Томский государственный университет
Факультет информатики
Кафедра теоретических основ информатики

Допустить к защите в ГАК
зав. кафедрой теоретических основ информатики,
кандидат технических наук, доцент
_____*Костюк Ю.Л.*
«____» _____ 2000 г.

Толмачев Григорий Владиславович

Разработка библиотек ANSI C и перенос пакета программ для операционной системы NIKOS

Дипломная работа

Научный руководитель:
_____ *Гусев И. С.*

Исполнитель:
студент группы 1451
_____ *Толмачев Г. В.*

Электронная версия отчета
помещена в библиотеку
Администратор сети _____

Томск 2000

Реферат

Отчет о дипломной работе 51 с., 4 источника, 2 приложения.

ANSI, NIKOS, ЯЗЫК ПРОГРАММИРОВАНИЯ СИ, ISO/IEC 9899:1993, МОБИЛЬНОЕ ПРОГРАММИРОВАНИЕ

Цель работы – разработка библиотек стандарта ANSI C для операционной системы NIKOS.

Достигнутые результаты – созданы следующие библиотеки стандарта ANSI C и с использованием этих библиотек перенесен на платформу ОС NIKOS редактор связей FLTLINK32; компилятор ассемблера — Netwide Assembler.

Содержание

Введение

Проблема переносимости программного обеспечения на различные операционные системы (ОС) возникла почти одновременно с их появлением.

В 1965 году сотрудники исследовательского центра по информатике фирмы Bell Laboratories Кен Томпсон, Дэннис Ричи и другие создали прототип будущей операционной системы UNIX.

Код этой ОС был полностью написан на ассемблере. Не останавливаясь на подробностях, скажем, что для данной операционной системы Кеном Томпсоном был создан язык программирования высокого уровня, названный Би (B). Би был интерпретирующим языком со всеми недостатками, присущими подобным языкам, поэтому Дэннис Ричи переделал его в новую разновидность, получившую название Си (C) и разрешающую генерировать машинный код, объявлять типы данных и определять структуры данных. В 1973 году система была написана заново на Си, благодаря чему ее легко читать, понимать, изменять и переносить на другие машины. По оценкам, сделанным Дэннисом Ричи, первый вариант системы на Си имел на 20-40 % больший объем и работал медленнее по сравнению с вариантом написанным на ассемблере, однако преимущества использования языка высокого уровня намного перевешивают недостатки [1].

Особая роль языка программирования Си состоит в том, что он, с одной стороны, позволяет писать для UNIX-систем практически столь же эффективный код, что и языки ассемблера, а с другой, является основным средством переноса программ между UNIX-системами. Можно сказать, что Си является машинно-независимым языком ассемблера для UNIX-систем [2]. Это делает его основным средством написания эффективных и переносимых программ для этого класса вычислительных систем.

Очень важным в мире операционных систем является принятый сначала ANSI, а потом и ISO международный стандарт языка

программирования Си. Дело в том, что в этом стандарте специфицирован не только непосредственно язык Си, но и библиотеки, необходимые в каждой стандартной реализации. Поскольку с самого своего появления язык Си и соответствующие системы программирования были неразрывно связаны с ОС UNIX, то состав стандартных библиотек достаточно точно соответствует стандартной среде операционной системы UNIX.

Сказанное не означает, что любая программа, написанная на ANSI C и отлаженная в одной вычислительной системе (ВС), безусловно переносима на любую другую вычислительную систему, также имеющую компилятор языка Си, отвечающий требованиям ANSI. Однако, язык ANSI C определен таким образом, чтобы можно было писать программы, подвергающиеся минимальным изменениям при их переносе на другие вычислительные системы.

Программа на ANSI C переносима из исходной вычислительной системы в целевую, если она успешно компилируется в целевой ВС и ее работа функционально эквивалентна работе в исходной ВС.

Целью данной работы является реализация стандартных библиотек ANSI C для операционной системы NIKOS. Это позволит перенести на данную операционную систему различные программы, созданные для других ОС с использованием стандарта ANSI C, в частности, компилятор языка C — LCC 4.1; компилятор ассемблера — Netwide Assembler; редактор связей для операционной системы NIKOS — FLTLINK32.

1. Операционная система NIKOS

Операционная система NIKOS разрабатывается на факультете информатики Томского Государственного Университета.

ОС NIKOS представляет собой встраиваемую многопользовательскую операционную систему с вытесняющей многозадачностью с жестким распределением времени. Система многопоточна, все процессы работают в защищенном виртуальном пространстве. Общение пользовательских процессов с ядром системы осуществляется посредством системных вызовов.

Система не является UNIX-совместимой, поскольку имеет уникальное модульное ядро — не превышающее по размеру 128 килобайт — на данный момент содержащее:

Минимальный набор функций для работы с внешними устройствами, устройствами ввода/вывода;

Уникальные алгоритмы планирования;

Функции работы с процессами, потоками;

Минимальный набор функций обеспечения безопасности системы (login, logout);

Модульность ядра является уникальной особенностью операционной системы NIKOS. При необходимости любая из подсистем ядра может быть отключена или, наоборот, включена в систему в процессе ее работы (то есть без перезагрузки системы) — устройства ввода/вывода, дисковые накопители и даже система виртуальной памяти.

В настоящее время ОС NIKOS находится в процессе своего развития, и, закономерно, возникла необходимость создания средств разработки и переноса уже наработанных программ для данной операционной системы, для чего необходимо реализовать библиотеки стандарта ANSI C.

2. Стандартные библиотеки языка Си

2.1 Файлы заголовков

Использование текстовых файлов заголовков (header-файлов), которые вставляются в текст программы на языке Си с помощью директивы `include` препроцессора Си, является традиционной техникой программирования на языке Си, обеспечивающей синтаксическую правильность использования библиотечных функций (в том числе и системных вызовов) в прикладной программе. Ранее файлы заголовков, главным образом, содержали определения типов и символических констант (констант, которым сопоставлены имена посредством директивы `define` препроцессора Си), используемых в интерфейсах соответствующих библиотечных функций. Корректное применение файлов заголовков позволяло программистам не заботиться о правильности типов данных, используемых при обращении к библиотечным функциям и обработке их результатов.

Однако, традиционные файлы заголовков не гарантировали того, что набор параметров вызываемой библиотечной функции соответствовал ее интерфейсу, поскольку объявление функции, содержащее ее интерфейс, в файле компиляции отсутствовало [2]. В лучшем случае ошибки такого рода устойчиво проявлялись во время выполнения программы, хотя далеко не всегда было просто понять их природу. В худшем случае ошибка возникала при переносе программы, поскольку одноименные библиотечные функции действительно обладали разными интерфейсами в разных средах, и в исходной операционной среде ошибки в параметрах не было.

Эту проблему удалось решить (хотя и не абсолютно) за счет введения в язык Си понятия прототипа функции. Грубо говоря, прототип функции - это часть ее объявления, содержащая только интерфейс (без тела функции).

Наличие прототипа любой функции допускается в любом файле компиляции, даже не обязательно содержащем вызов этой функции. Однако, если вызов функции содержится в файле компиляции, то набор параметров вызова должен точно соответствовать интерфейсу вызываемой функции, определенному в ее прототипе.

Дальнейший ход рассуждений очевиден. Для группы родственных библиотечных функций делается общий файл заголовков, содержащий необходимые определения типов данных и символических констант, а также набор прототипов этих библиотечных функций. После включения в файл компиляции такого файла заголовков на стадии компиляции будут обнаружены все синтаксические ошибки обращения к библиотечным функциям. Однако это решение не абсолютно. Это действительно так, поскольку в принципе никто не может заставить программиста на языке Си включать в текст программы все требуемые файлы заголовков. Пусть этот вопрос решает каждый программист в отдельности.

Последнее замечание относительно файлов заголовков. В последнее время они содержат большое количество операторов условной компиляции, относящихся большей частью к определению символических констант. Дело в том, что в зависимости от версии операционной системы значения констант, используемых с одним и тем же смыслом, часто меняются. Конечно, прикладная программа не должна зависеть от таких изменений. Наличие операторов условной компиляции внутри файла заголовков разрешает эту проблему.

2.2 О стандартных библиотеках

В языке Си стандартная библиотека более сильно интегрирована с языком по сравнению с другими языками программирования высокого уровня. Без использования функций стандартной библиотеки не может быть

написана ни одна серьезная программа на языке Си, в частности потому, что в самом языке нет никаких средств ввода/вывода информации.

Стандартную библиотеку функций языка Си можно разделить на две категории: функции, которые имеются в библиотеке любой системы программирования языка Си для различных операционных систем и различных архитектур компьютеров, и функции, которые являются уникальными в рамках какой-либо системы программирования или обеспечивают доступ к специфическим возможностям конкретной операционной системы, или связаны с конкретной архитектурой компьютера.

Функции первой категории образуют переностимое ядро библиотеки; программы, в которых используются только такие библиотечные функции, могут быть перенесены в другую систему программирования, другую операционную систему или на другой тип архитектуры компьютера с наименьшими затратами. Плата за универсальность — невозможность воспользоваться специфическими средствами, предоставляемыми конкретной вычислительной средой.

Функции второй категории предоставляют возможность получить доступ к функциям ядра данной операционной системы к внутренним структурам данной операционной системы, к регистрам используемых аппаратных устройств. Кроме того, ко второй категории относятся функции, которые добавлены в библиотеку, исходя из личных предпочтений разработчиков конкретной системы программирования — как им видится удобный набор средств для разработки различных алгоритмов — сравните, например, функции memset и setmem.

Со времени принятия стандарта языка Си, что окончательно произошло в 1989 году, в системах программирования Си такие необоснованные расширения библиотек практически исчезли, но в ранних версиях языка подобный разнобой был очень велик.

К сожалению, наборы функций второй категории не согласованы даже для различных систем программирования в рамках одной операционной системы на одном типе архитектур компьютеров. Это четко прослеживается

на примере систем программирования Turbo C от Borland и Microsoft C: библиотечные функции, обеспечивающие интерфейс для вызова одной и той же функции операционной системы, могут иметь не только разные параметры, но и разные названия [3].

Эти несогласованности объяснялись, с одной стороны, коммерческими соображениями стремлением удержать под контролем рынок программного обеспечения, чтобы пользователи, начавшие программировать с использованием одной системы программирования, покупали затем более новые программные продуктытой же фирмы, а с другой стороны, поздним появлением стандарта на язык и на его библиотеки и независимые эволюции от версии к версии каждой системы программирования [3]. (Напомним, что стандарт языка был принят примерно через пятнадцать лет после его появления и через десять лет после того, как его уже начали активно использовать) При этом, надо отметить, происходило постепенное сближение различных систем программирования по мере того, как каждая из них заимствовала наиболее ценные идеи у конкурентов. Так, различия между библиотеками последних систем программирования Turbo C и Microsoft C практически отсутствуют.

2.3 Стандарт ANSI C

В этом разделе содержатся нестрогие выдержки из стандарта ISO/IEC 9899:1993 — Programming language C [5], касающиеся некоторых соглашений о стандартных библиотеках.

1. Каждая функция описывается в библиотеке, где содержатся прототипы (описания) нескольких схожих функций (например, функции ввода/вывода) и, при необходимости, требуемые структуры данных и макроопределения. Библиотека подключается путем указания директивы препроцессора *#include* за которым следует заголовок библиотеки;

2. Стандартными заголовками являются: <assert.h> , <complex.h> , <ctype.h>, <errno.h> , <fenv.h> , <float.h> , <inttypes.h> , <iso646.h>, <limits.h> , <local.h> , <math.h> , <setjmp.h>, <signal.h> , <stdarg.h> , <stdbool.h> , <stddef.h> , <stdio.h> , <stdlib.h> , <strng.h> , <tgmath.h>, <time.h>, <wchar.h>, <wctype.h>;
3. Все стандартные баблиотеки могут включаться в программу в любом месте, предусмотренном синтаксисом языка, кроме библиотеки <assert.h> (результат зависит от значения переменной NDEBUG). Это не оказывает никакого влияния на эффективность программы. Однако корректное выполнение любой функции, может быть произведено, только если библиотека, ее описывающая, будет включена в программу раньше вызова этой функции;
4. Программа не может содержать переменных, имена которых лексически полностью совпадают с макроопределениями в какой-либо подключенной библиотеки. В противном случае, препроцессор в программе заменит имена макросов на вновь определенные, а не на библиотечные выражения;
5. Библиотеки не должны включать определения локальных переменных без указания external (external linkage).

3. Мобильное программирование

3.1 Требования к системе, поддерживающей мобильное программирование

Язык Си является одним из наиболее качественно стандартизованных языков. Поэтому программы, написанные на языке Си, при использовании правильного стиля программирования обладают весьма высоким уровнем мобильности, то есть их можно достаточно просто переносить на другие операционные системы и аппаратные платформы.

Очевидным требованием к операционной среде, поддерживающей мобильное прикладное программирование, является то, что все функции, предоставляемые ею прикладной программе, должны быть четко специфицированы и должны точно соответствовать этим спецификациям в любой реализации операционной среды [2]. Это требование может удовлетворяться за счет наличия нескольких стандартизованных библиотек функций и соответствующих наборов файлов заголовков (header-файлов), о чем было сказано в предыдущей главе.

Понятно, что при прикладном программировании используются не только библиотеки системных вызовов и ввода/вывода. Существует масса других библиотечных функций, предназначенных, например, для разнообразных преобразований форматов данных, математических вычислений и так далее. К таким библиотекам нужно относиться очень осторожно, поскольку в целях повышения эффективности соответствующие функции могут быть машинно-зависимыми и по этой причине обладать специфическими интерфейсами (хотя, скорее всего, не зависят от особенностей операционной системы). Сама по себе машинная зависимость библиотечной функции не представляет опасности, поскольку при переносе программы на компьютер с другой архитектурой все равно потребуется

перекомпиляция и перекомпоновка прикладной программы, но специфичность интерфейсов может причинить большие неприятности.

Наиболее безопасным решением на сегодняшний день (при программировании на языке Си) является использование библиотек, специфицированных в стандарте языка Си. Наверное, стандартных библиотек Си окажется недостаточно в случае сложных приложений, но если при указании опции "ANSI" ваша система программирования успешно производит сборку выполняемой программы, можно быть почти уверенным, что вы не будете иметь проблем при переносе программы на компьютер, на котором установлен компилятор стандартного языка Си.

Если стандартных библиотек оказывается недостаточно и приходится использовать функцию из некоторой дополнительной библиотеки, поддерживаемой в вашей системе, постарайтесь проверить, насколько она стандартна. Если вы не уверены в стандартности используемой функции, то лучше напишите собственную интерфейсную функцию с известным вам интерфейсом, а при переносе прикладной программы состыкуйте эту функцию (может быть, придется ее переписать) с подходящей библиотечной функцией целевой системы (однако нет гарантии, что вам удастся ее найти).

На переносимость программы также влияют особенности как аппаратного, так и программного окружения языка в исходной и в целевой вычислительной системы. Можно выделить четыре фактора, влияющих на переносимость программы:

- архитектура вычислительных систем;
- метрические ограничения компиляторов;
- алгоритмы работы компиляторов;
- особенности операционных систем.

Архитектура существенно влияет на семантику языка, а, следовательно, и на переносимость программных файлов. Во-первых, архитектура определяет множества значений арифметических типов, фиксируя тем самым семантику большинства операций языка. Во-вторых, от архитектуры, а именно, от

системы команд, зависит интерпретация операций языка, остающихся недоопределенными даже после фиксирования множеств значений соответствующих типов. В-третьих, от архитектуры зависит схема размещения данных тех или иных типов в соответствующих элементах памяти.

Даже если программа удовлетворяет всем ограничениям ANSI C и прошла стадию компиляции в исходной вычислительной системе, может случиться, что в целевой системе она эту стадию не пройдет из-за того, что некоторые метрические характеристики программы не удовлетворяют ограничениям, принятым в последней. Примерами таких характеристик являются: число уровней вложенностей составных операторов, операторов цикла и операторов выбора варианта; число описателей указателя, массива и функции, модифицирующих базовый тип в описании объекта; число выражений, вложенных друг в друга по круглым скобкам и тому подобное.

3.2 Особенности мобильного программирования на языке Си

От алгоритмов работы компилятора зависит порядок вычисления выражений, что влияет как на значения выражений, так и на вырабатываемый ими побочный эффект. Семантика многих стандартных библиотечных функций (например, функций ввода/вывода) зависит от особенностей операционной системы.

Все перечисленные факторы учтены в определении ANSI C путем фиксирования неуточняемого (стандартом) поведения программ, неопределенного поведения программ и поведения программ, определяемого реализацией [2].

- Неуточняемое поведение (unspecified behavior) - это поведение правильных программ с корректными данными в ситуациях, для которых стандарт не выдвигает никаких требований.
- Неопределенное поведение (undefined behavior) - поведение (динамически) ошибочных программ с возможно некорректными данными или объектами с неопределенными значениями, для которых стандарт не выдвигает никаких требований. Диапазон неопределенного поведения может быть очень разнообразен: от полного игнорирования ситуации с непредсказуемыми результатами до поведения (во время трансляции или выполнения) в соответствии с документацией, описывающей характеристики среды (с выдачей диагностических сообщений или без таковой); возможны случаи преждевременного завершения трансляции или вычислений (с обязательной выдачей диагностического сообщения).
- Поведение, определяемое реализацией (implementation-defined behavior) - поведение правильно написанной программы с правильными данными, которое зависит от характеристик реализации и которое должно быть документировано каждой реализацией. «

Далее мы приводим примеры неуточняемого, неопределенного и зависящего от реализации поведения программ:

- Неуточняемое поведение — вывод текста на монитор:

Ситуация, когда выдается печатный символ, а активная позиция находится в конце строки,

Ситуация, когда выдается символ «шаг назад», а активная позиция находится в начале строки,

Ситуация, когда выдается символ «горизонтальная табуляция», а активная позиция находится «на» или «за» последней определенной позицией горизонтальной табуляции,

Ситуация, когда выдается символ «вертикальная табуляция», а активная позиция находится «на» или «за» последней определенной позицией вертикальной табуляции;

- Неопределенное поведение — арифметическая операция неверна (например, деление на 0) или выдает результат, который нельзя представить в отведенном пространстве (например, переполнение);
- Поведение зависящее от реализации — имеет ли значение регистр символов, входящих в идентификаторы с внешней связью.

3.3 Основные принципы мобильного программирования

На основе сказанного мы можем сформулировать основные принципы мобильного программирования [2]:

- Если для разрабатываемой вами прикладной системы оказывается достаточным использование библиотек, специфицированных в стандарте языка ANSI C, ограничьтесь использованием этих библиотек;
- Проектируя и разрабатывая прикладную систему, убедитесь, что вы не используете системные вызовы, не входящие в стандарт;
- Если для разрабатываемой вами прикладной программы достаточно возможностей библиотеки ввода/вывода, ограничьтесь использованием этой библиотеки;
- В качестве общей рекомендации по написанию переносимых программ можно посоветовать, во-первых, безусловно избегать использования в программах языковых конструкций с неопределенным поведением, во-вторых, избегать конструкций с неуточняемым поведением в случаях, когда результат ее работы не является однозначным, и, наконец, минимизировать число

конструкций, чье поведение определяется реализацией и существенно влияет на результат работы программы;

- Другая общая рекомендация заключается в использовании возможностей препроцессора Си для локализации непереносимых фрагментов программы. Это касается использования макроимен вместо явных констант, зависящих от реализации; использования условной трансляции для включения в окончательный текст программы того или иного фрагмента в зависимости от вычислительной системы (особенно это касается конструкций, чье поведение определяется реализацией и существенно влияет на результат работы программы) и так далее.

Программа, написанная с учетом вышеперечисленных рекомендаций переностися на любую операционную систему практически без изменений ее (программы) кода.

4. Некоторые особенности реализации библиотек ASNI C

4.1 Библиотека < ctype.h >

В данную библиотеку входят функции позволяющие определить является ли символ прописной либо строчной буквой латинского алфавита, цифрой, пробелом; проеобразовать прописную букву в строчную и наоборот и так далее.

- *int isspace (int)* — является ли заданный символ пробелом.

Макрос: *#define isspace(ch) (ch==' ')*

- *int isdigit (int)* — является ли заданный символ цифрой.

Макрос: *#define isdigit(ch) ((unsigned)(ch-'0')<=9)*

- *int isalpha (int)* — является ли заданный символ буквой латинского алфавита.

Макрос: *#define isalpha(ch) (((ch >= 'a') && (ch <= 'z')) || ((ch >= 'A') && (ch<=' Z ')))*

- *int isalnum (int)* — является ли заданный символ буквой или цифрой.

Макрос: *#define isalnum(ch) (isalpha(ch)||isdigit(ch))*

- *int islower (int)* — является ли заданный символ прописной буквой латинского алфавита.

Макрос: *#define islower(ch) ((ch>='a')&&(ch<='z'))*

- *int isupper (int)* — является ли заданный символ прописной латинской буквой.

Макрос: *#define isupper(ch) ((ch>='A')&&(ch<='Z'))*

- *int isxdigit (int)* — является ли заданный символ шестнадцатеричной цифрой.

Макрос: *#define isxdigit(ch) (isdigit (ch) || ((ch >= 'a') && (ch <= 'f')) || ((ch>='A')&&(ch<='F')))*

- *int isprint (int)* — является ли заданный символ печатным.

Макрос: *#define isprint(ch) ((ch>=0x20)&&(ch<=0x7E))*

- *int iscntrl (int)* — является ли заданный символ управляющим.

Макрос: *#define isctrl(ch) ((ch==127)||((ch>=0)&&(ch<=0x31)))*

- *int ispunct (int)* — является ли заданный символ знаком пунктуации.

Макрос: *#define ispunct(ch) (isctrl(ch)||isspace(ch))*

- *int isgraph (int)* — проверка на печатный символ исключая пробел.

Макрос: *#define isgraph(ch) (isprint(ch)&&!isspace(ch))*

- *int isascii (int)* — является ли заданный символ символом из набора кодировки ASCII.

Макрос: *#define isascii(ch) ((ch>=0)&&(ch<=127))*

- *int tolower (int)* — функция, проверяющая и преобразующая в прописную букву латинского алфавита, если буква строчная. Если же заданный символ не является строчной буквой, функция возвращает первоначальное значение без каких-либо изменений.
- *int toupper (int)* — проверка и преобразование в строчную букву латинского алфавита, если буква прописная. Работает аналогично функции *tolower*.

4.2 Библиотека < string.h >

Форматные преобразования данных, работа с областями данных и строками.

Библиотека содержит функции для обработки областей памяти, которые рассматриваются как последовательность байтов. Если размер области, с которой необходимо работать задается явно, будем называть такую

область буфером. Функции из этой библиотеки позволяют сравнивать указанной число байтов из двух буферов, копировать определенное количество символов из одного буфера в другой, присваивать каждому байту в пределах заданного буфера указанное значение.

Другая часть функций этой библиотеки работает со строками. Отличие строки от буфера состоит в том, что ее размер задается не явно, а определяется первым встретившимся при просмотре строки слева направо нулевым байтом, причем считается, что этот нулевой байт также принадлежит строке. Данные функции осуществляют такие операции над строками как конкатенация строк, сравнивание строк копирование одной строки в другую, вычисление длины строки, копирование части одной строки в другую строку и прочее.

- *void *memchr(const void *s,register unsigned char c,register size_t n)* — возвращает указатель на первое вхождение заданного символа в буфере. Функция ведет поиск символа *c* в первых *n* байтах символьного массива, адрес которого определяется значением параметра *s*. Алгоритм просматривает содержимое буфера до тех пор, пока не встретит заданный символ. Если необходимый символ не найден, или *n* равно нулю возвращает значение *NULL*.
- *int memcmp (const void *s1,const void *s2,size_t n)* — функция лексикографически сравнивает первые *n* байтов областей памяти, адреса которых определяются значениями параметров *s1* и *s2*, и возвращает значение, определяющее соотношение содержимого областей: меньше нуля, если содержимое *s1* меньше, чем *s2*; ноль, если содержимое областей эквивалентно; больше нуля, если содержимое *s1* больше, чем *s2*. Алгоритм просматривает первые *n* байтов буферов. Если найдены соответственно различные символы, возвращается разность между ними. Если все символы эквивалентны друг другу, либо n равно нулю возвращается значение ноль.

- *void *memcpy (void *dst, const void *src, register size_t length)* — копирует *length* байтов из области, адрес которой определяется значением параметра *src*, в область, адрес которой определяется значением параметра *dst*. Если некоторые участки *src* и *dst* перекрываются, функция гарантирует, что байты в перекрывающемся участке скопируются раньше, чем будут перекрыты. Возвращаемое значение — значение параметра *dest*.
- *void cdecl *memmove (void *, const void *, size_t)* — копирует указанное количество символов из одного буфера в другой. Полностью идентична функции *memcpy*.
- *void *memset (void *dst, register int c, register size_t length)* — функция присваивает значение *c* первым *length* байтам области памяти, адрес которой задается значением параметра *dest*. Возвращаемое значение — значение параметра *dest*.
- *char *strcat(register char *s, register const char *append)* — конкотенация (склеивание) двух строк. Функция добавляет строку, адрес которой задается значением аргумента *append*, в конец строки, адрес которой определяется значением аргумента *s*, записывает в конец строки результата нулевой символ, и возвращает указатель на сцепленную строку (значение аргумента *s*).
- *char *strchr(register const char *s, register const char c)* — возвращает указатель на первое вхождение символа *c* в строке адрес которой задается значением аргумента *s*. Символ *c* может быть нулевым символом (*'\n'*). Функция возвращает *NULL*, если символ не найден. Алгоритм в цикле просматривает строку до тех пор, пока не найдет искомый символ, либо пока не дойдет до конца строки.
- *int strcmp(register const char *s1, register const char *s2)* — лексикографически сравнивает две строки, адреса которых задаются параметрами *s1* и *s2* и возвращает значение: меньше нуля, если содержимое *s1* меньше, чем *s2*; ноль, если содержимое областей

эквивалентно; больше нуля, если содержимое *s1* больше, чем *s2*. Принципиально, алгоритм полностью аналогичен алгоритму функции *memcmp*.

- *char *strcpy(register char *to,register const char *from)* — копирует строку, адрес которой задается аргументом *from*, включая завершающий нулевой символ, по адресу, определяемому аргументом *to*, и возвращает значение аргумента *to*.
- *size_t strspn(const char *s1,register const char *s2)* — функция возвращает индекс первого символа строки, адрес которой задается значением параметра *s1*, который не принадлежит набору символов, содержащихся в строке, адрес которой определяется значением параметра *s2*. Это значение эквивалентно длине начальной подстроки *s1*, которая содержит только символы из набора символов в строке *s2*. Нулевой символ в конце *s2* не рассматривается. Алгоритм для каждого символа *s1* перебирает все символы *s2*, пока не встретится символ из *s1* не входящий в набор символов *s2*, индекс которого и возвращается.
- *int strlen(const char *str)* — возвращает длину (в байтах) строки, адрес которой определяется значением параметра *str*. Завершающий строку нулевой символ не учитывается. Алгоритм в цикле рассматривает все символы строки, увеличивая счетчик текущей длины строки, пока не встретится нулевой символ.
- *char *strncat (char *dest, const char *source, register size_t n)* — функция добавляет не более, чем *n* первых символов из строки, адрес которой определяется значением параметра *source*, в строку, адрес которой определяется значением параметра *dest*, завершая результирующую строку нулевым символом. Алгоритм вначале сдвигает указатель *dest* на конец строки, а затем добавляет к этой строке нужное количество символов из *source* (всю строку *s2*, если n больше длины *s2*).

- *int strncmp (register const char *s1 , register const char *s2 , register size_t n)* — функция лексикографически сравнивает не более *n* символов в строках, адреса которых определяются значениями параметров *s1* и *s2* и возвращает значение, определяющее соотношение между строками: меньше нуля, если содержимое *s1* меньше, чем *s2*; ноль, если содержимое областей эквивалентно; больше нуля, если содержимое *s1* больше, чем *s2*. Алгоритм сканирует строки и в случае неравенства соответствующих символов возвращает разницу между кодами текущих символов строк *s1* и *s2*.
- *char *strncpy (char *dst, const char *src, register size_t n)* — копирует *n* символов строки, адрес которой определяется значением параметра *src*, в строку, адрес которой определяется значением параметра *dst*. Функция возвращает адрес строки-результата (значение параметра *dst*). Если значение *n* меньше, чем длина строки *src*, нулевой символ не добавляется в новую строку. Если значение *n* больше, чем длина строки *src*, то в строку-результат добавляется в конец нулевой символ.
- *char *strpbrk (register const char *s1, register const char *s2)* — функция находит первое вхождение в строке, адрес которой определяется значением аргумента *s1*, любого символа из набора символов, содержащихся в строке, адрес которой задается значением аргумента *s2*. Завершающий нулевой символ не включается в поиск. Алгоритм в двойном цикле сравнивает каждый символ строки *s1* с каждым символом строки *s2* и возвращает указатель на первое местоположение любого символа из *s1* в *s2*. Если нет общих символов возвращается значение *NULL*.
- *char *strrchr (register const char *s, register int c)* — функция находит последнее вхождение символа *c* в строке, адрес которой задается значением параметра *s*. Завершающий нулевой символ

включается в поиск. Алгоритм просматривает строку, запоминая последнее местоположение символа *c* в строке *s*. Возвращаемое значение: если заданный символ не найден — *NULL*, иначе — указатель на последнее вхождение символа *c* в строке *s*.

- *size_t strcspn (const char *s1, register const char *s2)* — функция возвращает индекс первого символа в строке, адрес которой задается значением аргумента *s1*, который принадлежит набору символов, содержащихся в строке, адрес которой задается значением аргумента *s2*. Это значение эквивалентно длине начальной подстроки *s1*, которая не содержит ни одного символа из *s2*. Завершающий нулевой символ не учитывается при поиске. Если *s1* начинается с символа из *s2*, то возвращается значение ноль. Алгоритм по своей работе напоминает работу функции *strspn*.
- *char *strstr (register const char *s,register const char *find)* — функция возвращает указатель на первое вхождение подстроки, которая содержится в символьном массиве, адрес которого значением параметра *find*, в строке, адрес которой определяется значением параметра *s*. Алгоритм в двойном цикле просматривает строку *s* и каждый символ строки *find* сравнивается с текущим символом строки *s*. Если символы не равны, указатель строки *s* сдвигается на последний символ равный первому символу *find* и все повторяется. Возвращаемые значения: значение *NULL*, если вхождение не найдено; в противном случае — указатель по строке *s* на символ, с которого начинается первое вхождение подстроки *find* в строке *s*.
- *char *strdup (const char *str)* — получает область памяти (через вызов функции *malloc*) для копирования строки и возвращает указатель на область памяти, в которую скопирована строка. Возвращается значение *NULL*, если не удалось выделить область памяти, через вызов функции *malloc*.

4.3 Библиотека < time.h >

Содержит функции работы со временем.

Стуктуры

- Typedef unsigned long time_t
- Struct tm {

  Int tm_sec — секунды

  Int tm_min — минуты

  Int tm_hour — часы (0 – 24)

  Int tm_mday — день месяца (1 – 31)

  Int tm_mon — месяц (0 – 11), январь = 0

  Int tm_year — год (текущий год минус 1900)

  Int tm_wday — день недели (0 – 6), воскресенье = 0

  Int tm_yday — день года (0 – 365), первое января = 0

  Int tm_isdst — ненулевое, если установлено Daylight Saving Time, иначе нулевое. Поскольку на данный момент в операционной системе отсутствует переменная окружения TZ, не используется.

  }

Функции

- *char * asctime (const struct tm * timeptr)* — функция преобразует время из внутреннего представления, хранящегося в структуре *tm*, в строку символов. Возвращаемое значение — указатель на строку-результат. Системные вызовы операционной системы не используются.
- *char *ctime (const time_t *time)* — преобразование времени из длинного целого в строку символов. Строка-результат содержит 26 символов и имеет следующий вид:

  Fri Jul 07 05:30:05 2000\n\0

Каждое поле имеет символьный формат. Символ новой строки (*'\n'*) и нулевой символ занимают две последние позиции в строке. Возвращаемое значение — строка-результат. Системные вызовы операционной системы не используются.

- *struct tm *gmtime (const time_t *timer)* — функция преобразует время из длинного целого в структуру. Возвращаемое значение — указатель на структуру типа *tm*. Системные вызовы операционной системы не используются.
- *time_t time (time_t *timeptr)* — функция возвращает число секунд, прошедших с момента 00:00:00 по Гринвичу 1 января 1970 года согласно часам системы. Возвращаемое значение записывается в переменную, адрес которой задается значением аргумента *timeptr*, значение аргумента может быть равным *NULL*, в этом случае запись возвращаемого значения не производится. Используются системные вызовы операционной системы: *int scGetDate (unsigned long *Data)* — получить текущую дату; *int scGetTime (unsigned long *Time)* — получить текущее время.
- *int stime (time_t *)* — функция устанавливает системное время и дату на основании переменной, адрес которой определяется значением параметра *timeptr*, трактуя значение этой переменной как количество секунд, прошедших с момента 00:00:00 по Гринвичу первого января 1970 года. Функция возвращает значение ноль. Используютя системные вызовы операционной системы: *int scSetDate(unsigned long Date)* — установить дату; *int scSetTime (unsigned long Time)* — установить время.
- *double difftime (time_t time1, time_t time2)* — функция вычисляет разницу в секундах между временем, задаваемым параметром *time2*, и временем, задаваемым параметром *time1*. Возвращаемое значение — число двойной точности, которое

является разницей в секундах от *time1* до *time2*. Системные вызовы операционной системы не используются.

4.4 Библиотека < stdlib.h >

В данной библиотеке содержатся функции отвечающие за форматное преобразование данных, генерацию псевдослучайных последовательностей чисел, работу с переменными окружения, простейшие математические функции.

Структуры

- typedef struct {
  int quot — частное от деления
  int rem — остаток от деления
  } div_t;
- typedef struct {
  long quot — частное от деления
  long rem — остаток от деления
  } ldiv_t;

Функции

- *int abs (int value)* — возвращает абсолютное значение целочисленного аргумента.
- *long int labs (long int value)* — возвращает абсолютное значение длинного целочисленного аргумента.
- *int atoi (const char *str)* — функция преобразует строку, символов *str* в целое число. Считается, что строка *str* модет быть интерпретирована как числовое значение целого типа. Функция заканчивает чтение символов из входной строки, как только встретит симвоол, который не может быть распознан, как цифра

(этим символом может быть символ конца строки). При этом пробелы в начале строки *str* игнорируются.

- *long atol (const char *str)* — функция преобразует строку, символов *str* в длинное целое число. Полностью анологична функции *atoi* за исключением типа входного аргумента и типа возвращаемого значения.
- *char *itoa (register int value, char *string, register const int radix)* — функция преобразует целое значение, заданное параметром *value* в строку символов, завершающуюся нулевым символом, и помещает результат по адресу, определяемым параметром *string*. Значение аргумента *radix* определяет систему исчисления для представления результата, значение должно быть в пределах от 0 до 36. Возвращаемый результат — указатель на строку-результат (копия значения параметра *string*).
- *char *ltoa (register long value, char *string, register const int radix)* — преобразование длинного целого в строку. Аналогична функции *itoa*.
- *char *ultoa (unsigned long, char *, int)* — преобразование беззнакового длинного целого в строку. Аналогична функции *itoa*.
- *(type) max (a, b)* — нахождение максимального числа из двух заданных. Макрос: *#define max(a,b) (((a) > (b)) ? (a) : (b))*
- *(type) min (a, b)* — нахождение минимального числа из двух заданных. Макрос: *#define min(a,b) ((a) < (b)) ?((a) : (b))*
- *div_t div (int num, int denom)* — функция предназначена для деления двух целых чисел с выделением частного и остатка от деления. Значения параметров *num* и *denom* задают делимое и делитель соответственно. Возвращаемое значение — структура из двух элементов типа *div_t* (возвращается сама структура, а не указатель на нее), которые содержат частное и остаток от деления.

- *ldiv_t ldiv (long int num, long int denom)* — деление двух длинных целых с выделением частного и остатка. Функция анлогична функции *div* за исключением различных типов операндов и возвращаемого значения.
- *int random (int num)* — получить псевдослучайное число в пределах от 0 до *num-1*. Макрос: # *define random (num) (rand()%(num))*
- *void randomize (void)* — инициализация датчика случайных чисел. Макрос: *#define randomize() srand((unsigned)time(NULL))*
- *int rand (void)* — функция возвращает псевдослучайное целое значение в пределах от 0 до 32767. Перед первым обращением датчик случайных чисел необходимо проинициализировать посредством вызова функции *srand*. Используется датчик случайных чисел с периодом 2^{32}.
- *void srand (unsigned num)* — устанавливает начальное значение для генератора псевдослучайных чисел.
- *void abort (void)* — выводит сообщение Abnormal programm termination (ненормальное завершение программы) в поток stderr, затем прерывает программу, посредством вызова функции exut() с кодом завершения, равным 3. Функция использует системные вызовы операционной системы — *int scTerminateProcess (unsigned long TerminateCode)* — завершение процесса.
- *void exit (int status)* — функция завершения программы. Значение аргумента status равное нулю сигнализирует о нормальном завершении процесса. Вызывается функция *fcloseall()*, затем — системная функция *int scTerminateProcess (unsigned long TerminateCode)* — завершение процесса. Функция *fcloseall* не реализована, поскольку для нее нет аналога в системных функциях,

а создавать дополнительные таблицы открытых файлов (частично дублирующие системные таблицы) не рационально.

4.5 Библиотека < stdio.h >

Стандартная библиотека ввода/вывода — библиотека функций более высокого уровня, чем библиотека системных вызовов. Основной набор функций этой библиотеки служит для выполнения файловых операций с буферизацией данных в памяти пользовательского процесса. Библиотека ввода/вывода фактически стандартизована очень давно, и ей можно безопасно пользоваться в любой операционной среде. В частности, единообразные библиотеки ввода/вывода поддерживаются во всех современных реализациях системы программирования языка Си.

Структуры

- struct __sbuf { — буфер для операций ввода/вывода.
  unsigned char *Base
  unsigned long Size
  }
- typedef struct __sFILE {
  long FileID — идентификатор файла;
  unsigned flags — флаги состояния файла;
  struct __sbuf Buffer — буфер для кэширования ввола/вывода;
  unsigned long ActBufDataSize — размер актуальных данных в буфере;
  unsigned long CurBufOfs — текущее смещение в буфере;
  struct __sbuf UngetBuffer — буфер для ungets();
  unsigned long CurUgBufOfs — текущее смещение в буфере;
  unsigned long ubuf — гарантированный буфер для ungets();
  } FILE

Функции

- *int putchar (int c)* — запись символа в стандартный поток вывода *stdout*. Макрос: *#define putchar(c) f putchar((c))*.
- *int getchar (void)* — чтение символа из стандартного потока ввода *stdin*. Макрос: *#define getchar () fgetchar()*.
- *int fgetchar (void)* — чтение символа из стандартного потока ввода *stdin*. Аналогична *getchar,* но является функцией. Функции с разными именами, но выполняющие одинаковые действия были оставлены в стандарте для совместимости с системами программирования разных производителей (*putchar–fputchar, getc–fgetc*). Используемые системные вызовы: *void csReadKey(char *AscCode, char *ScanCode)* — прочитать символ с клавиатуры.
- *int fputchar (register int c)* — запись символа в стандартный поток вывода stdout. Аналогична *putchar,* но является функцией. Используемые системные вызовы: *void scCharOut(char value)* — вывод символа на монитор.
- *int fclose (register FILE *f)* — функция закрывает поток, связанный с открытым при помощи функции *fopen* файлом. Все буфера, связанные с потоком сбрасываются перед закрытием. Возвращаемое значение — ноль, если закрытие прошло успешно, иначе возвращается значение EOF. Функция использует системные вызовы операционной системы: *int scCloseFile (long Handle).*
- *int fileno (FILE *)* — функция возвращает значение дескриптора *FileID* (используемого ситемными функциями ввода – вывода), связанного с заданным указателем потока stream (используемого функциями ввода – вывода верхнего уровня). Поскольку в операционной системе *NIKOS* дескрипторы потока имеют тип длинного целого, а в стандарте *ANSI C* данная функция возвращает целое значение, происходит приведение типов (из *long int* в *int*). На данный момент, так как операционная система находится в процессе

своего становления, это не играет важной роли. Однако пользоваться данной функцией необходимо с большой осторожностью.

- *int remove (const char *Pathname)* — функция удаляет файл или каталог имя которого определяется строкой, адрес которой задается параметром *Pathname*. Макрос: *remove(Pathname) scDeleteDirFile(PathName)*.
- *int unlink (register const char *PathName)* — функция аналогична remove, но является функцией. Используется вызов системной функции операционной системы — *int scDeleteDirFile(char *PathName)* — удаление файла или каталога.
- *int rename (register const char *PathName1, register const char *PathName2)* — функция переименовывает файл или каталог, имя которого определяется символьной строкой, адрес которой задается значением параметра *PathName1*, давая файлу новое имя, определяемое строкой, адрес которой задается значением параметра *PathName2*. Строка *PathName1* должна определять путь существующего файла или каталога. Строка *PathName2* не должна определять путь существующего файла или каталога. Функция может быть использована для перемещения файла из одного каталога в другой. Используется вызов системной функции операционной системы — *int scRenMoveFile(const char *PathName1, consr char *PathName2)* — перемещение и переименование файла.
- *int printf (const char *format [, argument, ...])* — печатает символы, формирует и печатает задаваемые аргументами значения в стандартный поток вывода *stdout*. Имеет переменное число параметров. Более подробно строка описания формата вывода, адрес которой задается единственным параметром *format*, определяется в описании функции *vfprintf*, которое дается ниже. Алгоритм работы:

переменные параметры *[, arguments, ...]* преобразуются к типу *va_list* и вызывается фукнция *vprintf*.

- *int vprintf (const char *format, va_list ap)* — печатает символы, формирует и печатает задаваемые аргументами значения в стандартный поток вывода *stdout.* Функция подменяет два макроопределения, которые затем используются при вызове функции *vfprintf*:
    - *#define PRINT_CHAR(c) scCharOut(c)* — системная функция вывода символа на монитор;
    - *#define PRINT_STRING(s) scStringOut(s)* — системная функция вывода строки символов, адрес которой определяется параметром *s,* на монитор.
- *int fprintf (FILE *stream, const char * format[, argument, ...])* — печатает символы, формирует и печатает задаваемые аргументами значения в поток stream. Имеет переменное число параметров. Более подробно строка описания формата вывода *format* дается ниже. Алгоритм работы: переменные параметры *[, arguments, ...]* преобразуются к типу *va_list* и вызывается фукнция *vfprintf*.
- *int sprintf (char *str, const char *format [, argument, ...])* — функция форматирует и выводит последовательности символов и значений аргументов в символьный массив, адрес которого задается значением аргумента *str*. Каждый аргумент преобразуется и выводится согласно соответствующей спецификации формата в строке, адрес которой задается значением параметра *format*. Строка описания формата вывода интерпретируется так же, как и для функции *vfprintf* (описание ниже). Функция преобразует переменные параметры *[, arguments, ...]* к типу *va_list* и вызывается фукнция *vsprintf*.
- *int vsprintf (char *str, const char *format, va_list ap)* — печатает символы, формирует и печатает задаваемые аргументами значения в

символьный массив, адрес которого определяется параметром *str*. Функция создает мнимый буферизированный поток (структуру типа *FILE*), в которой в качестве значения поля *Buffer.Base* устанавливается указатель на *str*. В итоге вывод данных в этот поток приведет к их помещению в *str*. Затем вызывается функция *vfprintf*.

- *int vfprintf (FILE *stream, const char *format, va_list ap)* — функция печатает задаваемые аргументами значения в поток *stream*. Функция имеет переменное число параметров. Строка описания формата вывода, адрес которой задается значением единственного обязательного параметра *format*, состоит из обычных символов, специальных управляющих последовательностей символов (*escape*-последовательностей) и, если за параметром *format* следуют еще дополнительные аргументы, спецификаций полей формата вывода по одному для каждого дополнительного аргумента.

Спецификации формата начинаются с символа процента. Строка описания формата прочитывается слева-направо. Когда встречается первая спецификация формата, значение первого аргумента после параметра *format* преобразуется и выводится согласно спецификации формата. Вторая спецификация формата преобразует и выводит второй аргумент, и, таким образом, обработка продолжается до конца *format*. Если задается больше аргументов, чем спецификаций формата, лишние аргументы игнорируются.

Спецификация формата имеет следущую форму:

% [flags] [width] [/precision] [h | l] type

Спецификация формата не содержит внутри себя пробелов, каждое поле спецификации формата есть одиночнгый символ или число, означающие необязательный параметр формата

Flags — выворнивание выводных символов, управление печатью знаковых символов, пробелов, десятичных точек, восьмеричных и шестнадцатеричных префиксов.

Width — минимальное число выводимых символов

Precision — максимальное число символов, которые будут напечатаны для всех или части выводных полей; или минимальное число цифр, которые будут печататься для значения целого

h, l — размер аргумента (*h* – короткое целое ,*l* длинное целое (используется как префикс с целыми типами *d i o u x X*).

type — тип выводимого в поток аргумента. Более подробную информацию о поле type можно получить из описания стандарта ANSI C [4] либо из другой литературы.

Единственное на сегодняшний день средство создания программ для *ОС NIKOS* — редактор связей *fltlnk32* — не позволяет работать с плавающей арифметикой. Поэтому вывод чисел с плавающей точкой не реализован. Алгоритм рассматривает строку *format* и соответственно преобразует список аргументов *ap*, затем выполняет функции, описанные в соответствующем макроопределении — *PUT_CHAR, PUT_STRING*. При этом происходит вывод информации либо в стандартный поток вывода *stdout*, либо если, макросы не определены, в поток *stream*. При выводе в поток, используются системные вызовы операционной системы: *int scWriteFile(long Handle, char *Buffer, long *Size)* — системная функция записи буфера в файл.

- *int scanf (const char * format [, address, ...])* — функция читает данные из стандартного ввода *stdin* в переменные, адреса которых задаются параметрами *[, address, ...]*. Строка формата подробнее описана ниже, при описании функции *vfscanf*. Функция преобразует параметры *[, address, ...]* в аргумент типа *va_list*, затем вызовает функцию *vscanf*.
- *int vscanf (const char *format, va_list ap)* — функция читает данные из стандартного ввода *stdin* в переменные, адреса которых задаются

параметром *ap*. Подменяется функция, которая затем используется при вызове функции *vfscanf*:

- *char GET_CHAR(void) { char ScanCode, AscCode; void csReadKey(&AscCode, &ScanCode); return(AcsCode); }* — прочитать символ с клавиатуры.

- *int fscanf (FILE *stream, const char *format [, address, ...])* — функция читает данные из указанного входного потока *stream*, выполняет форматные преобразования и полученные значения значения записывает в переменные, адреса которых задаются параметрами *[, address, ...]*. Функция преобразует параметры *[, address, ...]* в аргумент типа *va_list*, затем вызовает функцию *vfscanf*.
- *int sscanf (const char *s, const char *format [, address, ...])* — функция читает данные из символьного массива, адрес которого задается значением параметра *s*, в переменные, адреса которых задаются аргументами *[, address, ...]*. Каждый аргумент должен быть указателем на переменную с типом, соответствующим типу, определенному в строке описания формата *format*. Функция преобразует переменные параметры *[, address, ...]* к типу *va_list* и вызывается фукнция *vsscacnf*.
- *int vsscanf (const char *s, const char *format, va_list ap)* — функция яитает данные из строки, адрес которой задается аргументом *s*, в переменные *ap*. Строка описания формата управляет интерпретацией входных полей и имеет тот же вид, что и для функции *vfscanf*. Затем создается мнимый буферизированный поток (структуру типа *FILE*), в которой в качестве значения поля *Buffer.Base* устанавливается указатель на *s*. В итоге ввод данных из этот поток приведет к обращению к строке *s*. Затем вызывается функция *vfcanf*.
- *int vfscanf (FILE *stream, const char *format, va_list ap)* — читает данные из вводного потока *stream* в переменные, содержащиеся в

параметре *ap*. Значение первого параметра *format* задает адрес строки, которая управляет интерпритацией элементов ввода (вводных полей). Строка описания формата может содержать:

- пробельные символы — пробел (‘ ‘), табуляция (*'\t'*), символ новой строки (*‘\n’*). Если пробельный символ встретился в строке format, то с этого момента пробельные символы до первого не являющегося пробельным игнорируются;
- печатные символы — все прочие ASCII-символы, за исключением символа *‘%’,* Если печатные символы встречаются в строке format, то из потока ввода считываются символы, которые не участвуют в выработке присваиваемых переменным значений, но которые должны соответствовать печатным символам строки format. Если последовательность печатных символов из потока ввода не соответствует последовательности печатных символов в format, работа функции прекращается;
- спецификация формата — знак *‘%’*. Почти полностью повторяет описание формата для функции *vfprintf*. Более подробное и полное описание моджно узнать из стандарта *ANSI C* [4].

Алгоритм рассматривает строку *format* и соответственно изменяет список аргументов *ap* При этом происходит ввод информации либо из стандартного потока ввода *stdin*, либо, в зависимости от определения функции *GET_CHAR()*, из потока *stream*. При вводе из потока, используются системные вызовы операционной системы: *int scReadFile(long Handle, char *Buffer, long *Size)* — системная функция чтения буфера из файла.

- *int fgetc (FILE *stream)* — функция читает один символ из вводного потока stream с текущей позиции и перемещает указатель файла на следующий символ. Используется системный вызов

функции операционной системы — *int scReadFile(long Handle, char *Buffer, long *Size)* — чтение буфера из файла.

- *int getc (FILE *stream)* — эквивалентна функции *fgetc*. Макрос: *#define putc(c,stream) fgetc(stream)*.
- *char* gets (char *s)* — чтение строки из стандартного потока ввода *stdin*. В строку считываются все символы, до первого встретившегося символа новой строки (*'n'*), не включая его. При этом используется системный вызов функции операционной системы — *void csReadKey(char *AscCode, char *ScanCode)* — прочитать символ с клавиатуры. Если строка считана удачно возвращается адрес буфера, иначе — *NULL*.
- *int fputc (int c, FILE *stream)* — записывает одиночный символ *c* в поток *stream* в текущую позицию. Используется системная функция *int scWriteFile(long Handle, char *Buffer, long *Size)* — запись буфера в файл.
- *int putc (int c, FILE *stream)* — эквивалентна функции fputc. Макрос: *#define putc(c,stream) fputc(c,stream)*.
- *int puts (const char *s)* — запись строки в стандартный поток вывода *stdout*. Макрос: *#define puts(s) scStringOut(s)* — вывод строки на монитор.
- *int fputs (const char *s, FILE *stream)* — функция копирует строку *s* в поток *stream* с текущей позиции. Завершающий нулевой символ (*'\0'*) не копируется. Используется функция *int scWriteFile(long Handle, char *Buffer, long *Size)* — запись буфера в файл.
- *char* fgets (char *s, int n, FILE *stream)* — функция читает строку из входного потока *stream* и помещает ее в строку, адрес которой задается параметром *s*. Символы читаются из потока до тех пор, пока не будет прочитан символ новой строки (*'\n'*), который включается в строку, или пока не наступит конец потока, или пока

не будет прочитано (*n-1*) символов. Результат помещается в *s* и заканчивается нулевым символом. Если *n* равно *1*, то формируется пустая строка. Используется функция *int scReadFile(long Handle, char *Buffer, long *Size)* — чтение буфера из файла.

- *size_t fread (void *buffer, size_t size, size_t count, FILE *stream)* — функция читает *count* элементов длины *size* из входного потока *stream* и помещает их в заданный массив *buffer*. Указатель файла, связанный с потоком *stream*, увеличивается на число действительно прочитанных байтов. Форматных преобразований данных не производится. Символы перевода строки специально не обрабатываются. Используется функция *int scReadFile(long Handle, char *Buffer, long *Size)* — чтение буфера из файла. Возвращаемое значение — количество реально прочитанных элементов из файла.
- *size_t fwrite (const void *buffer, size_t size, size_t count, FILE *stream)* — функция дописывает *count* записей по *size* байтов в выходной поток *stream*. Указатель текущей позиции в файле увеличивается на число записанных байтов. Форматных преобразований данных не производится. Символы перевода строки специально не обрабатываются. Используется системная функция *int scWriteFile(long Handle, char *Buffer, long *Size)* — запись буфера в файл. Возвращаемое значение — количество реально помещенных в файл записей.
- *void setbuf (FILE *stream, char *buffer)* — функция позволяет управлять буферизацией заданного потока *stream*. Значение аргумента *stream* должно соответствовать ранее открытому файлу (функция *fopen*). Если значение элемента *beffer* равняется *NULL*, то это означает отмену буферизации. Иначе, значение аргумента *buffer* определяет адрес массива символов длины *BUFSIZ*, где *BUFSIZ* – размер буфера по умолчанию (определяется в файле *stdio.h* и равен *1024* байтов).

- *int setvbuf (FILE *stream, char *buffer, int type, size_t size)* — позволяет управлять буферизацией файла и размером буфера файла, связанного спотоком *stream*. Поток *stream* должен относиться к открытому файлу. Массив, адрес которого задается параметром *buffer*, используется в качестве буфера, если это значение параметра не *NULL*, в последнем случае поток не буферизуется. Если поток буферизуется, то значение параметра *type* определяет тип буферизации:
    - *_IOFBF* — буферизация на полный объем буфера. Значение параметра *size* используется как размер буфера;
    - *_IOLBF* — построчная буферизация: появление символа новой строки приводит к скидыванию буфера при записи в файл,
    - *_IONBF* — файл не буферизуется. Значения параметров *size* и *buffer* игнорируются.
- *FILE * fopen (const char *pathname, const char *type)* — функция открывает файл, имя которого задается аргументом *pathname*, и связывает с ним поток. Аргумент *type* указывает на строку символов, определяющих способ доступа к файлу: *“r”* – открыть существующий файл для чтения; *“w”* – открыть пустой файл для записи, если файл существует, то его содержимое теряется; *“a”* – открыть файл для добавления (для записи в конец файла); файл создается, если он не существует. Используются вызовы системных функций операционной системы — *int scCreateFile(char *PathName)* — создание файла, *int scOpenFile(char * PathName, long Handle, long OpenType)* — открытие файла.
- *FILE* fdopen (int handle , char *type)* — функция связывает поток ввода/вывода с файлом, связанным с дескриптором *handle*. Таким образом, сущесивует возможность для файлов, открытых при помощи системных функций начать буферизацию и работать с ними

через функции форматированного ввода/вывода. Аргумент *type* анологичен одноименному аргументу функции *fopen*. Но, поскольку, дескрипторы файлов в операционной системе *NIKOS* описываются как длянные целые, а в стандарте *ANSI C* — как целые, то после приведения типов возможно возникновение некоторых проблем.

- *int fseek (FILE *stream, long offset, int origin)* — функция перемещает указатель файла, связанного с потоком *stream* на новое место в файле, которое вычисляется по смещению *offset* и указанию направления отсчета *origin*. Следущая операция ввода/вывода с указанным потоком *stream* будет выполнена, начиная с той позиции, на которую произведено перемещение. Аргумент *origin* может быть одной из следующих констант: *SEEK_SET* – начало файла, *SEEK_CUR* – текущая позиция указателя файла, *SEEK_END* – конец файла. Используемые вызовы системных функций — *int scSeekFile(long Handle, long LowPos, long HighPos, long Type)* — установка позиции в файле.
- *long ftell (FILE *stream)* — функция позволяет получить текущую позицию указателя файла, связанного с потоком *stream*. Позиция задается как смещение, относительно начала файла.
- *void rewind (FILE *stream)* — функция перемещает указатель файла, связанного с потоком *stream* на начало файла. Вызов функции *rewind* эквивалентен вызову функции *fseek(stream,0L,SEEK_SET)*, с той лишь разницей, что данная функция не возвращает значения.
- *int feof (FILE *stream)* — функция определяет, достигнут ли конец потока *stream*. Если конец потока достигнут, операция чтения будет возвращать символ конца файла (*EOF*) до тех пор, пока поток не будет закрыт или пока не будеи вызвана функция *rewind*.

Заключение

В результате проделанной работы было реализовано около ста наиболее часто используемых функций стандарта ANSI C для операционной системы NIKOS. Однако это далеко не все функции, описанные в данном стандарте.

Тем не менее, реализация описанных библиотек позволила перенести на платформу ОС NIKOS редактор связей FLTLINK32 и компилятор ассемблера Netwide Assembler. Перенос данных программ на новую операционную систему не составил большого труда, поскольку они созданы с учетом всех принципов мобильного программирования, о которых было рассказано в третьей главе.

Литература

1. **Бах М. Дж.** Архитектура операционной системы UNIX. Перевод с английского Крюкова А.В. — Нью-Джерси: Prentice-Hall Corp. – Simon & Schuster, 1986. — 400 с.
2. **Кузнецов С. Д.** Операционная система UNIX. — www.doc.trecom. tomsk.su/win/operating_system/unix/
3. **Бочков С. О. Субботин Д. М.** Язык программирования для первонального компьютера. — М.: Радио и связь, 1990. — 384 с.
4. ISO/IEC 9899:1993 Programming languages – C. — 582 с.

Приложение 1. Руководство программиста

Библиотека < string.h >. Список реализованных функций

void **memchr (const void *, int, size_t)* — возвращает указатель на первое вхождение заданного символа в буфере;

int *memcmp (const void *, const void *, size_t)* — сравнивает указанное число символов в буфере;

void **memcpy (void *, const void *, size_t)* — копирует указанное количество из одного буфера в другой;

void **memmove (void *, const void *, size_t)* — копирует указанное количество символов из одного буфера в другой;

void **memset (void *, int, size_t)* — инициализирует заданным значением указанное количество байтов в буфере;

char **strcat (char *, const char *)* — конкотенация (склеивание) двух строк;

char **strchr (const char *, int)* — находит первое вхождение символа в строке;

int *strcmp (const char *, const char *)* — сравнивает две строки;

char **strcpy (char *, const char *)* — копирует одну строку в другую;

size_t *strcspn (const char *, const char *)* — находит первое вхождение символа из заданного набора символов в строке;

size_t *strlen (const char *)* — вычислить длину строки;

char **strncat (char *, const char *, size_t)* — добавить заданное количество символов в строку;

int *strncmp (const char *, const char *, size_t)* — сравнить заданное количество символов в двух строках;

char **strncpy (char *, const char *, size_t)* — скопировать заданное количество символов из одной строки в другую;

*char *strpbrk (const char *, const char *)* — найти первое вхождение любого символа из заданного набора в строке;

*char *strrchr (const char *, int)* — найти последнее вхождение заданного символа в строке;

*size_t strspn (const char *, const char *)* — найти первую подстроку из заданного символов в строке;

*char *strstr (const char *, const char *)* — найти первую подстановку одной строки (более короткой) в другой;

*char *strdup (const char *)* — дублирование строки.

Библиотека < ctype.h >. Список реализованных функций

int isalnum (int) — проверка, является ли заданный символ буквой или цифрой;

int islower (int) — проверка, явлиется ли заданный символ прописной буквой латинского алфавита;

int isalpha (int) — проверка, является ли заданный символ буквой латинского алфавита (прописной или строчной);

int isprint (int) — проверка,явлиется ли заданный символ печатным, то есть его код находится в пределах от 32 до 127;

int isascii (int) — провевка, является ли заданный символ символом из набора кодировки ASCII;

int ispunct (int) — проверка, является ли заданный символ знаком пунктуации;

int iscntrl (int) — проверка, является ли заданный символ управляющис, то есть его код находится в пределах от 0x00h до 0x1Fh либо равен 0x7Fh;

int isspace (int) — проверка, является ли заданный символ пробелом;

int isdigit (int) — проверка, является ли заданный символ цифрой;

int isupper (int) — проверка, является ли забанный символ прописной латинской буквой;

int isgraph (int) — проверка на печатный символ исключая пробел;

int isxdigit (int) — проверка, является ли заданный символ шестнадцатеричной цифрой, то есть одним из символов: '0' – '9', 'a' – ‘f’, ‘A’ – ‘F’;

int tolower (int) — проверка и преобразование в прописную букву латинского алфавита, если буква строчная;

int toupper (int) — проверка и преобразование в строчную букву латинского алфавита, если буква прописная.

Библиотека < time.h >. Список реализованных функций

*char *asctime (const struct tm *)* — преобразование ывремени из структуры (внутренней формы) в символьную строку;

*char *ctime (const time_t *)* — преобразование времени из длинного целого (long int) в строку символов;

*struct tm *gmtime (const time_t *)* — преобразование времени из целого (int) в структуру;

*time_t time (time_t *)* — получить текущее системное время как длинное целое (long int);

*int stime (time_t *)* — установить системное время;

double difftime (time_t, time_t) — вычисляет сколько времени прошло между заданными промежутками времени;

Библиотека <stdlib.h>. Список реализованных функций

void abort (void) — завершить процесс;

int abs (int) — получение абсолютного значения (модуля) целого (int) числа;

*int atoi (const char *)* — преобразование строки в целое (int);

*long atol (const char *)* — преобразование строки в длинное целое (long int);

void exit (int) — завершить программу;

*char *itoa (int, char *, int)* — преобразование целого (int) в строку;

long int labs (long int) — получение абсолютного значения (модуля) длинного целого (long int) числа;

*char * ltoa (long, char *, int)* — преобразование длинного целого (long int) в строку;

(type) max (a, b) — нахождение максимального числа из двух заданных;

(type) min (a, b) — нахождение минимального числа из двух заданных;

*char *ultoa (unsigned long, char *, int)* — преобразование длинного беззнакового целого (unsigned long int) в строку.

div_t div (int, int) — деление двух целых (int) с выделением частного и остатка;

ldiv_t ldiv (long int, long int) — деление двух длинных целых (long int) с выделением частного и остатка;

int rand (void) — получить псевдослучайное число в пределах от 0 до 32767;

int random (int) — получить псевдослучайное число в пределах от 0 до заданного значения минус один;

void randomize (void) — инициализация датчика случайных чисел;

void srand (unsigned) — инициализация датчика случайных чисел, устанавливает первое значение последовательности псевдослучайных чисел;

Библиотека < stdio.h >. Список реализованных функций

*int fclose (FILE *)* — закрытие потока;

FILE fdopen (int , char *)* — создание потока для файла, ранее открытого на нижнем уровне, используя дескриптор;

*int feof (FILE *)* — проверка на конец потока;

*int fgetc (FILE *)* — чтение символа из потока;

int fgetchar (void) — чтение символа из стандартного потока ввода stdin;

char fgets (char *, int, FILE *)* — чтение строки из потока;

*int fileno (FILE *)* — получение дескриптора файла, связанного с потоком;

*FILE * fopen (const char *, const char *)* — открытие потока (открыть файл и связать его с потоком);

*int fprintf (FILE *, const char * [, argument, ...])* — запись данных в поток по формату;

*int fputc (int, FILE *)* — запись символа в поток;

int fputchar (int) — запись символа в стандартный поток вывода stdout;

*int fputs (const char *, FILE *)* — запись строки в поток;

*size_t fread (void *, size_t, size_t, FILE *)* — неформатированное чтение данных из потока;

*int fscanf (FILE *, const char * [, address, ...])* — чтение из потока по формату;

*int fseek (FILE *, long, int)* — перемещение указателя файла в заданную позицию;

*long ftell (FILE *)* — получение текущей позиции указателя файла;

*size_t fwrite (const void *, size_t, size_t, FILE*)* — неформатированная запись данных в поток;

*int getc (FILE *)* — чтение символа из потока;

int getchar (void) — чтение символа из стандартного потока ввода stdin;

char gets (char *)* — чтение строки из стандартного потока ввода stdin;

*int printf (const char * [, argument, ...])* — запись данных в стандартный поток вывода stdout по формату;

*int putc (int, FILE *)* — запись символа в поток;

int putchar (int) — запись символа в стандартный поток вывода stdout;

*int puts (const char *)* — запись строки в стандартный поток вывода stdout;

*int remove (const char *)* — удаление файла с заданным именем;

*int rename (const char *, const char *)* — переименование файла или каталога;

*void rewind (FILE *)* — установка указателя файла на начало;

*int scanf (const char * [, address, ...])* — чтение данных из стандартного потока ввода stdin по формату;

*void setbuf (FILE *, char *)* — управление буферизацией потока;

*int setvbuf (FILE *, char *, int , size_t)* — управление буферизацией потока и размеров буфера;

*int sprintf (char *, const char * [, argument, ...])* — запись данных в строку по формату;

*int sscanf (const char *, const char * [, address, ...])* — чтение данных из строки по формату;

*int unlink (const char *)* — удаление файла с заданным именем;

*int vfprintf (FILE *, const char *, va_list)* — запись данных в поток по формату;

*int vfscanf (FILE *,const char *, va_list)* — чтение данных из потока по формату;

*int vprintf (const char *, va_list)* — запись данных в стандартный поток вывода stdout по формату;

*int vscanf (const char *, va_list)* — чтение данных из стандартного потока ввода stdin по формату;

*int vsprintf (char *, const char *, va_list)* — запись данных в строку по формату;

*int vsscanf (const char *, const char *, va_list)* — чтение данных из строки по формату.

Приложение 2. Список прилагаемых файлов

Stdio.h, Stdio.c — файлы стандартной библиотеки ввода/вывода (*stdio*).

Stdlib.h, Stdlib.c — файлы библиотеки общих утилит (*stdlib*).

Time.h, Time.c — файлы библиотеки работы со временем (*time*).

Ctype.h, Ctype.c — файлы библиотеки определения класса символов и преобразования символов (*ctype*)..

String.h , String.c — файлы библиотеки работы со строками (*string*).

Vfprintf.c — файл, содержащий код функции vfprintf. Все функции форматного вывода реализованы через эту функцию.

Vfscanf.c — файл, содержащий код функции vfscanf. Все функции форматного ввода реализованы через эту функцию.