения в приложении Б.

2.4 Инструкция по эксплуатации

2.4.1 Назначение программ

В задачу программы keusbd входит циклическая проверка доступности заданных частей сети, путём отправки служебных ICMP-запросов. В случае недоступности, с помощью программы keusb, в сетевом оборудовании, ответственном за связь с данной подсетью, на несколько секунд разрывается электрическая цепь, осуществляя тем самым жёсткий сброс виновного устройства. Вновь получив питание, устройство заново проводит инициализацию, самотестирование и восстанавливает соединение.

2.4.2 Условия выполнения

- а) ЭВМ на базе процессора типа x86, amd64, sparc, ppc, ppc64, alpha, hppa, mips, ia64 или arm;
- б) 8 мегабайт оперативной и 50 кибайт постоянной памяти;
- в) POSIX-совместимая операционная система с установленным пакетом ip-utils (среди них BeOS, Mac OS X, OpenSolaris, OpenVMS, QNX, BeOS, FreeBSD и GNU/Linux) или операционная система Windows (поддерживается Windows 95 и выше);
- г) наличие интерфейса USB и драйвера виртуального серийного порта;
- д) устройство Ke-USB24R;
- е) наличие соединения прибора и интерфейса USB-кабелем типа “Б”.

2.4.3 Подготовка к запуску

- а) Подключите устройство Ke-USB24R и, в случае необходимости, настройте его (подробнее смотрите в инструкции к устройству).
- б) Установите данное программное обеспечение. Установка может производиться тремя различными путями:

1) Произведите установку средствами Вашего пакетного менеджера (например: “emerge keusb”). Этот способ является наиболее предпочтительным.

2) Пару файлов keusb и keusbd, соответствующую Вашей архитектуре, скопируйте в любую стандартную директорию, например /usr/bin или C:/Windows, сценарий инициализации поместите в директорию /etc/init.d, конфигурационный файл в /etc/conf.d.

3) В каталоге с исходными кодами выполните команду “make all”, а затем от суперпользователя “make install” (или произведите установку вручную согласно второму пункту).

в) Настройте программу.

2.4.4 *Настройка программы*

Настройки программы хранятся в файле /etc/conf.d/keusb (или в файле keusbd.conf в каталоге с программой в случае работы под ОС Windows).

Конфигурационный файл представляет из себя список директив, которые условно можно разделить на три группы: установка основных параметров работы программы, дерево анализа сети, узлы которого суть логические операции или адреса тестируемых узлов и закреплённые за этими деревьями команды.

Всего поддерживается два задаваемых параметра: количество посылаемых ICMP-пакетов за одну сессию и продолжительность промежутка времени простоя после завершения анализа; директивы установки параметров начинаются с восклицательного знака, после которого идёт название параметра и его устанавливаемое значение. Для значений временных интервалов существует система постфиксов: s — для секунд, m — для минут и h — для часов. В качестве значений по умолчанию используется отсылка 10 пакетов и 5 минут ожидания.

Далее задаётся действие, представляющее из себя синтаксически верную последовательность символов, исполняемую системным командным интерпретатором, записанное после знака процента. На следующей строке пишется условие выполнения этого действия. Стоит отметить, что связь действия с условием является обязательной, в противном случае конфигурационный файл будет считаться неверным с точки зрения семантики.

Условие представляет собой дерево логических выражений, записанных в префиксной нотации (вместо “X and Y” будет “and X Y”). Поддерживаются следующие логические операции: логическое И, логическое ИЛИ, логическое НЕ и логическое исключающее ИЛИ. В качестве операндов выступают адреса тестируемых узлов (поддерживаются как символические, так и IP-адреса).

Количество пробельных символов в любой части конфигурационного файла не играет никакой роли, таким образом, длинное условие можно разбить на части — по одной на строку, выравняв отступами которые, можно добиться повышения зрительного восприятия структуры условия. Также в любом месте файла могут быть вставлены однострочные комментарии, начинающиеся со знака решётки.

Рассмотрим пример типичного конфигурационного файла:

```
# Комментарии начинаются со знака решётки

! tries 5                # Пять пробных пакетов
! wait 3m                # Ждать три минуты

% keusb reset 2          # Сброс второго устройства,
or                        # если или
    and                  # одновременно
        192.168.1.1      # недоступен первый,
        not 192.168.1.2  # но доступен второй,

    xor                  # или недоступен только
        8.8.8.8          # один из перечисленных
        mail.google.com  # (один работает, другой нет)

% keusb reset 4          # Сброс четвёртого устройства,
www.kernel.org          # если не доступен заданный адрес

# Конец конфигурационного файла
```

2.4.5 *Запуск программы*

Работа с обеими программами осуществляется через эмулятор терминала (командную строку) по трём существенным причинам:

а) данные программы будут использованы на серверах, 90 процентов которых попросту не имеет оконной подсистемы для отрисовки графического пользовательского интерфейса (далее: GUI);

б) разработка GUI никоим образом не упростит логику настройки и запуска программ;

в) целевая группа людей, работающих с данной программой — высококвалифицированные специалисты, которые могут без проблем разобраться со спецификой работы подобного рода программ.

В случае наличия централизованного управления демонами (openrc, upstart, initng и тому подобные), вся работа с демоном осуществляется через системное окружение и сценарий инициализации. Ниже представлен ряд команд для эффективного управления (все команды должны быть выполнены с правами суперпользователя):

- **/etc/init.d/keusbd start** — запуск демона;
- **/etc/init.d/keusbd stop** — останов демона;
- **/etc/init.d/keusbd restart** — перезапуск демона;
- **rc-update add keusbd default** — включение автозагрузки;
- **rc-update del keusbd default** — выключение автозагрузки.

Учитывая, что операционная система Windows не предоставляет такого функционала, предусмотрен ряд опций — аргументов командной строки, передаваемых демону при запуске. Они в себя включают:

- **--help** — вывод справки по опциям;
- **--version** — вывод версии программы;
- **--daemon** — запуск демона;

- **-f <Файл>** — использовать Файл в качестве конфигурационного файла;
- **-l <Файл>** — использовать Файл в качестве лог-файла;
- **-p <Файл>** — использовать Файл в качестве файла-защёлки;
- **-c** — проверка конфигурационного файла и выход;
- **-d** — тоже самое, что и **-daemon**;
- **-k** — останов демона;
- **-ay** — включение автозагрузки (только для ОС Windows);
- **-an** — выключение автозагрузки (только для ОС Windows).

Для управления устройством Ke-USB24R используется программа keusb. Для связи с устройством ей должен быть передан идентификатор, который может из себя представлять:

- адрес виртуального порта, к которому подключено устройство (например, “com5:”);
- блочное устройство (например, “/dev/ttyUSB0”);
- серийный номер вида “xxxx-xxxx-xxxx-xxxx”.

Если в системе зарегистрировано лишь одно устройство, данный параметр является необязательным и может опускаться.

Следующими параметрами должны идти команды управления устройством и состоянием реле определённого канала. Они в себя включают:

- **--help** — вывод справки по опциям;
- **--version** — вывод версии программы;
- **turn_on <N>** — включение питания на канале N;
- **turn_off <N>** — выключение питания на канале N;
- **toggle <N>** — переключение состояния реле на канале N;
- **reset <N>** — сброс питания на 5 секунд на канале N;
- **reset <N> <K>** — сброс питания на K секунд на канале N;
- **reset -1** — жёсткий сброс устройства;

- **status** — вывод информации об устройстве;
- **status <N>** — вывод состояния реле на канале N;
- **status all** — вывод состояния всех реле.

Примеры команд:

- **keusb reset 2 15** — пятнадцатисекундный сброс питания на 2 канале на первом зарегистрированном в системе устройстве;
- **keusb /dev/ttyACM1 toggle 1** — переключение состояния реле на первом канале устройства, привязанного к блочному файлу /dev/ttyACM1;
- **keusb 25fc-q13f-dt7h-56ee reset -1** — жёсткий сброс устройства с данным серийным номером.

2.4.6 Сообщения и ошибки

Программа **keusb** может отслеживать появление ряда ошибок, которые будут незамедлительно выведены в текущий сеанс терминала, после чего работа программы сразу же прекратится. Ниже представлены возможные сообщения об ошибках:

- **cmdline_parse: missing operand** — пропущен необходимый параметр;
- **keusb_connect: can't open device** — невозможно подключиться к устройству;
- **keusb_connect: can't find device** — невозможно найти искомое устройство;
- **keusb_main: status failed** — устройство вернуло неверное состояние реле;
- **keusb_main: failed** — устройство получило или вернуло неверные данные;

Остальные сообщения имеют вид “**keusb_XXXXX: request failed**”, где “XXXXX” — имя функции модуля **device**, в которой произошёл сбой (устройство получило или вернуло неверные данные).

Все сообщения демона **keusbd** делятся на два вида: сообщения, выводимые в терминал и сообщения, выводимые в специальный лог-файл. К первому виду

относятся критические ошибки, возникающие до демонизации. В лог-файл идут все сообщения, сопутствующие работе демона, как то: уведомление о выполнении действия из конфигурационного файла с дальнейшим перенаправленным выводом запускаемой команды, а также предупреждения, связанные с невозможностью совершения того или иного действия, некритичные для дальнейшей работы программы. Ниже представлен список всех ошибок и предупреждений:

- **xalloc: can't allocate memory** — невозможно выделить память;
- **cmdline_parse: missing operand** — пропущен необходимый параметр;
- **cmdline_parse: invalid argument** — передан неверный параметр;
- **daemonize: daemon is already running** — одна копия демона уже запущена;
- **daemon_kill: it seems daemon isn't running** — невозможно найти процесс демона;
- **daemon_kill: can't kill process** — невозможно завершить процесс демона;
- **logger_init: can't open file** — невозможно открыть лог-файл;
- **daemon_pid: can't open pid file** — невозможно открыть файл-защёлку;
- **config_init: can't open config** — невозможно открыть конфигурационный файл;
- **config_parse: argument not found** — отсутствует значение параметра;
- **config_parse: invalid argument** — неверное значение параметра;
- **config_parse: invalid postfix** — неверный постфикс у временного интервала;
- **config_parse: invalid option** — неверный параметр.
- **rules_add: missing action** — условия прикреплены к отсутствующему действию;
- **rules_exec: unlinked expression** — к действию не прикреплено условие выполнения;
- **rules_test_expr: missing argument** — пропущен аргумент у логической операции;

- **rules_exec: can't open pipe** — невозможно перехватить вывод запускаемой программы (является предупреждением и не останавливает работу программы);
- **icmp_init: can't initialize winsock** — невозможно подключиться к системной сетевой службе;
- **icmp_init: unable to load icmp.dll** — невозможно загрузить системную библиотеку icmp.dll;
- **icmp_init: cannot find icmp functions** — в данной версии библиотеки не найдены необходимые функции;
- **icmp_init: unable to open ping service** — невозможно открыть сокет для передачи данных.

Файл-защёлка keusbd.pid, расположенный в /var/run (в ОС Windows он находится в текущем каталоге с программой), предназначен для служебных нужд и ручной правке не подлежит.

В лог-файл keusbd.log (он находится в /var/log или текущем каталоге с программой) производится запись всех событий, произошедших за время работы программы с указанием даты и времени возникновения события. Ниже представлена часть реального лог-файла:

```
(II) Mar 10 03:18:22 > keusbd started
(==) Mar 10 08:03:31 > keusb reset 4
(==) Mar 13 06:48:04 > keusb reset 4
(==) Mar 15 01:59:47 > keusb reset 4
(==) Mar 17 05:46:47 > keusb reset 2
(==) Mar 17 12:37:48 > keusb reset 2
(==) Mar 18 03:45:34 > keusb reset 4
(==) Mar 19 01:30:57 > keusb reset 4
(==) Mar 22 12:23:59 > keusb reset 1
(==) Mar 24 07:23:37 > keusb reset 4
(==) Mar 24 18:58:17 > keusb reset 4
(II) Mar 28 17:59:34 > keusbd stopped
```

2.4.7 *Завершение работы программы*

Останов демона `keusbd` может быть осуществлён двумя путями:

- **`sudo /etc/init.d/keusbd stop`** — через систему управления сервисами;
- **`keusbd -k`** — напрямую, в случае невозможности первого варианта.

Программа `keusb` не требует принудительного останова по причине мгновенного завершения работы после получения информации от устройства (в среднем на работу этой программы уходит 0.5–1.5 секунды). В случае неисправности оборудования, она может довольно долго ждать окончания чтения данных, получаемых с устройства, и поэтому может быть принудительно завершена сигналом SIGTERM.

3 ОХРАНА ТРУДА

3.1 Введение

Охрана труда — это система законодательных актов, социально-экономических, организационных, технических, гигиенических и лечебно-профилактических мероприятий и средств, обеспечивающих безопасность, сохранение здоровья и работоспособности человека в процессе труда.

В практической деятельности принято рассматривать законодательные акты и социально-экономические мероприятия в понятиях трудовое законодательство и управление охраной труда; технические мероприятия и средства — техника безопасности и система противопожарной защиты; гигиенические и лечебно-профилактические мероприятия и средства — гигиена труда и производственная санитария; организационные мероприятия — в понятиях как техники безопасности и производственной санитарии, так и управления охраной труда.

Все многообразие законодательных актов, мероприятий и средств, включенных в понятие охраны труда, направлено на создание таких условий труда, при которых исключено воздействие на работающих опасных и вредных производственных факторов.

Опасный производственный фактор — производственный фактор, воздействие которого на работающего в определённых условиях приводит к травме или другому внезапному резкому ухудшению здоровья. К резкому ухудшению здоровья можно отнести отравление, облучение, тепловой удар и прочее.

Вредный производственный фактор — производственный фактор, воздействие которого на работающего в определённых условиях приводит к заболеванию или снижению работоспособности. В зависимости от уровня и продолжительности воздействия, вредный производственный фактор может стать опасным.

Трудовое законодательство включает мероприятия правового порядка. К правовым вопросам относят обязанности администрации предприятий и права рабочих и служащих на обеспечение здоровых и безопасных условий труда.

Управление охраной труда — целенаправленное воздействие органов управления на охрану труда. Управление охраной труда осуществляет функции планирования, контроля, информации, управляющего воздействия и стимулирования.

Техника безопасности — система организационных мероприятий и техниче-

ских средств, предотвращающих воздействие на работающих опасных производственных факторов. Техника безопасности предусматривает обеспечение безопасности производственного оборудования и производственных процессов; внедрение новых машин механизмов, инструмента, сконструированных с соблюдением всех требований охраны труда; установку оградительных и блокирующих устройств; внедрение автоматической сигнализации, обеспечивающей безопасные условия на рабочих местах; применение средств коллективной и индивидуальной защиты и прочее. Техника безопасности изучает производственное оборудование, трудовые процессы и производственные условия, а также определяет правила поведения людей на работе.

Система противопожарной защиты — совокупность организационных мероприятий и технических средств, направленных на предотвращение воздействия на людей опасных факторов пожара и ограничение материального ущерба от него.

Гигиена труда — наука, изучающая влияние на организм работающих производственной среды и трудового процесса. На основе изучения технологии производства и трудовых процессов, вредного воздействия на организм человека применяемого сырья, материалов, полуфабрикатов и производственных отходов разрабатывают гигиенические и лечебно-профилактические нормативные и рекомендательные мероприятия, направленные на оздоровление условий труда, на охрану здоровья работающих и повышение производительности труда.

Осуществление в конкретных практических условиях этих мероприятий обеспечивается мерами и средствами производственной санитарии.

Производственная санитария — система организационных мероприятий и технических средств, предотвращающих или уменьшающих воздействие на работающих вредных производственных факторов. Производственная санитария обеспечивает на рабочих местах нормальные условия воздушной среды, необходимую освещённость, устраняет вредное воздействие шума и вибрации на работающих, предусматривает оборудование на производстве санитарно-бытовых помещений и прочее.

3.2 Основные понятия гигиены, физиологии и психологии труда

Гигиена труда изучает влияние производственной среды на здоровье работающих. Для разработки рекомендаций, исключающих возможность неблагоприятного воздействия факторов внешней среды на организм работающих в ВЦ, необходимо тщательно изучить особенности технологического процесса, санитарно-гигиенические условия труда.

Физиология труда изучает изменения, происходящие в организме работающего под влиянием трудового процесса и внешней среды. Важнейшими задачами физиологии труда являются разработка наиболее рациональных трудовых приемов, обеспечивающих сохранение работоспособности и предупреждение утомления, а следовательно, повышение производительности труда; научное обоснование и рекомендации режимов труда и отдыха работающих и так далее.

Физиология труда находится в тесной взаимосвязи с психологией труда. Создание наиболее благоприятной в психологическом отношении обстановки на производстве и в быту способствует повышению работоспособности и производительности труда, снижению утомления. Психология труда изучает психологические особенности различных видов трудовой деятельности человека. На основе изучения закономерностей в психологической деятельности человека разрабатываются меры, способствующие улучшению трудового процесса.

Психология труда занимается вопросами оценки профессиональной пригодности работника, формирования профессиональной направленности, рационализации рабочей обстановки и рабочих мест, методов труда и обучения, взаимоотношений между людьми в процессе труда и прочим. Она тесно связана с гигиеной труда, врачебно-трудовой экспертизой, педагогикой и прочим.

Понижение работоспособности, возникающее в результате выполнения работы, называется утомлением. Это физиологическое состояние организма характеризуется рядом объективных признаков — изменением количества эритроцитов, лейкоцитов, гемоглобина, уменьшением содержания сахара в крови, повышением содержания молочной кислоты, субъективными ощущениями — нежеланием продолжать работу, усталостью и тому подобному.

Появление и развитие утомления связано с функциональными изменения-

ми, возникающими в процессе работы в центральной нервной системе, с тормозными процессами в коре головного мозга. Утомление, наступающее быстро, возникает в случаях отсутствия навыка в работе или при неблагоприятных условиях ее выполнения. В дальнейшем в процессе тренировки вырабатывается состояние высокой трудоспособности.

Медленно наступающее утомление — более глубокий процесс. Наступает оно при чрезмерно длительной работе, воздействии на организм работающего неблагоприятных факторов внешней среды, неправильной организации труда. Если за время, установленное для отдыха после работы, трудоспособность восстанавливается не полностью, то наступает так называемое переутомление. Причинами переутомления могут быть несоответствие между продолжительностью работы и временем отдыха, а также исходное состояние организма.

В борьбе с утомляемостью большое значение имеет физиологическая рационализация трудового процесса, которая включает разработку системы мер, касающихся экономии движений при работе, более равномерного распределения нагрузки между различными мышечными группами тела человека и прочим. Так, при работе сидя необходимо обращать внимание на правильную осанку, чтобы не было вынужденного положения, имелась возможность периодически менять позу.

Особое место в предупреждении утомления занимает построение физиологически обоснованного режима труда и отдыха, то есть рациональной системы чередования периодов работы и перерывов между ними. Наряду с пассивным отдыхом для предупреждения утомления в процессе труда применяют физические упражнения: производственную гимнастику, физкультурные паузы. Проводят их перед работой и в течение рабочего дня от одного до трех раз. Комплекс упражнений необходимо периодически изменять, так как в противном случае он перестает служить фактором, предупреждающим утомление.

На работоспособность человека влияют и неблагоприятные физические факторы внешней среды. К ним относятся микроклиматические условия, которые связаны со специфическими условиями производства. Наиболее часто изменения микроклимата в помещениях ВЦ вызываются повышением температуры. Температуру рабочих помещений рекомендуется регулировать в зависимости от времени года, тепловыделений и других факторов в соответствии с ГОСТ 12.1.005-76.

Работоспособность человека зависит, кроме того, от влажности и скорости движения воздуха, атмосферного давления, состава воздуха в помещениях, уровня

шума, освещенности, окраски оборудования, помещений и прочего.

Работа большинства сотрудников ВЦ связана с умственным трудом. Так, операторы ЭВМ, операторы по подготовке данных, программисты в течение рабочего дня должны воспринимать большой объем информации и быстро и точно на нее реагировать. Для предупреждения утомления и повышения работоспособности этих лиц в первую очередь необходимо установить наиболее рациональный режим труда и отдыха.

3.3 Техника безопасности при работе на компьютере

При разработке программного обеспечения основная часть рабочего времени проводится за компьютером, поэтому крайне важно уделить достаточное внимание вопросам обеспечения безопасности при работе с ЭВМ. Незнание мер безопасности или пренебрежение ими может привести к неприятным последствиям для здоровья оператора.

В первую очередь следует отметить нарушение зрения, утомление мышц рук и позвоночника, общую слабость.

Основные факторы вредного влияния компьютера на организм — это электромагнитные поля и излучения, в особенности, переменные низкочастотные магнитные поля; электронная развертка изображения и его мелькание на экране; длительная неподвижность оператора. Травмы повторяющихся нагрузок (ТПН) или, по другому, СДСН — синдром длительных статических нагрузок, возникают при частом и длительном выполнении одних и тех же движений. Предупредить воздействие этих факторов — значит сохранить здоровье.

Одним из важных факторов является производственный микроклимат, который характеризуется уровнем температуры, влажности воздуха, подвижностью воздуха и температурой окружающих поверхностей. Необходимая температура в лаборатории может поддерживаться с помощью системы вентиляции, отопления и кондиционера.

С целью создания нормальных условий установлены нормы производственного микроклимата по ГОСТ 12.1.005-88 Воздух в рабочей зоне. Значения тем-

пературы, относительной влажности, скорости движения воздуха удовлетворяют ГОСТу, но значения не оптимальны. Наиболее опасная ситуация связана с полями излучений очень низких частот, которые, как выяснилось способны вызвать биологические эффекты при воздействии на живые организмы.

Персональный компьютер конструктивно состоит из трех основных элементов: системного блока, клавиатуры и дисплея. Импульсный блок питания, входящий в состав системного блока, не представляет большой опасности, так как, чтобы не создавать помех, он тщательно экранируется. Наибольшую опасность для человека представляет дисплей с электронно-лучевой трубкой. Одной из наиболее вредных характеристик дисплея является излучение электромагнитного поля. Чтобы снизить уровень облучения переменными магнитными полями, следует расположить монитор так, чтобы расстояние до него составляло величину порядка 70 см. Поскольку магнитные поля сзади и по бокам большинства мониторов значительно сильнее, чем перед экраном, необходимо располагать свое рабочее место на расстоянии не менее 1.22 м от боковых и задних стенок других мониторов. При выборе оптимального положения дисплея относительно оператора рекомендуется использовать следующие условия:

- а) оператор должен находиться от экрана на расстоянии 75–90 см;
- б) при рассматривании изображения сбоку допустимый угол обзора составляет 45 градусов к нормали от поверхности экрана;
- в) не делать больше 160–200 нажатий на клавиши в минуту, что составляет около 1700 слов в час;
- г) каждый час 10–15 минут заниматься другой работой или просто отдыхать.

При выполнении этих условий оператор сможет сохранить своё здоровье и наиболее эффективно использовать рабочее время.

Организацию рабочих мест, при работе с персональным компьютером, необходимо осуществить на основе современных эргономических требований. Конструкция рабочей мебели (столы кресло или стулья) должна обеспечивать возможность индивидуальной регулировки соответственно росту работающего и создавать удобства.

Важную роль для работы оператора играет правильная компоновка клавиатуры. Неудачная организация клавиатуры либо неудобная конструкция способ-

на вызвать “накапливание” заболеваний сухожилий, мышц и нервных окончаний. Кроме того, возникновение болезней спины, шеи и рук специалисты объясняют тем, что при работе с компьютером пользователи с высокой скоростью повторяют одни и те же движения (нажатия на клавиши, перемещение, мыши, наклоны и повороты головы). Каждое нажатие на клавишу сопряжено с множественными сокращениями мышц, перемещением сухожилий вдоль кистей и соприкосновениями их с внутренними тканями. В итоге, из-за чрезмерной напряжённости, работы могут вызвать болезненные и воспалительные процессы.

При работе на ЭВМ есть опасность поражения электрическим током. Чтобы не попасть под действие электрического тока применяют следующие способы:

- а) недоступность токоведущих частей;
- б) защитное заземление;
- в) зануление;
- г) изоляция и двойная изоляция;
- д) защитное разделение цепей для борьбы с емкостными токами;
- е) защитное отключение.

В этой главе были рассмотрены вопросы, касающиеся безопасной работы с ЭВМ. Используя приведенные в главе рекомендации, оператор сможет сохранить свое здоровье в условиях воздействия электромагнитных излучений.

4 ЭКОНОМИЧЕСКАЯ ЧАСТЬ

4.1 Технико-экономическое обоснование

Данное программное обеспечение, состоящее из программ keusb и keusbd, предназначено для поддержания функционирования сетевой инфраструктуры путём автоматического управления электропитанием активного сетевого оборудования. В его задачу входит циклическая проверка доступности заданных частей сети, в случае недоступности в сетевом оборудовании, ответственном за связь с данной подсетью, на несколько секунд разрывается электрическая цепь. Вновь получив питание, устройство заново проводит инициализацию, самотестирование и восстанавливает соединение. Коммутирование цепей питания осуществляется через специальное устройство Ke-USB24R. Применение данного программного обеспечения значительно упрощает и удешевляет поддержание работоспособности локальной сети на предприятии.

Программа написана на языке C и выполняется на любом компьютере на базе процессора типа x86, amd64, sparc, ppc, ppc64, alpha, hppa, mips, ia64 или arm под управлением любой POSIX-совместимой операционной системой (также доступна поддержка ОС Windows).

Себестоимость — это затраты предприятия на изготовление и реализацию продукции, услуг и работ, выраженные в денежной форме.

Себестоимость можно рассчитать по следующей формуле:

$$C = M + \text{ПФ} + T_{\text{р.заг.}} + Z_{\text{осн.}} + Z_{\text{доп.}} + O_{\text{соц.}} + H_{\text{цех.}} + H_{\text{зав.}} + B_{\text{н.}},$$

где C — себестоимость в рублях;

M — стоимость материалов;

ПФ — стоимость полуфабрикатов;

$T_{\text{р.заг.}}$ — транспортно-заготовительные расходы;

$Z_{\text{осн.}}$ — основная заработная плата;

$Z_{\text{доп.}}$ — дополнительная заработная плата;

$O_{\text{соц.}}$ — отчисления на соцстрахование;

$H_{\text{цех.}}$ — накладные цеховые расходы;

$H_{\text{зав.}}$ — накладные заводские расходы;

$B_{\text{н.}}$ — внепроизводственные расходы.

4.2 *Расчёт трудоёмкости*

Для того чтобы определить себестоимость решения задачи, необходимо, прежде всего, найти трудоёмкость решения задачи.

Трудоёмкость — это сумма затрат труда (времени), необходимых для изготовления единицы продукции.

Трудоёмкость решаемой задачи можно определить по следующей формуле:

$$T_3 = T_{\text{и}} + T_{\text{а}} + T_{\text{бс}} + T_{\text{п}} + T_{\text{мр}} + T_{\text{отл}} + T_{\text{эвм}} + T_{\text{д}},$$

где T_3 — трудоёмкость в час;

$T_{\text{и}}$ — затраты труда на изучение материала, описание задачи;

$T_{\text{а}}$ — затраты труда на разработку алгоритмов решения задачи;

$T_{\text{бс}}$ — затраты труда на разработку схем алгоритмов программ;

$T_{\text{п}}$ — затраты труда на программирование;

$T_{\text{мр}}$ — затраты труда на машинно-ручные работы;

$T_{\text{отл}}$ — затраты труда на отладку программ;

$T_{\text{эвм}}$ — время машинного счёта на ЭВМ;

$T_{\text{д}}$ — затраты труда на оформление документации.

Слагаемые затрат труда определяются собственными наблюдениями через количество программных команд данной стадии разработки.

Затраты труда на изучение и описание задачи

Затраты труда на изучение и описание задачи определяются по формуле:

$$T_{\text{и}} = \frac{Q}{B \times K_{\text{кв}}} \times \beta,$$

где Q — предполагаемое число программных команд данной стадии разработки;

β — коэффициент, учитывающий качество описания задачи, $\beta = 1.2-1.5$;

B — производительность исполнителя, количество команд в час;

$K_{\text{кв}}$ — коэффициент квалификации исполнителя, $K_{\text{кв}} = 1.0$.

Предполагаемое число команд данной стадии разработки можно определить по следующей формуле:

$$Q = q \times K_{\text{сл.пр.}} \times t \times \left(1 + \sum_{i=1}^{n_k} P_i \right),$$

где q — предполагаемое число программных команд;

$K_{\text{сл.пр.}}$ — коэффициент, учитывающий сложность программы;

t — время работы программиста на данной стадии;

P_i — коэффициент коррекции программы при её разработке;

n_k — число коррекций в программе.

Рассчитаем время, затраченное на изучение и описание задачи по известной нам формуле:

$$q = 75; \quad t = 2; \quad P_i = 0.06; \quad B = 85;$$

$$K_{\text{сл.пр.}} = 1.25; \quad K_{\text{кв}} = 1.0; \quad n_k = 2; \quad \beta = 1.3.$$

$$Q = 75 \times 2 \times 1.25 \times \left(1 + \sum_{i=1}^2 0.06 \right) = 210 \text{ команд}$$

$$T_{\text{и}} = \frac{210 \times 1.3}{85 \times 1.0} = 3.21 \text{ часа}$$

Величины $T_{\text{а}}$, $T_{\text{бс}}$, $T_{\text{п}}$ и $T_{\text{отл}}$ вычисляются аналогичным способом по формуле:

$$T_j = \frac{Q}{B \times K_{\text{кв}}},$$

где вместо индекса j в каждом случае подставляется один из индексов рассчитываемых величин.

Затраты труда на разработку алгоритма решаемой задачи

Рассчитаем время, затраченное на разработку алгоритма решаемой задачи:

$$q = 20; \quad t = 12; \quad P_i = 0.06; \quad B = 25;$$

$$K_{\text{сл.пр.}} = 1.25; \quad K_{\text{кв}} = 1.0; \quad n_k = 3.$$

$$Q = 20 \times 12 \times 1.25 \times \left(1 + \sum_{i=1}^3 0.06 \right) = 354 \text{ команды}$$

$$T_{\text{а}} = \frac{354}{25 \times 1.0} = 14.16 \text{ часа}$$

Расчёт затрат труда на разработку схем алгоритмов программ

Рассчитаем время, затраченное на разработку блок-схемы программы:

$$q = 10; \quad t = 21; \quad P_i = 0.06; \quad B = 15;$$

$$K_{\text{сл.пр.}} = 1.25; \quad K_{\text{кв}} = 1.0; \quad n_k = 4.$$

$$Q = 10 \times 21 \times 1.25 \times \left(1 + \sum_{i=1}^4 0.06 \right) = 325.50 \text{ команд}$$

$$T_{\text{бс}} = \frac{325.5}{15 \times 1.0} = 21.70 \text{ часа}$$

Расчет затрат труда на программирование

Рассчитаем время, затраченное на программирование:

$$q = 20; \quad t = 32; \quad P_i = 0.06; \quad B = 25;$$

$$K_{\text{сл.пр.}} = 1.25; \quad K_{\text{кв}} = 1.0; \quad n_k = 13.$$

$$Q = 20 \times 32 \times 1.25 \times \left(1 + \sum_{i=1}^{13} 0.06 \right) = 1424 \text{ команды}$$

$$T_{\text{п}} = \frac{3364.20}{25 \times 1.0} = 56.96 \text{ часа}$$

Затраты труда на этапе отладки

Трудоёмкость на этапе отладки определяется по формуле:

$$T_{\text{отл}} = T_{\text{отл.исп.}} + T_{\text{отл.эвм}},$$

$$T_{\text{отл.исп.}} = \frac{Q}{B \times K_{\text{кв}}}$$

$$Q = q \times K_{\text{сл.пр.}} \times t \times \left(1 + \sum_{i=1}^{n_k} P_i \right)$$

$$q = 4; \quad t = 3.5; \quad P_i = 0.06; \quad B = 5;$$

$$K_{\text{сл.пр.}} = 1.25; \quad K_{\text{кв}} = 1.0; \quad n_k = 5.$$

$$Q = 23 \times 3.5 \times 1.25 \times \left(1 + \sum_{i=1}^5 0.06 \right) = 22.75 \text{ команд}$$

$$T_{\text{отл.исп.}} = \frac{22.75}{5 \times 1.0} = 4.55 \text{ часа}$$

Время отладки на ЭВМ определяется по статистическим данным:

$$T_{\text{отл.эвм}} = 0.14 + 0.23 + 0.09 + 0.03 + 0.01 = 0.5 \text{ ч}$$

$$T_{\text{отл}} = 4.55 + 0.5 = 5.05 \text{ ч}$$

Затраты труда, необходимые на оформление документации

Рассчитаем время, затраченное на оформление документации:

$$q = 15; \quad t = 10; \quad P_i = 0.06; \quad B = 20;$$

$$K_{\text{сл.пр.}} = 1.25; \quad K_{\text{кв}} = 0.8; \quad n_k = 5.$$

$$Q = 50 \times 27 \times 1.25 \times \left(1 + \sum_{i=1}^5 0.06 \right) = 243.75 \text{ команд}$$

$$T_{\text{д}} = \frac{243.75}{20 \times 1.0} = 12.19 \text{ часа}$$

Затраты труда на машинно-ручные работы

Трудоёмкость на этапе машинно-ручных работ определяется по формуле:

$$T_{\text{мр}} = \frac{Q_{\text{вв}}}{H_{\text{д}}},$$

где $Q_{\text{вв}}$ — объем вводимой информации в символах (123167 символов);

$H_{\text{д}}$ — норма производительности при занесении информации при помощи клавиатуры (около 3 символов за 1.5 сек).

$$T_{\text{мр}} = \frac{123167 \times 1.5}{3 \times 60 \times 60} = 17.11 \text{ часа}$$

Затраты труда, связанные со временем машинного счета на ЭВМ

Трудоёмкость на этапе машинного счета на ЭВМ определяется по формуле:

$$T_{\text{ЭВМ}} = T_{\text{ВВ}} + T_{\text{а}} + T_{\text{ВЫВ}},$$

где $T_{\text{ЭВМ}}$ — затраты труда на машинные операции;

$T_{\text{ВВ}}$ — затраты труда на ввод информации;

$T_{\text{ВЫВ}}$ — затраты труда на вывод информации;

$T_{\text{а}}$ — затраты труда на выполнение арифметических и логических операций.

Формула для расчета машинного времени, необходимого для ввода информации:

$$T_{\text{В}} = \sum_{i=1}^m \frac{Q_{ui}}{\sqrt{B_i}},$$

где $T_{\text{В}}$ — затраты труда на ввод/вывод информации;

m — количество устройств ввода/вывода;

Q_{ui} — объем вводимой/выводимой информации в i -е устройство ввода;

B_i — быстродействие i -го устройства ввода/вывода.

Рассчитаем затраты на ввод информации:

$$m = 1$$

$$Q_{ui} = 85 \text{ символов}$$

$$B_i = 1.5 \text{ символа}$$

$$T_{\text{ВВ}} = \frac{85}{\sqrt{1.5}} = 69.40 \text{ сек} = 0.02 \text{ часа}$$

Рассчитаем затраты на вывод информации:

$$m = 1$$

$$Q_{ui} = 120 \text{ символов}$$

$$B_i = 5000 \text{ символа}$$

$$T_{\text{ВЫВ}} = \frac{120}{\sqrt{5000}} = 1.70 \text{ сек} = 0.0005 \text{ часа}$$

Машинное время ЭВМ, необходимое для арифметической и логической обработки информации, можно определить по формуле:

$$T_a = n_a \times t_a + t_0 \times \sum_{i=1}^k n_i,$$

где n_a — количество арифметических операций;
 t_a — продолжительность арифметической операции;
 t_0 — среднее время обращения к запоминающему устройству;
 n_i — количество обращений к i -му внешнему запоминающему устройству;
 k — число внешних запоминающих устройств.

Расчет затрат на арифметические и логические операции

Рассчитаем время, затраченное на арифметические и логические операции:

$$n_a = 500 \text{ операций}$$

$$t_a = 0.005 \text{ сек} \approx 0.0000013 \text{ часа}$$

$$t_0 = 0.03 \text{ сек} \approx 0.000008 \text{ часа}$$

$$n_i = 4$$

$$k = 1$$

$$T_a = 500 \times 0.0000013 + 0.000008 \times 4 \approx 0.000682 \text{ часа}$$

$$T_{\text{ЭВМ}} = T_{\text{вв}} + T_{\text{выв}} + T_a = 0.019 + 0.0005 + 0.000682 \approx 0.02 \text{ часа}$$

Теперь, зная все затраты труда на всех этапах решения задачи, мы можем узнать трудоёмкость решаемой задачи:

$$\begin{aligned} T_{\Sigma} &= T_{\text{изучение мат.}} + T_{\text{алгоритм и реш.}} + T_{\text{блок-схемы}} + T_{\text{программирование}} + \\ &+ T_{\text{машинно-ручные раб.}} + T_{\text{отладки}} + T_{\text{расчёта на эвм}} + T_{\text{документации}} = \\ &= 3.21 + 14.16 + 21.70 + 56.96 + 17.11 + 5.05 + 0.02 + 12.19 = \\ &= 130.40 \text{ часа} \end{aligned}$$

Трудоёмкость решения данной задачи на ЭВМ составляет 130.40 часа.

4.3 Расчёт себестоимости

При решении задачи на ЭВМ себестоимость разработки программ определяется по формуле:

$$S_э = S_{и} + S_{алг} + S_{бс} + S_{мр} + S_{отл} + S_{эвм} + S_{д},$$

где $S_э$ — себестоимость разработанных программ в рублях;

$S_{и}$ — себестоимость на этапе изучения описания задачи;

$S_{алг}$ — себестоимость разработки алгоритмов решения задачи;

$S_{бс}$ — себестоимость разработки схем алгоритмов программ;

$S_{пр}$ — себестоимость на этапе программирования;

$S_{мр}$ — себестоимость на этапе машинно-ручных операций;

$S_{отл}$ — себестоимость на этапе отладки;

$S_{эвм}$ — затраты, связанные с вычислением расчётов непосредственно на ЭВМ;

$S_{д}$ — затраты на оформление документации в удобном для чтения виде.

$$S_{и-д} = L_{ср.ч.} \times T_{и-д} + З_{доп.} + O_{соц.} + H_{цех.} + H_{зав.} + C_{эвм} \times T_{отл.эвм},$$

где $L_{ср.ч.}$ — среднечасовая заработная плата работника.

Заработная плата — часть национального дохода, поступающая в личное пользование трудящихся в соответствии с количеством и качеством затраченного труда.

Основная заработная плата определяется в зависимости от формы оплаты труда — сдельной или повременной, как сумма расценок по операциям.

Заработная плата включает:

- основную заработную плату;
- дополнительную заработную плату;
- отчисления в фонд социального страхования.

4.3.1 Расчёт себестоимости на этапе машинно-ручных работ

Расчет основной заработной платы

Основная заработная плата определяется по формуле:

$$З_{\text{осн}} = L_{\text{ср.ч.}} \times T_{\text{мр}}$$

Часовая тарифная ставка определяется по формуле:

$$L_{\text{ср.ч.}} = \frac{\text{Месячный штатный оклад}}{F_{\text{д}} \times 8}$$

$$F_{\text{д}} = 21.8 - \text{Месячный фонд времени}$$

$$L_{\text{ср.ч.}} = \frac{8000}{21.8 \times 8} = 45.87 \text{ руб./час}$$

$$З_{\text{осн}} = 45.87 \times 17.11 = 784.83 \text{ руб.}$$

Расчет дополнительной заработной платы

Дополнительная заработная плата определяется в процентном отношении от основной заработной платы и составляет 80%. В дополнительную заработную плату включается оплата за отпуск, выполнение государственных обязанностей и так далее.

$$З_{\text{доп}} = З_{\text{осн}} \times 80\%$$

$$З_{\text{доп}} = \frac{784.83 \times 80}{100} = 627.86 \text{ руб.}$$

Расчет отчислений на социальные нужды

Отчисления на социальное страхование определяется в процентном отношении от суммы основной и дополнительной заработной платы и составляет 24%. Сюда входят оплата больничных листов, оплата путевок, выплата пенсий.

$$O_{\text{соц}} = (З_{\text{осн}} + З_{\text{доп}}) \times 24\%$$
$$O_{\text{соц}} = \frac{(784.83 + 627.86) \times 24}{100} = 339.04 \text{ руб.}$$

Расчет цеховых накладных расходов

Цеховые накладные расходы определяются в процентном отношении от основной зарплаты и составляют 150–400%. В состав цеховых накладных расходов включаются такие затраты как заработная плата аппарата управления цехом (начальника цеха, заместителя начальника, нормировщика), амортизационные отчисления на текущий ремонт заданий, сооружений, на охрану труда в данном цехе и на непроизводительные затраты.

$$H_{\text{цех}} = З_{\text{осн}} \times 200\%$$
$$H_{\text{цех}} = \frac{784.83 \times 200}{100} = 1569.66 \text{ руб.}$$

Расчет заводских накладных расходов

Накладные расходы определяются в процентном отношении от основной зарплаты и составляют 80–150%. Заводские накладные расходы — это расходы по управлению заводом или фабрикой, содержание общезаводского персонала с отчислением на социальное страхование, расходы по командировкам, амортизационные отчисления на текущий ремонт зданий общезаводского назначения, отчисление на содержание вышестоящих организаций, а также потери от порчи и недостачи сырья и материалов на заводских складах.

$$H_{\text{зав}} = З_{\text{осн}} \times 100\%$$
$$H_{\text{зав}} = \frac{784.83 \times 100}{100} = 784.83 \text{ руб.}$$

Расчет работы дисплея

Вычисляется по формуле:

$$C_{\text{дисп}} = \frac{\Pi_{\text{дисп}} \times \alpha + \mathcal{E} \times T_{\text{п}}}{T_{\text{п}} \times F_{\text{д}} \times K_{\text{н}}},$$

где $\Pi_{\text{дисп}}$ — балансовая стоимость дисплея;

α — коэффициент, учитывающий затраты на профилактику и ремонт оборудования;

\mathcal{E} — затраты на электроэнергию;

$T_{\text{п}}$ — срок службы оборудования;

$F_{\text{д}}$ — годовой действительный фонд времени работы оборудования;

$K_{\text{н}}$ — коэффициент использования рабочего времени.

Затраты на электроэнергию

Вычисляются по формуле:

$$\mathcal{E} = N_{\text{э}} \times \eta \times C_{\text{э}} \times F_{\text{д}},$$

где $N_{\text{э}}$ — паспортная потребляемая мощность;

η — коэффициент использования оборудования по мощности;

$F_{\text{д}}$ — действительный годовой фонд времени работы оборудования;

$C_{\text{э}}$ — промышленный тариф электроэнергии.

Действительный годовой фонд времени работы дисплея можно определить по формуле:

$$F_{\text{д}} = D \times p \times t_{\text{см}} \times (1 - 0.01 \times \psi),$$

где D — количество рабочих дней в планируемом периоде;

p — число смен работы оборудования;

ψ — планируемый процент потерь времени на профилактику и ремонт оборудования;

$t_{\text{см}}$ — средняя продолжительность смены.

По известной нам формуле определим действительный годовой фонд времени:

$$F_{\text{д}} = 256 \times 1 \times 8 \times (1 - 0.01 \times 6) = 1925.12 \text{ ч.}$$

Теперь рассчитаем затраты на электроэнергию:

$$\mathcal{E} = 0.1 \times 0.7 \times 3.07 \times 1925.12 = 413.70 \text{ руб.}$$

Зная все величины, мы можем по формуле определить себестоимость машино-часа работы дисплея:

$\mathcal{C}_д = 6700 \text{ руб.}$	$\alpha = 1.2$	$T_{п} = 10 \text{ лет}$	$K_{н} = 0.8$
$N_{э} = 0.07 \text{ кВт}$	$\eta = 0.8$	$C_{э} = 1.53$	
$D = 256$	$h = 1 \text{ смена}$	$t_{см} = 8$	$\psi = 6\%$

$$C_{\text{дисп}} = \frac{6700 \times 1.2 + 413.70 \times 10}{10 \times 1925.12 \times 0.8} = 0.79 \text{ руб.}$$

Расчёт себестоимости

Зная все величины, рассчитаем себестоимость по следующей формуле:

$$\begin{aligned} S_{\text{мр}} &= \mathcal{Z}_{\text{осн}} + \mathcal{Z}_{\text{доп}} + O_{\text{соц}} + H_{\text{цех}} + H_{\text{зав}} + T_{\text{мр}} \times C_{\text{дисп}} \\ S_{\text{мр}} &= 784.83 + 627.86 + 339.04 + 1569.66 + 784.83 + 17.11 \times 0.79 = \\ &= 4119.73 \text{ руб.} \end{aligned}$$

4.3.2 Расчёт себестоимости на этапах от C_u до C_δ

Расчёт основной заработной платы

Основная заработная плата определяется по формуле:

$$\mathcal{Z}_{\text{осн}} = L_{\text{ср.ч.}} \times (T_{\text{и}} + T_{\text{алг}} + T_{\text{бс}} + T_{\text{пр}} + T_{\text{отл}} + T_{\text{д}}),$$

где $T_{\text{и}}$ — затраты труда на изучение материала, описание задачи;

$T_{\text{а}}$ — затраты труда на разработку алгоритмов решения задачи;

$T_{\text{бс}}$ — затраты труда на разработку схем алгоритмов программ;

T_{Π} — затраты труда на программирование;

$T_{\text{отл}}$ — затраты труда на отладку программ;

$T_{\text{д}}$ — затраты труда на оформление документации.

$$З_{\text{осн}} = 45.87 \times (3.21 + 14.16 + 21.70 + 56.96 + 5.05 + 12.19) = 5195.69 \text{ руб.}$$

Расчет дополнительной заработной платы

Зная основную, вычислим дополнительную заработную плату:

$$\begin{aligned} З_{\text{доп}} &= З_{\text{осн}} \times 80\% \\ З_{\text{доп}} &= \frac{5195.69 \times 80}{100} = 4156.55 \text{ руб.} \end{aligned}$$

Расчет отчислений на социальные нужды

Зная основную и дополнительную заработные платы, вычислим размер отчислений на социальные нужды:

$$\begin{aligned} O_{\text{соц}} &= (З_{\text{осн}} + З_{\text{доп}}) \times 24\% \\ O_{\text{соц}} &= \frac{(5195.69 + 4156.55) \times 24}{100} = 2244.53 \text{ руб.} \end{aligned}$$

Расчет цеховых накладных расходов

Зная размер основной заработной платы, вычислим накладные цеховые расходы:

$$\begin{aligned} H_{\text{цех}} &= З_{\text{осн}} \times 200\% \\ H_{\text{цех}} &= \frac{5195.69 \times 200}{100} = 10391.38 \text{ руб.} \end{aligned}$$

Расчет заводских накладных расходов

Зная размер основной заработной платы, вычислим накладные заводские расходы:

$$H_{\text{зав}} = Z_{\text{осн}} \times 100\%$$
$$H_{\text{зав}} = \frac{5195.69 \times 100}{100} = 5195.69 \text{ руб.}$$

Расчет стоимости часа работы ЭВМ

Себестоимость машино-часа работы определяется по формуле:

$$C_{\text{ЭВМ}} = \frac{\Phi_3 + \mathcal{E} + A + P + M + \mathcal{Ж} + A_3 \times (1 + K_p)}{F_d},$$

где Φ_3 — годовой фонд основной и дополнительной зарплаты персонала, обслуживающего ЭВМ;

\mathcal{E} — затраты на электроэнергию;

A — годовые амортизационные отчисления;

P — затраты на ремонт основного и вспомогательного оборудования;

M — затраты на материалы, связанные с эксплуатацией ЭВМ;

$\mathcal{Ж}$ — затраты, связанные с эксплуатацией вспомогательного оборудования и инвентаря;

A_3 — годовые амортизационные отчисления за используемую производственную площадь, ВЦ;

K_p — коэффициент, учитывающий расходы по содержанию, освещению, отоплению производственных помещений, ВЦ;

F_d — действительный годовой фонд времени работы ЭВМ.

По данным предприятия, стоимость одного часа работы на ЭВМ равна 40 рублям.

Расчёт себестоимости

Зная все величины, рассчитаем себестоимость по известной формуле:

$$\begin{aligned} S_{\text{и-д}} &= Z_{\text{осн.}} + Z_{\text{доп.}} + O_{\text{соц.}} + H_{\text{цех.}} + H_{\text{зав.}} + C_{\text{ЭВМ}} \times T_{\text{отл.ЭВМ}} \\ S_{\text{и-д}} &= 5195.69 + 4156.55 + 2244.53 + \\ &+ 10391.38 + 5195.69 + 40 \times 0.5 = 27203.84 \text{ руб.} \end{aligned}$$

4.3.3 Расчёт полной себестоимости

Вычислим внепроизводственные расходы:

$$B_{\text{н}} = \frac{(S_{\text{мр}} + S_{\text{и-д}}) \times 5}{100} = \frac{(4119.73 + 27203.84) \times 5}{100} = 1566.17 \text{ руб.}$$

Зная внепроизводственные расходы, вычислим общую себестоимость:

$$\begin{aligned} S_{\text{э}} &= S_{\text{мр}} + S_{\text{и-д}} + B_{\text{н}} = \\ &= 4119.73 + 27203.84 + 1566.17 = 32889.74 \text{ руб.} \end{aligned}$$

Себестоимость решения данной задачи на ЭВМ составляет 32889.74 рублей.

4.3.4 Анализ структуры себестоимости

Калькуляция — это исчисление себестоимости продукции по основным затратам, которые входят в состав себестоимости изделия.

Себестоимость продукции представляется не только важнейшей экономической категорией, но и качественным показателем, так как она характеризует уровень использования всех производственных ресурсов, находящихся в распоряжении предприятия.

Данная программа является трудоемкой, так как затраты связанные с составлением и реализацией программы выше, чем стоимость материалов.

Возможные пути снижения себестоимости для данной задачи:

- применение современных методов разработки документаций;
- использование более экономных вычислительных систем энергосберегающего и офисного класса;
- более рациональное распределение времени на профилактику и диагностику оборудования;
- использование автоматических средств для мониторинга состояния ЭВМ и диагностики неисправностей;
- применение более совершенных средств разработки;
- реорганизация некоторых аспектов порядка и правил пользования ЭВМ с целью сокращения времени простоя оборудования.

4.4 Графическая часть

В таблице 4.1 приведена производительность труда программиста.

В таблице 4.2 приведены значения трудоёмкости на различных этапах.

В таблице 4.3 приведены технико-экономические показатели.

В таблице 4.4 приведены значения калькуляции себестоимости.

Таблица 4.1 – Таблица производительности труда программиста

Характер работы	Производительность количество команд в час
Изучение описания задачи	75-85
Разработка алгоритмов решения	20-25
Разработка схемы алгоритма	10-15
Программирование по схеме алгоритма с использованием алгоритмического языка	20-25
Автономная отладка программы	4-5
Оформление документации	15-20

Таблица 4.2 – Таблица трудоёмкости

Наименование затраты	Единица измерения	Трудоёмкость работы
Изучение описания задачи	час	3,21
Разработка алгоритма программы	час	14,16
Разработка схемы алгоритма	час	21,70
Программирование	час	56,96
Машинно-ручные работы	час	17,11
Отладка программы	час	5,05
Расчёт на ЭВМ	час	0,02
Оформление документации	час	12,19
Общая трудоёмкость решения задачи на ЭВМ	час	130,40

Таблица 4.3 – Таблица технико-экономических показателей

№	Наименование показателя	Единица измерения	Формула расчёта	Результат
1	Трудоёмкость решения задачи на ЭВМ	час	$T_{\Sigma} = T_u + T_a + T_{бс} + T_{п} + T_{мр} + T_{отл} + T_{эвм} + T_{д}$	130,40
2	Себестоимость решения задачи на ЭВМ	руб.	$C_{\Sigma} = C_u + C_{алг} + C_{бл.сх.} + C_{пр} + C_{мр} + C_{отл} + C_{эвм} + C_{д}$	32889,74

Таблица 4.4 – Таблица калькуляции

№	Наименование затрат	Сумма, руб.	% к итогу	% к $З_{осн}$
	<i>Прямые расходы</i>			
1	Основная заработная плата	5980,52	19,09	100,00
2	Дополнительная заработная плата	4784,41	15,27	80,00
3	Отчисления на соцстрах	2583,57	8,25	43,20
4	Стоимость работы ЭВМ	33,51	0,11	0,56
	<i>Косвенные расходы</i>			
5	Цеховые накладные расходы	11961,04	38,19	200,00
6	Заводские накладные расходы	5980,52	19,09	100,00
	Производственная себестоимость	31323,57	100,00	–
	Цеховая себестоимость	25343,05	–	–
	Внепроизводственные расходы	1566,17	5,00	–
	Полная себестоимость	32889,74	105,00	–

Пояснение к таблице калькуляции

$$З_{\text{осн.}} = З_{\text{осн.}(мр)} + З_{\text{осн.}(и-д)} = 784.83 + 5195.69 = 5980.52 \text{ руб.}$$

$$З_{\text{доп.}} = З_{\text{доп.}(мр)} + З_{\text{доп.}(и-д)} = 627.86 + 4156.55 = 4784.41 \text{ руб.}$$

$$O_{\text{соц.}} = O_{\text{соц.}(мр)} + O_{\text{соц.}(и-д)} = 339.04 + 2244.53 = 2583.57 \text{ руб.}$$

$$H_{\text{цех.}} = H_{\text{цех.}(мр)} + H_{\text{цех.}(и-д)} = 1569.66 + 10391.38 = 11961.04 \text{ руб.}$$

$$H_{\text{зав.}} = H_{\text{зав.}(мр)} + H_{\text{зав.}(и-д)} = 784.83 + 5195.69 = 5980.52 \text{ руб.}$$

$$C_{\text{эвм.}} = T_{\text{отл}} \times C_{\text{эвм}} + T_{\text{мр}} \times C_{\text{дисп}} = 40 \times 0.5 + 17.11 \times 0.79 = 33.51 \text{ руб.}$$

$$\begin{aligned} C_{\text{цех.}} &= З_{\text{осн.}} + З_{\text{доп.}} + O_{\text{соц.}} + C_{\text{эвм.}} + H_{\text{цех.}} = \\ &= 5980.52 + 4784.41 + 2583.57 + 33.51 + 11961.04 = 25343.05 \text{ руб.} \end{aligned}$$

$$C_{\text{произ.}} = C_{\text{цех.}} + H_{\text{зав.}} = 25343.05 + 5980.52 = 31323.57 \text{ руб.}$$

$$B_{\text{н}} = (C_{\text{произ.}} \times 5)/100 = (31323.57 \times 5)/100 = 1566.17 \text{ руб.}$$

$$C_{\text{полн.}} = C_{\text{произ.}} + B_{\text{н.}} = 31323.57 + 1566.17 = 32889.74 \text{ руб.}$$

4.5 Эффективность от внедрения

Выразить или подсчитать экономическую эффективность от внедрения практически не представляется возможным, так как данное программное обеспечение не является заменой какому-либо существующему решению данной задачи. Можно лишь отметить, что данное программное обеспечение будет приносить существенную пользу в виде обеспечения возможности бесперебойной работы всех участков сети на конкретном предприятии.

5 ЗАКЛЮЧЕНИЕ

В данной дипломной работе была проведена разработка специализированного программного обеспечения для поддержания функционирования сетевой инфраструктуры, путём автоматического управления электропитанием активного сетевого оборудования. Применение данного программного обеспечения значительно упрощает поддержание работоспособности локальной сети на предприятии.

Программа написана на языке C и выполняется на любом компьютере на базе процессора типа x86, amd64, sparc, ppc, ppc64, alpha, hppa, mips, ia64 или arm под управлением любой POSIX-совместимой операционной системой (также доступна поддержка ОС Windows).

Детальные сообщения об ошибках позволяют диагностировать неисправности без обращения к руководству оператора. Универсальный алгоритм позволяет работать с любыми аналогичными устройствами, подключёнными к любому USB-порту и использующими аналогичный формат обмена. Возможность одновременной работы с несколькими устройствами удешевляет практическое внедрение системы.

С экономической точки зрения разработка данной программы является трудоёмкой. Себестоимость данной задачи составила 32889,74 рублей.

Применение данной системы при анализе и мониторинге состояния различных частей сети позволяет своевременно обнаруживать и устранять проблемы, связанные с невозможностью доступа в те или иные сегменты, получать статистическую информацию об отказе оборудования, предотвращать простои рабочего процесса, а также централизованно управлять большим комплексом систем.

Список используемой литературы

- 1 *Awodey S.* Category Theory. — Oxford University Press, 2006. — 272 с.
- 2 *Абельсон Х., Сассман Д.* Структура и интерпретация компьютерных программ. — Добросвет, 2006. — 608 с.
- 3 *Гуссенс М., Миттельбах Ф., Самарин А.* Путеводитель по пакету \LaTeX и его расширению $\text{\LaTeX 2}_{\epsilon}$: Пер. с англ. — М.: Мир, 1999. — 606 с.
- 4 *Керниган Б., Ритчи Д.* Язык программирования Си. — М.: Вильямс, 2007. — 304 с.
- 5 *Столлман Р.* Руководство по GNU Emacs. — АНО “ИЛКиРЛ”, 1999. — 612 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программы

```
/*
 * Дипломная работа на тему "Разработка программного обеспечения для
 * автоматического управления электропитанием активного сетевого оборудования"
 *
 * Разработал студент группы МП41–06, Муковников М. Ю.
 * Версия 1.0.0
 *
 */

/*
 * keusb.c — интерфейс к устройству Ke-USB24R
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "build.h"
#include "device.h"

#define RESET_SLEEP 5

int
die(const char *str)
{
    fprintf(stderr, "error: %s\n", str);
    exit(1);
    return 0;
}

int
print_usage()
{
    puts("KeUSB* module interface.\n"
        "Usage: keusb [<device>] <command> <#>\n\n"
        "device:\n"
        "  " USAGE_COM "                Use specified device\n"
        "  xxxx-xxxx-xxxx-xxxx         Use signature to find device\n\n"
        "command:\n"
        "  —help                        Display this help\n"
        "  —vesrion                     Display version information\n\n"
        "  turn_on <#>                 Turn relay # on\n"
        "  turn_off <#>                Turn relay # off\n"
        "  toggle <#>                  Toggle relay #\n"
        "  reset <#> [<sec>]           Reset relay # on 5 or <sec> seconds\n"
        "  status <#>                  Display status of relay #\n"
        "  status all                   Display status of all relays\n"
        "  status                       Display device information\n"
        "  reset -l                     Hard reset device\n\n"
        "Copyright Mikhail Mukovnikov, 2010\n")
}
```

```
    "Email bugs to <mix_mix@pop3.ru>");

    return 2;
}

int
main(int argc, char *argv[])
{
    int stat;

    if (!shift())
        return print_usage();

    if (eq("--help"))
        return print_usage();

    if (eq("--version"))
        return !puts("keusb " VERSION_STR);

    if (!(eq("turn_on")
        || eq("turn_off")
        || eq("toggle")
        || eq("reset")
        || eq("status"))))
    {
        if (strlen(argv[0]) == 19
            && strchr(argv[0], '-') )
            stat = keusb_connect(CONNECT_SIG, argv[0]);
        else
            stat = keusb_connect(CONNECT_FILE, argv[0]);

        if (!shift())
            return print_usage();
    }
    else
        stat = keusb_connect(CONNECT_ANY, 0);

    if (!stat)
        return !die("keusb_main: can't open device");

    if (eq("turn_on"))
    {
        shift_or_die();
        kexec(keusb_turn_on_off(atoi(argv[0]), 1));
        return 0;
    }

    if (eq("turn_off"))
    {
        shift_or_die();
        kexec(keusb_turn_on_off(atoi(argv[0]), 0));
        return 0;
    }
}
```

```
if (eq("toggle"))
{
    shift_or_die();
    kexec(keusb_toggle(atoi(argv[0])));
    return 0;
}

if (eq("reset"))
{
    int op1, op2;

    shift_or_die();
    op1 = atoi(argv[0]);
    op2 = shift() ? atoi(argv[0]) : RESET_SLEEP;

    if (op1 == -1)
    { kexec(keusb_hard_reset()); }
    else
    { kexec(keusb_reset(op1, op2)); }

    return 0;
}

if (eq("status"))
{
    if (shift())
    {
        if(eq("all"))
        {
            puts(keusb_status_all());
            return 0;
        }

        stat = keusb_status(atoi(argv[0]));

        if (stat == 2)
            return die("keusb_main: status failed") - 1;

        printf("%d\n", stat);
        return stat;
    }
    else
        puts(keusb_selftest());

    return 0;
}

return 1;
}
```

```
/*  
 * keusbd.c — анализатор доступности узлов сети  
 *  
 * Разработал: Муковников М. Ю.  
 * Версия: 1.0.0  
 * Изменён: 16.04.2010  
 *  
 */
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <signal.h>  
#include <unistd.h>  
#include <time.h>
```

```
#include "build.h"  
#include "config.h"  
#include "daemon.h"
```

```
FILE* log_fd;
```

```
int  
logger(char level, const char *str)  
{  
    char buffer[80];  
    time_t rawtime;  
    struct tm *timeinfo;  
    const char *p = str;  
  
    while (*p++);  
    if (str == —p)  
        return 0;  
  
    time(&rawtime);  
    timeinfo = localtime(&rawtime);  
    strftime(buffer, 80, "%b %d %X", timeinfo);  
  
    fprintf(log_fd, "(%c%c) %s > %s",  
            level, level, buffer, str);  
  
    if (*--p != '\n')  
        fputc('\n', log_fd);  
  
    fflush(log_fd);  
    return 0;  
}
```

```
void  
finalize(int code)  
{  
    logger('I', "keusbd stopped");  
}
```

```
fclose(log_fd);  
exit(code);  
}
```

```
int  
die(const char *str)  
{  
    logger('E', str);  
    finalize(1);  
    return 0;  
}
```

```
int  
print_usage()  
{  
    puts("KeUSB daemon.\n"  
        "Usage: keusbd [<args>]\n\n"  
        "general:\n"  
        "  --help                Display this help\n"  
        "  --version            Display version information\n"  
        "  --daemon            Run daemon\n\n"  
        "files:\n"  
        "  -f <cfg>             Config file\n"  
        "  -l <log>           Log file\n"  
        "  -p <pid>           Pid file\n\n"  
        "command:\n"  
        "  -c                 Validate config and exit\n"  
        "  -d                 Same as --daemon\n"  
        "  -k                 Kill daemon\n\n"  
        OS_SPECIFIC_OPTIONS  
        "\nCopyright Mikhail Mukovnikov, 2010\n"  
        "Email bugs to <mix_mix@pop3.ru>");  
  
    return 2;  
}
```

```
int  
main(int argc, char *argv[])  
{  
    FILE *pid_fd;  
    char cfg_name[] = CONFIG_NAME;  
    char log_name[] = LOG_NAME;  
    char pid_name[] = PID_NAME;  
    char *file_n[3] = {cfg_name, log_name, pid_name};  
  
    signal(SIGINT, finalize);  
    signal(SIGTERM, finalize);  
  
    if (argc < 2)  
        return print_usage();  
  
    log_fd = stdout;
```

```
while (shift())
{
    if (eq("--help"))
        return print_usage();

    if (eq("--version"))
        return !puts("keusbd " VERSION_STR);

    if (eq("--daemon") || eq("-d"))
        continue;

    if (eq("-f"))
    {
        shift_or_die();
        file_n[0] = argv[0];
        continue;
    }

    if (eq("-l"))
    {
        shift_or_die();
        file_n[1] = argv[0];
        continue;
    }

    if (eq("-p"))
    {
        shift_or_die();
        file_n[2] = argv[0];
        continue;
    }

    if (eq("-c"))
    {
        config_parse(file_n[0]);
        rules_test();
        logger('I', "config was successfully parsed");
        return 0;
    }

    if (eq("-k"))
    {
        int size;
        pid_t pid;

        pid_fd = fopen(file_n[2], "r");
        die_p(pid_fd, "daemon_kill: it seems daemon isn't running");
        size = fscanf(pid_fd, "%u", &pid);
        fclose(pid_fd);

        die_p(!kill(pid, SIGTERM), "daemon_kill: can't kill process");
        return 0;
    }

    die_p(cmdline_parse_rest(argc, argv),
```



```
        "cmdline_parse: invalid argument");
    }

    daemonize();

    log_fd = fopen(file_n[1], "a");
    die_p(log_fd, "logger_init: can't open file");
    fputc('\n', log_fd);
    logger('I', "keusbd started");

    pid_fd = fopen(file_n[2], "w+");
    die_p(pid_fd, "daemon_pid: can't open pid file");
    fprintf(pid_fd, "%u\n", getpid());
    fclose(pid_fd);

    config_parse(file_n[0]);
    rules_test();

    while (1)
    {
        rules_exec();
    }

    finalize(0);
    return 0;
}
```

```
/*
 * config.h — заголовочный файл модуля config.c
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */
```

```
#ifndef CONFIG_H
#define CONFIG_H

char *config_parse_line(char **str);
void config_parse(char *name);

void rules_test();
void rules_exec();
void rules_free();

#endif // CONFIG_H
```

```
/*
 * config.c — модуль обработки конфигурационного файла
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */
```

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```
#include "build.h"
#include "daemon.h"
```

```
typedef struct ring_t
{
    char *str;
    struct ring_t *expr;
    struct ring_t *next;
    struct ring_t *prev;
} ring_t;
```

```
static int conf_wait_time = 5*60;
static int conf_tries_num = 10;
static char *config_file;
static ring_t rules = {0, 0, &rules, &rules};
```

```
static void *
xalloc(size_t size)
{
    void *ptr = malloc(size);
    die_p(ptr, "xalloc: can't allocate memory");
    return ptr;
}
```

```
void
rules_add(ring_t *rule, char *str)
{
    die_p(rule, "rules_add: missing action");

    while (1)
    {
        ring_t *t = (ring_t *) xalloc(sizeof(ring_t));

        t->str = str;
        t->expr = 0;
```

```
t->next = rule;
t->prev = rule->prev;
rule->prev->next = t;
rule->prev = t;

while (*str && *str != ' ') str++;
if (!*str) break;
*str++ = '\0';
}
}

int
rules_test_expr(ring_t **expr)
{
    char *str;

    *expr = (*expr)->next;
    str = (*expr)->str;

    die_p(str, "rules_test_expr: missing argument");

    if (eq2(str, "and")
        || eq2(str, "or")
        || eq2(str, "xor"))
        return rules_test_expr(expr)
            + rules_test_expr(expr) + 1;

    if (eq2(str, "not"))
        return rules_test_expr(expr) + 1;

    return 1;
}

int
rules_eval_expr(ring_t **expr)
{
    char *str;

    *expr = (*expr)->next;
    str = (*expr)->str;

    if (eq2(str, "and"))
    {
        int a, b;
        a = rules_eval_expr(expr);
        b = a ? rules_eval_expr(expr) : rules_test_expr(expr);
        return a && b;
    }

    if (eq2(str, "or"))
    {
        int a, b;
        a = rules_eval_expr(expr);
```

```
    b = a ? rules_test_expr(expr) : rules_eval_expr(expr);
    return a || b;
}

if (eq2(str, "not"))
    return !rules_eval_expr(expr);

if (eq2(str, "xor"))
    return rules_eval_expr(expr)
        ^ rules_eval_expr(expr);

return ping(str, conf_tries_num);
}

void
rules_test()
{
    ring_t *k;

    for (k = rules.next; k != &rules; k = k->next)
    {
        ring_t *t = k->expr;
        rules_test_expr(&t);
        die_p(t->next == k->expr, "rules_exec: unlinked expression");
    }
}

void
rules_exec()
{
    ring_t *k;

    for (k = rules.next; k != &rules; k = k->next)
    {
        ring_t *t = k->expr;
        int result = rules_eval_expr(&t);

        if (result)
        {
            FILE *pipe;
            char buf[85];

            logger('=', k->str);
            strncpy(buf, k->str, 80);
            strcat(buf, " 2>&1");
            pipe = popen(buf, "r");

            if (!pipe)
            {
                logger('W', "rules_exec: can't open pipe");
                continue;
            }
        }
    }
}
```

```
        while (fgets(buf, 85, pipe))
            logger('-', buf);

        pclose(pipe);
    }
}

sleep(conf_wait_time);
}

void
rules_free()
{
    ring_t *k, *r, *t;

    k = rules.next;
    while (k != &rules)
    {
        r = k->expr->next;
        while (r != k->expr)
        {
            t = r;
            r = r->next;
            free(t);
        }

        t = k;
        k = k->next;
        free(t);
    }

    free(config_file);
}

void
config_init(const char *name)
{
    int size;
    FILE *fp = fopen(name, "r");

    die_p(fp, "config_init: can't open config");

    fseek(fp, 0L, SEEK_END);
    size = ftell(fp);
    rewind(fp);

    config_file = xalloc(size + 2);
    memset(config_file, 0, size + 2);

    size = fread(config_file, size, sizeof(char), fp);
    fclose(fp);
    atexit(rules_free);
}
```

```
char *  
config_parse_line(char **str)  
{  
    char *p, *q, *k;  
  
    while (1)  
    {  
        int fl = 0;  
  
        p = *str;  
        while (*p && isspace(*p)) p++;  
        if (!*p) return 0;  
  
        q = p;  
        while (*q && *q != '\n' && *q != '#') q++;  
        if (p == q) q--;  
        while (q > p && isspace(*--q));  
  
        k = q;  
        while (*++k && *k != '\n');  
        if (*k) k++;  
  
        *++q = '\0';  
        *str = q = p--;  
        while (*++p)  
        {  
            int t = isspace(*p);  
  
            if (t && fl) continue;  
            fl = t ? 1 : 0;  
            *q++ = t ? ' ' : *p;  
        }  
  
        *q = '\0';  
        if (q - *str) return k;  
  
        *str = k;  
    }  
  
    return 0;  
}
```

```
void  
config_parse(const char *name)  
{  
    char *p, *q;  
  
    config_init(name);  
  
    p = config_file;  
    while ((q = p, p = config_parse_line(&q)))  
    {
```

```
if (q[0] == '!')
{
    char *c = (*++q == ' ') ? ++q : q;

    while (*q && *q != ' ') q++;
    die_p(*q, "config_parse: argument not found");
    *q++ = '\\0';

    if (eq2(c, "tries"))
    {
        conf_tries_num = atoi(q);
        die_p(conf_tries_num > 0 && conf_tries_num < 100,
            "config_parse: invalid argument");
        continue;
    }

    if (eq2(c, "wait"))
    {
        char *postfix;

        conf_wait_time = strtol(q, &postfix, 10);
        die_p(conf_wait_time > 0, "config_parse: invalid argument");

        switch (*postfix)
        {
            case 's': case '\\0': break;
            case 'm': conf_wait_time *= 60; break;
            case 'h': conf_wait_time *= 3600; break;
            default: die("config_parse: invalid postfix"); break;
        }

        continue;
    }

    die("config_parse: invalid option");
}

if (q[0] == '%')
{
    ring_t *t = (ring_t *) xalloc(sizeof(ring_t));

    t->str = (*++q == ' ') ? q + 1 : q;
    t->expr = (ring_t *) xalloc(sizeof(ring_t));
    t->expr->str = 0;
    t->expr->expr = 0;
    t->expr->next = t->expr;
    t->expr->prev = t->expr;
    t->next = &rules;
    t->prev = rules.prev;
    rules.prev->next = t;
    rules.prev = t;
    continue;
}

rules_add(rules.prev->expr, q);
```

```
}  
}
```

```
/*  
 * device.h — заголовочный файл для модуля device.c  
 *  
 * Разработал: Муковников М. Ю.  
 * Версия:      1.0.0  
 * Изменён:     16.04.2010  
 *  
 */
```

```
#ifndef DEVICE_H  
#define DEVICE_H
```

```
#define CONNECT_FILE 0  
#define CONNECT_SIG  1  
#define CONNECT_ANY  2
```

```
int keusb_request(const char *command, ...);  
int keusb_connect(int type, char *path);
```

```
int keusb_status(int r_num);  
char *keusb_status_all();
```

```
int keusb_turn_on_off(int r_num, int on_off);  
int keusb_toggle(int r_num);  
int keusb_reset(int r_num, int wait);  
int keusb_hard_reset();
```

```
char *keusb_get_signature();  
char *keusb_selftest();
```

```
#endif // DEVICE_H
```

```
/*  
 * device.c — модуль связи с устройством Ke-USB24R  
 *  
 * Разработал: Муковников М. Ю.  
 * Версия:      1.0.0  
 * Изменён:     16.04.2010  
 *  
 */
```

```
#include <stdio.h>
```



```
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
#include <ctype.h>

#include "build.h"
#include "device.h"
#include "driver.h"

#define REQUEST_SIZE 200
#define REPLY_PARTS 10

static char *airbag = "";
static char *device_name = "";
static int reply_size = 0;
static char reply_buf[REQUEST_SIZE];
static char *reply_part[REPLY_PARTS];

int
keusb_request(const char *command, ...)
{
    va_list ap;
    int length;
    int tries = 3;
    char request[REQUEST_SIZE + 3];

    va_start(ap, command);
    vsnprintf(request, REQUEST_SIZE, command, ap);
    va_end(ap);

    length = strlen(request);
    request[length] = '\r';
    request[length + 1] = '\n';
    request[length + 2] = '\0';

    while (tries--)
    {
        int parts = 0;
        char *p, *q;

        reply_buf[0] = '\0';
        device_write(request, length + 2);
        device_read(reply_buf, REQUEST_SIZE);

        if (!strncmp(reply_buf, "#ERR\r\n", 6)
            || reply_buf[0] != '#')
            continue;

        for (p = q = reply_buf + 1; *p != '\r'; p++)
            if (*p == ',')
            {
```

```
        reply_part[parts++] = q;
        *p = '\0';
        q = p + 1;
    }

    *p = '\0';
    reply_size = parts + 1;
    reply_part[parts] = q;
    break;
}

return tries + 1;
}

char *
str_to_lower(char *str)
{
    char *p = str;

    while (*p && *p != ' ')
        *p = tolower(*p), p++;

    *p = '\0';
    return str;
}

char *
keusb_get_signature()
{
    if (!keusb_request("$KE,SER") || reply_size < 2)
        return airbag;

    return str_to_lower(reply_part[1]);
}

int
keusb_connect(int type, char *path)
{
    char *name;
    device_name = path ? path : airbag;

    if (type == CONNECT_FILE)
        return (device_open(path)
                && keusb_request("$KE"))
            || die("keusb_connect: can't open device");

    while ((name = device_gen_name()))
    {
        if (!device_open(name))
            continue;

        if (!keusb_request("$KE"))
```

```
{
    device_close();
    continue;
}

if (type == CONNECT_SIG
    && strcmp(keusb_get_signature(), str_to_lower(path)))
{
    device_close();
    continue;
}

device_name = name;
atexit(device_close);
return 1;
}

return die("keusb_connect: can't find device");
}

int
keusb_status(int r_num)
{
    if (!keusb_request("$KE,RDR,%d", r_num) || reply_size < 3)
        die("keusb_status: request failed");

    return reply_part[2] ? atoi(reply_part[2]) : 2;
}

char *
keusb_status_all()
{
    int i;
    char buf[REQUEST_SIZE];

    if (!keusb_request("$KE,RDR,ALL"))
    {
        die("keusb_status_all: request failed");
        return airbag;
    }

    buf[0] = '\0';
    for (i = 2; i < reply_size; i++)
    {
        strcat(buf, reply_part[i]);
        strcat(buf, "\t");
    }

    return strcpy(reply_buf, buf);
}
```

```
int
keusb_turn_on_off(int r_num, int on_off)
{
    return keusb_request("$KE,REL,%d,%d", r_num, on_off)
        || die("keusb_turn_on_off: request failed");
}
```

```
int
keusb_toggle(int r_num)
{
    int status = keusb_status(r_num);

    if (status == 2)
        return die("keusb_toggle: request failed");
    else
        return keusb_turn_on_off(r_num, !status)
            || die("keusb_toggle: turn_on_off failed");
}
```

```
int
keusb_reset(int r_num, int wait)
{
    int status = keusb_status(r_num);

    if (status == 2)
        return die("keusb_reset: status failed");
    else
        return (keusb_turn_on_off(r_num, !status)
            && !sleep(wait)
            && keusb_turn_on_off(r_num, status))
            || die("keusb_reset: turn_on_off failed");
}
```

```
char *
keusb_selftest()
{
    char info[REQUEST_SIZE];

    if (!keusb_request("$KE,USB,GET") || reply_size < 2)
    {
        die("keusb_selftest: request failed");
        return airbag;
    }

    strncpy(info, device_name, 50);
    strcat(info, " | ");
    strncat(info, reply_part[1], 50);
    strcat(info, " | ");
    strncat(info, keusb_get_signature(), 80);
    strcat(info, " | status: OK");
    strcpy(reply_buf, info);
}
```

```
    return reply_buf;
}

int
keusb_hard_reset()
{
    return keusb_request("$KE,RST")
        || die("keusb_hard_reset: request failed");
}

/*
 * posix/daemon.h — заголовочный файл модуля daemon.c
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */

#ifndef DAEMON_H
#define DAEMON_H

void daemonize();
int ping(const char *host, int tries);
int cmdline_parse_rest(int argc, char *argv[]);

#endif // DAEMON_H

/*
 * posix/daemon.c — модуль связи с операционной системой
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>

#include "build.h"
```

```
void
daemonize ()
{
    pid_t pid = fork ();

    if (pid < 0) exit (1);
    if (pid > 0) exit (0);

    umask (0);
    setsid ();

    close (STDIN_FILENO);
    close (STDOUT_FILENO);
    close (STDERR_FILENO);
}

int
ping(const char *host, int tries)
{
    int status;
    char buf[80];

    snprintf(buf, 80, "ping -c %d %s > /dev/null 2>&1", tries, host);
    status = system(buf);

    return (status != -1) ? WEXITSTATUS(status) : 2;
}

int
cmdline_parse_rest(int argc, char *argv[])
{
    return 0;
}

/*
 * posix/driver.h — заголовочный файл модуля driver.c
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */

#ifndef DRIVER_H
#define DRIVER_H

char *device_gen_name();
```

```
int device_open(const char *name);
void device_close();

int device_write(const void *buf, size_t count);
int device_read(void *buf, size_t count);

#endif // DRIVER_H

/*
 * posix/driver.c — модуль подключения к устройству Ke-USB24R
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */

#include <unistd.h>
#include <termios.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#define READ_TIMEOUT 100

static char device_name[] = "/dev/ttyACM?";
static int device_fd = -1;

char *
device_gen_name()
{
    static int try_num = 0;
    static int mod_num = 0;

    if (try_num > 9)
    {
        if (!mod_num++)
        {
            try_num = 0;
            device_name[8] = 'U';
            device_name[9] = 'S';
            device_name[10] = 'B';
        }
        else
            return 0;
    }

    device_name[11] = '0' + try_num++;
    return device_name;
}
```

```
}
```

```
int
```

```
device_open(const char *name)
```

```
{
```

```
    struct termios tty;
```

```
    device_fd = open(name, O_RDWR | O_NDELAY | O_NOCTTY);
```

```
    if (device_fd == -1)
```

```
        return 0;
```

```
    ioctl(device_fd, TCGETA, &tty);
```

```
    tty.c_lflag      = 0;
```

```
    tty.c_iflag      = BRKINT;
```

```
    tty.c_oflag      = 0;
```

```
    tty.c_cflag      = B9600 | CS8 | CREAD | CLOCAL | PARENB;
```

```
    tty.c_cflag      &= ~PARODD;
```

```
    tty.c_cc[VMIN]   = 0;
```

```
    tty.c_cc[VTIME]  = 1;
```

```
    ioctl(device_fd, TCSETA, &tty);
```

```
    return 1;
```

```
}
```

```
int
```

```
device_write(const void *buf, size_t count)
```

```
{
```

```
    return write(device_fd, buf, count) + 1;
```

```
}
```

```
int
```

```
device_read(void *buf, size_t count)
```

```
{
```

```
    usleep(READ_TIMEOUT * 1000);
```

```
    return read(device_fd, buf, count) + 1;
```

```
}
```

```
void
```

```
device_close()
```

```
{
```

```
    close(device_fd);
```

```
}
```

```
/*
```

```
 * posix/build.h — макроопределения и параметры компиляции
```

```
 *
```

```
 * Разработал: Муковников М. Ю.
```

```
 * Версия:      1.0.0
```


** Изменён: 16.04.2010*

**/*

```
#ifndef BUILD_H
#define BUILD_H

#define shift() (++argv, --argc)
#define shift_or_die() if (!shift()) return die("cmdline_parse: missing operand")
#define eq(x) !strcmp(argv[0], (x))
#define eq2(x, y) !strcmp((x), (y))
#define die_p(x, y) if (!(x)) die((y))
#define kexec(x) if (!(x)) return !die("keusb_main: failed")

#define VERSION_STR "1.0.0"
#define USAGE_COM  "/dev/ttyXX"
#define CONFIG_NAME "/etc/conf.d/keusb"
#define LOG_NAME    "/var/log/keusb.log"
#define PID_NAME    "/var/run/keusb.pid"
#define OS_SPECIFIC_OPTIONS

int die(const char *str);
int logger(char level, const char *str);

#endif // BUILD_H

/*
 * windows/daemon.h — заголовочный файл модуля daemon.c
 *
 * Разработал: Муковников М. Ю.
 * Версия: 1.0.0
 * Изменён: 16.04.2010
 *
 */

#ifndef DAEMON_H
#define DAEMON_H

void daemonize();
int ping(const char *host, int tries);
int cmdline_parse_rest(int argc, char *argv[]);

#endif // DAEMON_H
```

```

/*
 * windows/daemon.c — модуль связи с операционной системой
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */

```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <windows.h>
#include <winsock.h>
```

```
#include "build.h"
```

```
#define TIME_TO_WAIT 5
#define PING_PACKET_SIZE 64
```

```
typedef struct
{
    unsigned char Ttl;
    unsigned char Tos;
    unsigned char Flags;
    unsigned char OptionsSize;
    unsigned char *OptionsData;
} IP_OPTION_INFORMATION, *PIP_OPTION_INFORMATION;
```

```
typedef struct
{
    DWORD Address;
    unsigned long Status;
    unsigned long RoundTripTime;
    unsigned short DataSize;
    unsigned short Reserved;
    void *Data;
    IP_OPTION_INFORMATION Options;
    char Answer[PING_PACKET_SIZE];
} IP ECHO REPLY, *PIP ECHO REPLY;
```

```
typedef HANDLE (WINAPI* pfnHV)(VOID);  
typedef BOOL (WINAPI* pfnBH)(HANDLE);  
typedef DWORD (WINAPI* pfnDHDPWPipPDD)(HANDLE, DWORD, LPVOID, WORD,  
PIP_OPTION_INFORMATION,  
LPVOID, DWORD, DWORD);
```

```
const char *hklm_run = "Software\\Microsoft\\Windows\\CurrentVersion\\Run";
```

```
HANDLE hIP;  
WSADATA wsaData;  
HINSTANCE hIcmp;  
pfnHV pIcmpCreateFile;  
pfnBH pIcmpCloseHandle;  
pfnDHDPWPipPDD pIcmpSendEcho;
```

```
void  
daemonize()  
{  
    char *p;  
    char buf[260];  
  
    CreateMutex(NULL, TRUE, "keusbd");  
    if (GetLastError() == ERROR_ALREADY_EXISTS)  
        die("daemonize: daemon is already running");  
  
    GetModuleFileName(0, buf, 260);  
  
    p = buf;  
    while(*p++);  
    while(*p != '\\') p--;  
    *p = '\\0';  
  
    chdir(buf);  
    FreeConsole();  
}
```

```
void  
icmp_free()  
{  
    pIcmpCloseHandle(hIP);  
    FreeLibrary(hIcmp);  
    WSACleanup();  
}
```

```
int  
icmp_init()  
{  
    static int icmp_done = 0;  
  
    if (icmp_done) return 0;  
  
    if (WSAStartup(MAKEWORD(1, 1), &wsaData))  
        return die("icmp_init: can't initialize winsock");  
  
    hIcmp = LoadLibrary("icmp.dll");  
    die_p(hIcmp, "icmp_init: unable to load icmp.dll");  
}
```

```
pIcmpCreateFile = (pfnHV) GetProcAddress(hIcmp, "IcmpCreateFile");
pIcmpCloseHandle = (pfnBH) GetProcAddress(hIcmp, "IcmpCloseHandle");
pIcmpSendEcho = (pfnDHDPWPipPDD) GetProcAddress(hIcmp, "IcmpSendEcho");

if (!pIcmpCreateFile || !pIcmpCloseHandle || !pIcmpSendEcho)
    return die("icmp_init: cannot find icmp functions");

hIP = pIcmpCreateFile();
if (hIP == INVALID_HANDLE_VALUE)
    return die("icmp_init: unable to open ping service");

icmp_done = 1;
atexit(icmp_free);
return 0;
}

int
icmp_ping(const char *host)
{
    IP_ECHO_REPLY Ipe;
    struct hostent* phe;
    char acPingBuffer[PING_PACKET_SIZE];
    DWORD dwStatus;

    phe = gethostbyname(host);
    if (!phe) return 1;

    memset(&Ipe, '\0', sizeof(IP_ECHO_REPLY));
    memset(acPingBuffer, '\xAA', sizeof(acPingBuffer));

    Ipe.Data = acPingBuffer;
    Ipe.DataSize = sizeof(acPingBuffer);

    dwStatus = pIcmpSendEcho(hIP, *((DWORD*)phe->h_addr_list[0]),
                             acPingBuffer, sizeof(acPingBuffer), NULL, &Ipe,
                             sizeof(IP_ECHO_REPLY), TIME_TO_WAIT * 1000);

    if (dwStatus && memcmp(Ipe.Answer, acPingBuffer, PING_PACKET_SIZE))
        dwStatus = 0;

    return !dwStatus;
}

int
ping(const char *host, int tries)
{
    int i, failed = 0;

    icmp_init();
    for (i = 0; i < tries; i++)
        failed += icmp_ping(host);

    return (failed == tries);
}
```

```
}

int
cmdline_parse_rest(int argc, char *argv[])
{
    HKEY hKey;

    if (eq("-ay"))
    {
        char buf[260];

        GetModuleFileName(0, buf, 260);
        strcat(buf, " —daemon");
        RegOpenKeyEx(HKEY_LOCAL_MACHINE, hklm_run, 0, KEY_SET_VALUE, &hKey);
        RegSetValueEx(hKey, "keusbd", 0, REG_SZ, (const BYTE*) buf, strlen(buf));
        RegCloseKey(hKey);
        exit(0);
    }

    if (eq("-an"))
    {
        RegOpenKeyEx(HKEY_LOCAL_MACHINE, hklm_run, 0, KEY_SET_VALUE, &hKey);
        RegDeleteValue(hKey, "keusbd");
        RegCloseKey(hKey);
        exit(0);
    }

    return 0;
}

/*
 * windows/driver.h — заголовочный файл модуля driver.c
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */

#ifndef DRIVER_H
#define DRIVER_H

char *device_gen_name();

int device_open(const char *name);
void device_close();

int device_write(const void *buf, size_t count);
int device_read(void *buf, size_t count);
```

```
#endif // DRIVER_H
```

```
/*  
 * windows/driver.c — модуль подключения к устройству Ke-USB24R  
 *  
 * Разработал: Муковников М. Ю.  
 * Версия:      1.0.0  
 * Изменён:     16.04.2010  
 *  
 */
```

```
#include <windows.h>  
#include <stdio.h>
```

```
#define READ_TIMEOUT 100
```

```
static char device_name[] = "\\\\.\\COM?";  
static HANDLE device_fd = 0;
```

```
char *  
device_gen_name()  
{  
    static int try_num = 1;  
  
    if (try_num > 9)  
        return 0;  
  
    device_name[7] = '0' + try_num++;  
    return device_name;  
}
```

```
int  
device_open(const char *name)  
{  
    DCB dcb;  
    COMMTIMEOUTS CommTimeOuts;  
  
    device_fd = CreateFile(name, GENERIC_READ | GENERIC_WRITE,  
                           0, NULL, OPEN_EXISTING, 0, NULL);  
  
    if (device_fd == INVALID_HANDLE_VALUE)  
        return 0;  
  
    GetCommState(device_fd, &dcb);  
  
    dcb.BaudRate = CBR_9600;  
    dcb.ByteSize = 8;
```

```
dcb.Parity    = NOPARITY;
dcb.StopBits  = ONESTOPBIT;

CommTimeOuts.ReadIntervalTimeout      = MAXDWORD;
CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
CommTimeOuts.ReadTotalTimeoutConstant  = 0;
CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
CommTimeOuts.WriteTotalTimeoutConstant  = 1000;

SetCommTimeouts(device_fd, &CommTimeOuts);
SetCommState(device_fd, &dcb);

    return 1;
}

int
device_write(const void *buf, size_t count)
{
    DWORD bytes_written;

    WriteFile(device_fd, buf, count, &bytes_written, NULL);
    return bytes_written;
}

int
device_read(void *buf, size_t count)
{
    DWORD bytes_read;

    Sleep(READ_TIMEOUT);
    ReadFile(device_fd, buf, count, &bytes_read, NULL);
    return bytes_read;
}

void
device_close()
{
    CloseHandle(device_fd);
}

/*
 * windows/build.h — макроопределения и параметры компиляции
 *
 * Разработал: Муковников М. Ю.
 * Версия:      1.0.0
 * Изменён:     16.04.2010
 *
 */
```

```
#ifndef BUILD_H
#define BUILD_H

#include <windows.h>

#define sleep(x) (Sleep((x)*1000),0)
#define kill(x, y) !TerminateProcess(OpenProcess(PROCESS_TERMINATE, 0, (x)), 0)

#define shift() (++argv, --argc)
#define shift_or_die() if (!shift()) return die("cmdline_parse: missing operand")
#define eq(x) !strcmp(argv[0], (x))
#define eq2(x, y) !strcmp((x), (y))
#define die_p(x, y) if (!(x)) die((y))
#define kexec(x) if (!(x)) return !die("keusb_main: failed")

#define VERSION_STR "1.0.0"
#define USAGE_COM "com1: .. com9:"
#define CONFIG_NAME "keusbd.conf"
#define LOG_NAME "keusbd.log"
#define PID_NAME "keusbd.pid"

#define OS_SPECIFIC_OPTIONS \
    " -ay Set autorun on\n" \
    " -an Set autorun off\n"

int die(const char *str);
int logger(char level, const char *str);

#endif // BUILD_H

#
# Makefile — сценарий автоматической сборки проекта
#
# Разработал: Муковников М. Ю.
# Версия: 1.0.0
# Изменён: 16.04.2010
#

CC = gcc
CFLAGS = -Wall -O2 -fomit-frame-pointer
BINS = /usr/bin
OBS1 = driver.o device.o
OBS2 = daemon.o config.o
SRCS = device.c driver.c daemon.c config.c keusb.c keusbd.c

VPATH = posix

all: keusb keusbd

keusb: keusb.o ${OBS1}
```



```
{CC} {CFLAGS} -o $@ keusb.o ${OBJS1}

keusbd: keusbd.o ${OBJS2}
    {CC} {CFLAGS} -o $@ keusbd.o ${OBJS2}

.c.o:
    {CC} {CFLAGS} -I ${VPATH} -c $<

clean:
    rm -f *~ *.o keusb keusbd

install:
    cp -f keusb keusbd ${BINS}
    cp -f conf.d/keusb.example /etc/conf.d
    cp -f init.d/keusbd /etc/init.d

deinstall:
    rm -f /etc/init.d/keusbd
    rm -f ${BINS}/keusb
    rm -f ${BINS}/keusbd

#
# Makefile.win32 — сценарий автоматической сборки проекта для ОС Windows
#
# Разработал: Муковников М. Ю.
# Версия:      1.0.0
# Изменён:     16.04.2010
#

CC      = gcc
CFLAGS  = -Wall -O2 -fomit-frame-pointer
LIBS     = -lwsock32
OBJS1    = driver.o device.o
OBJS2    = daemon.o config.o
SRCS     = driver.c device.c daemon.c config.c keusb.c keusbd.c

VPATH    = windows

all: keusb keusbd

keusb: keusb.o ${OBJS1}
    {CC} {CFLAGS} -o $@ keusb.o ${OBJS1}

keusbd: keusbd.o ${OBJS2}
    {CC} {CFLAGS} -o $@ keusbd.o ${OBJS2} ${LIBS}

.c.o:
    {CC} {CFLAGS} -I ${VPATH} -c $<

clean:
    rm -f *.o *~ *.log *.bak *.pid
```

```
#!/sbin/runscript
#
# init.d/keusbd — сценарий управления демоном keusbd
#
# Разработал: Муковников М. Ю.
# Версия:      1.0.0
# Изменён:     16.04.2010
#

depend() {
    need net
}

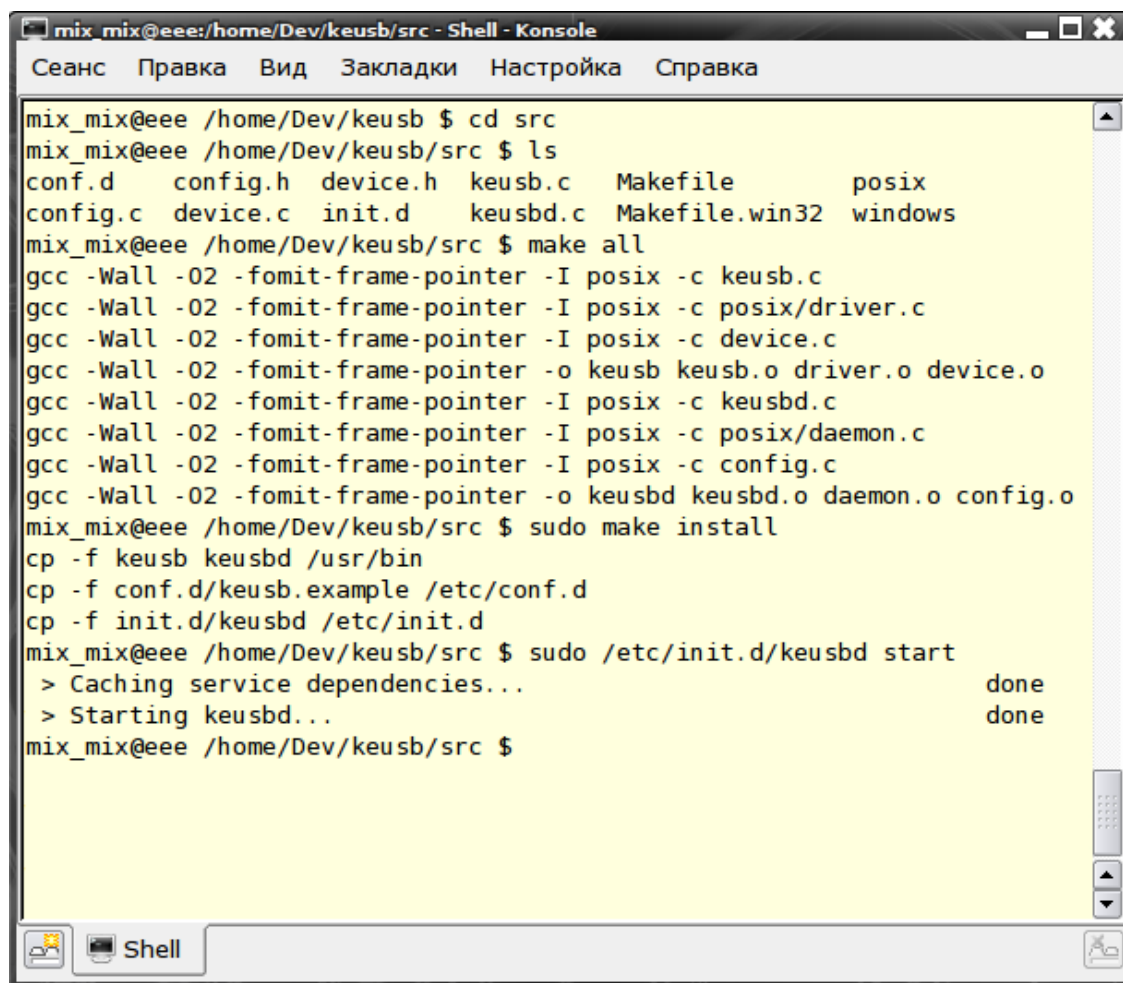
start() {
    ebegin "Starting keusbd"
    start-stop-daemon —start —quiet —exec /usr/bin/keusbd — —daemon
    eend $?
}

stop() {
    ebegin "Stopping keusbd"
    start-stop-daemon —stop —quiet —pidfile /var/run/keusbd.pid
    eend $?
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Результаты выполнения программы

Компиляция из исходных кодов, установка и запуск демона показаны на рисунке Б.1.

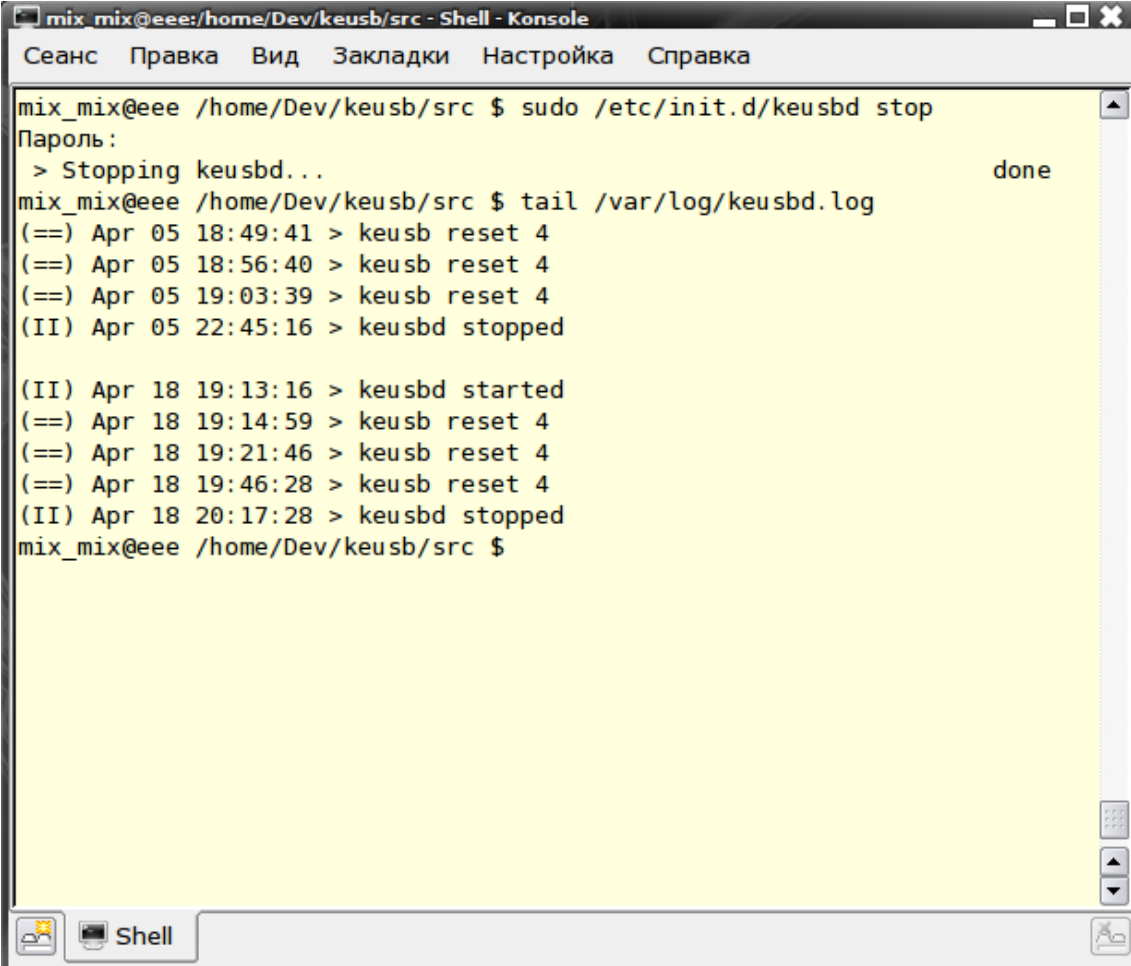


```
mix_mix@eee:/home/Dev/keusb/src - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

mix_mix@eee /home/Dev/keusb $ cd src
mix_mix@eee /home/Dev/keusb/src $ ls
conf.d  config.h  device.h  keusb.c  Makefile      posix
config.c  device.c  init.d    keusbd.c  Makefile.win32  windows
mix_mix@eee /home/Dev/keusb/src $ make all
gcc -Wall -O2 -fomit-frame-pointer -I posix -c keusb.c
gcc -Wall -O2 -fomit-frame-pointer -I posix -c posix/driver.c
gcc -Wall -O2 -fomit-frame-pointer -I posix -c device.c
gcc -Wall -O2 -fomit-frame-pointer -o keusb keusb.o driver.o device.o
gcc -Wall -O2 -fomit-frame-pointer -I posix -c keusbd.c
gcc -Wall -O2 -fomit-frame-pointer -I posix -c posix/daemon.c
gcc -Wall -O2 -fomit-frame-pointer -I posix -c config.c
gcc -Wall -O2 -fomit-frame-pointer -o keusbd keusbd.o daemon.o config.o
mix_mix@eee /home/Dev/keusb/src $ sudo make install
cp -f keusb keusbd /usr/bin
cp -f conf.d/keusb.example /etc/conf.d
cp -f init.d/keusbd /etc/init.d
mix_mix@eee /home/Dev/keusb/src $ sudo /etc/init.d/keusbd start
> Caching service dependencies... done
> Starting keusbd... done
mix_mix@eee /home/Dev/keusb/src $
```

Рисунок Б.1 – Запуск демона

Останов демона и содержимое лог-файла за час работы программы продемонстрированы на рисунке Б.2.



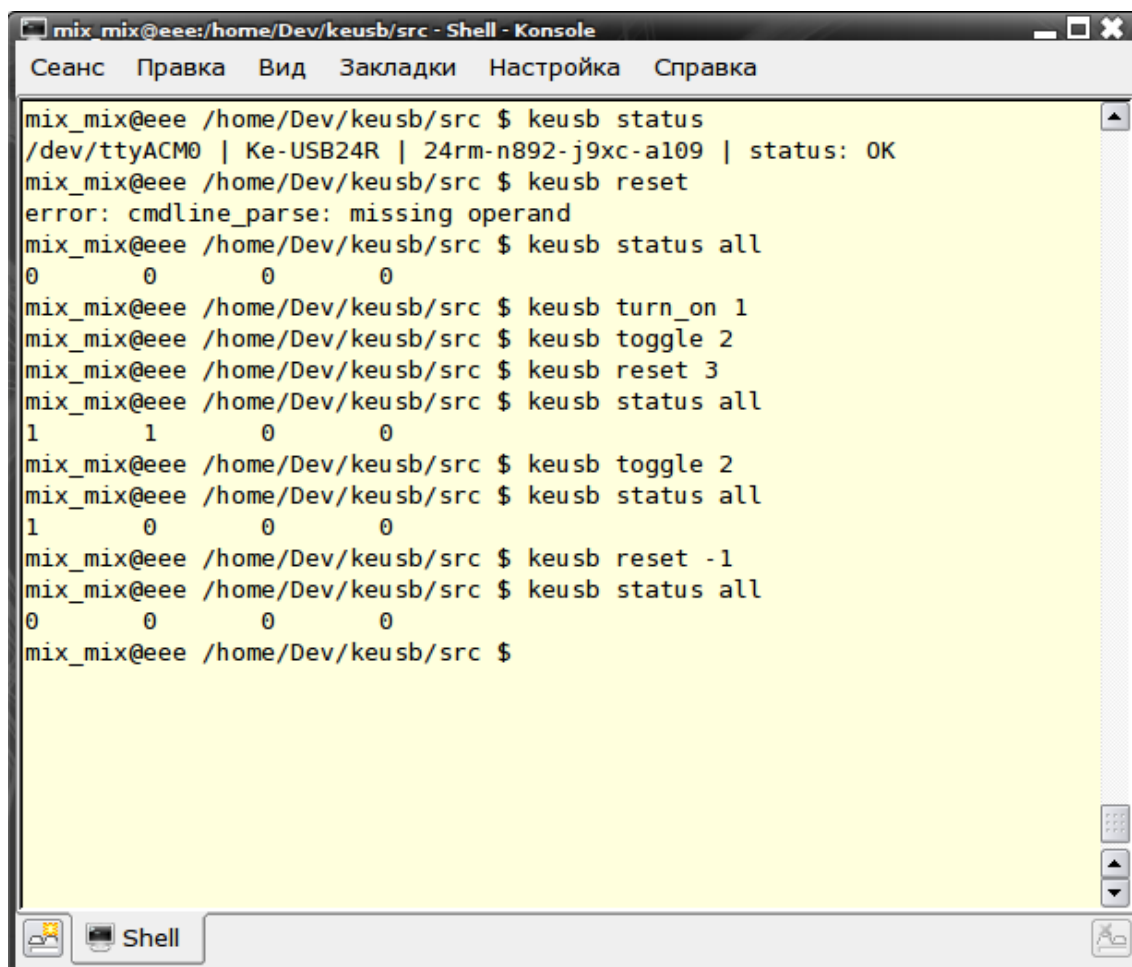
```
mix_mix@eee:/home/Dev/keusb/src - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

mix_mix@eee /home/Dev/keusb/src $ sudo /etc/init.d/keusbd stop
Пароль:
> Stopping keusbd... done
mix_mix@eee /home/Dev/keusb/src $ tail /var/log/keusbd.log
(==) Apr 05 18:49:41 > keusb reset 4
(==) Apr 05 18:56:40 > keusb reset 4
(==) Apr 05 19:03:39 > keusb reset 4
(II) Apr 05 22:45:16 > keusbd stopped

(II) Apr 18 19:13:16 > keusbd started
(==) Apr 18 19:14:59 > keusb reset 4
(==) Apr 18 19:21:46 > keusb reset 4
(==) Apr 18 19:46:28 > keusb reset 4
(II) Apr 18 20:17:28 > keusbd stopped
mix_mix@eee /home/Dev/keusb/src $
```

Рисунок Б.2 – Останов демона

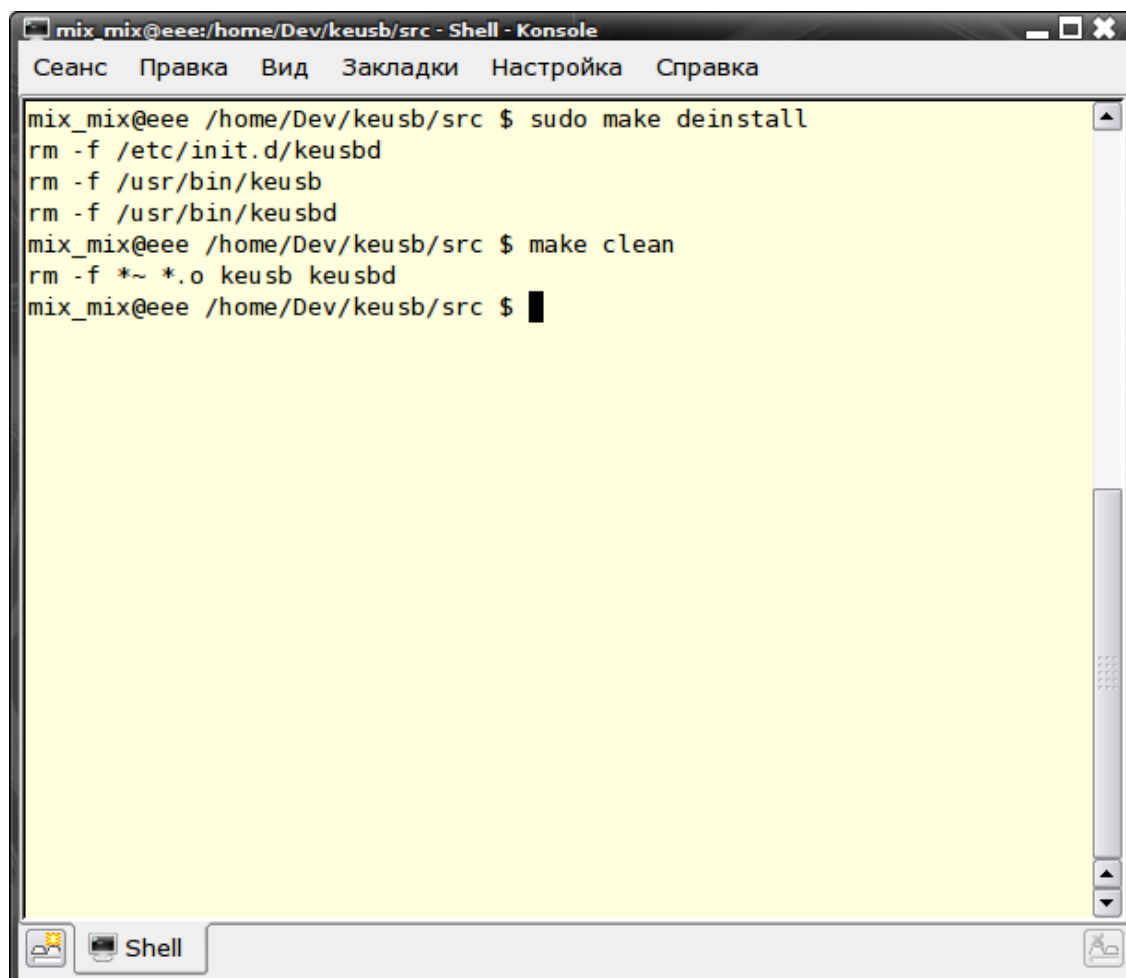
Ручное управление устройством через программу keusb продемонстрировано на рисунке Б.3.



```
mix_mix@eee /home/Dev/keusb/src $ keusb status
/dev/ttyACM0 | Ke-USB24R | 24rm-n892-j9xc-a109 | status: OK
mix_mix@eee /home/Dev/keusb/src $ keusb reset
error: cmdline_parse: missing operand
mix_mix@eee /home/Dev/keusb/src $ keusb status all
0      0      0      0
mix_mix@eee /home/Dev/keusb/src $ keusb turn_on 1
mix_mix@eee /home/Dev/keusb/src $ keusb toggle 2
mix_mix@eee /home/Dev/keusb/src $ keusb reset 3
mix_mix@eee /home/Dev/keusb/src $ keusb status all
1      1      0      0
mix_mix@eee /home/Dev/keusb/src $ keusb toggle 2
mix_mix@eee /home/Dev/keusb/src $ keusb status all
1      0      0      0
mix_mix@eee /home/Dev/keusb/src $ keusb reset -1
mix_mix@eee /home/Dev/keusb/src $ keusb status all
0      0      0      0
mix_mix@eee /home/Dev/keusb/src $
```

Рисунок Б.3 – Ручное управление устройством

Корректное удаление программ из системы и очистка дерева исходных кодов показаны на рисунке Б.4.



```
mix_mix@eee:/home/Dev/keusb/src - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

mix_mix@eee /home/Dev/keusb/src $ sudo make deinstall
rm -f /etc/init.d/keusbd
rm -f /usr/bin/keusb
rm -f /usr/bin/keusbd
mix_mix@eee /home/Dev/keusb/src $ make clean
rm -f *~ *.o keusb keusbd
mix_mix@eee /home/Dev/keusb/src $
```

Рисунок Б.4 – Деинсталляция программ из системы