Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

Кафедра инженерной кибернетики

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ПРОГРАММИРОВАНИЕ

Методические указания по выполнению лабораторных работ
для студентов специальности 5В070200 – Автоматизация и управление

Алматы 2015

СОСТАВИТЕЛЬ: Н.В.Сябина. Системное программное обеспечение и программирование. Методические указания по выполнению лабораторных работ для студентов специальности 5В070200 – Автоматизация и управление. – Алматы: АУЭС, 2015. – 35 с.

Настоящие методические указания содержат 8 лабораторных работ по дисциплине «Системное программное обеспечение и программирование», которые позволят студентам усовершенствовать навыки разработки программ на языках программирования C/C++, а также закрепить полученные теоретические знания в области системного программного обеспечения. В конце каждой лабораторной работы приведены контрольные вопросы. В приложения включены примеры листингов программ.

Методические указания предназначены для студентов всех форм обучения специальности 5В070200 – Автоматизация и управление.

Ил. 1, табл. 11, библиогр. – 8 назв.

Рецензент: ст. преп., канд. техн. наук Мусапирова Г.Д.

Печатается по плану издания некоммерческого акционерного общества «Алматинский университет энергетики и связи» на 2014 г.

1 Лабораторная работа №1. Работа с символьной информацией

Цель: получить практические навыки в работе с массивами и указателями языка С, научиться обеспечивать функциональную модульность.

1.1 Задание к лабораторной работе

Согласно выбранному варианту (таблица 1.1), создать функцию для обработки символьных строк. За образец следует взять библиотечные функции обработки строк языка С, но применять их в своей функции не разрешается. Следует предусмотреть обработку ошибок в задании параметров и особые случаи. Требуется разработать два варианта заданной функции, используя традиционную обработку массивов и используя адресную арифметику.

Таблица 1.1 – Варианты заданий

№	Функция	Назначение
1	Copies(s,s1,n)	копирование строки s в строку s1 n раз
2	Words(s)	подсчет слов в строке s
3	Parse(s,t)	разделение строки s на две части: до первого вхождения символа t и после него
4	Center(s1,s2,l)	размещение строки s1 в середине строки s2 длиной l
5	Delete(s,n,l)	удаление из строки s подстроки, начиная с позиции n, длиной l
6	Insert(s,s1,n)	вставка в строку s подстроки s1, начиная с позиции n
7	Reverse(s)	изменение порядка символов в строке s на обратный
8	Pos(s,s1)	поиск первого вхождения подстроки s1 в строку s
9	LastPos(s,s1)	поиск последнего вхождения подстроки s1 в строку s
10	WordIndex(s,n)	определение позиции начала в строке s слова с номером n
11	WordLength(s,n)	определение длины слова с номером n в строке s
12	SubWord(s,n,m)	выделение из строки s m слов, начиная со слова n
13	WordCmp(s1,s2)	сравнение строк (с игнорированием пробелов)
14	Compul(s1,s2)	сравнение строк s1 и s2, игнорируя различия в регистрах
15	Overlay(s,s1,n)	перекрытие части строки s, начиная с позиции n, строкой s1
16	StrSet(s,n,l,t)	установка k символов строки s, начиная с позиции n, в значение t
17	Space(s,l)	доведение строки s до длины l путем равномерной вставки пробелов между словами
18	CopyStr(s1,s2)	копирование подстроки str строки s1 в строку s2, начиная с позиции n

Продолжение таблицы 1.1

19	Findwords(s,s1)	поиск вхождения в строку s заданной фразы s1
20	Replace(s,s1,s2)	замена в строке s подстроки символов s1 на s2
Примечание - под *«словом»* везде понимается последовательность символов, которая не содержит пробелов.		

1.2 Общие указания к выполнению лабораторной работы

При выполнении задания следует придерживаться следующих рекомендаций:

а) выполнить постановку задачи, с учетом выбранного варианта;

б) при работе со строкой, как с массивом, нужно иметь в виду, что длина строки заранее неизвестна, поэтому циклы целесообразно организовывать не со счетчиком, а до появления признака конца строки;

в) создаваемая функция *function* должна реализовывать только поставленную задачу — и ничего более. Тогда при ошибочном задании параметров или при каких-то особых случаях в их значениях, функция не должна аварийно завершать программу или выводить какие-то сообщения на экран, но должна возвращать какое-то прогнозируемое значение, по которому можно сделать вывод об ошибке или об особом случае;

г) определить состав параметров функции *function* и установить ее возможные возвращаемые значения;

д) интерпретировать конфигурацию параметров (ограничения, условия) и выбор реакции на неправильное задание;

е) описать логическую структуру программы;

ж) разработать два варианта заданной функции *function_mas* и *function_ptr*, используя традиционную обработку массивов и используя адресную арифметику;

и) выполнить реализацию программы и тестирование ее работы. Тестирование должно обеспечить проверку работоспособности функций для всех вариантов входных данных. Входные данные, на которых проводится тестирование, сводятся в таблицу (таблица 1.2).

Пример реализации приведен в приложении А.

Таблица 1.2 – Данные для тестирования

№ теста	Входные данные		Выходные данные	
	…	…	…	…
1				
2				
…				

1.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- листинги программ с комментариями;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

1.4 Контрольные вопросы

1.4.1 В чем заключается специфика системного программного обеспечения?

1.4.2 Какие особенности языка С позволяют использовать его в качестве инструмента для системного программирования?

1.4.3 Каким образом представляются строки символов в С?

1.4.4 Что представляют собой функции?

1.4.5 С какой целью используются прототипы функций?

1.4.6 В чем разница между локальными и глобальными переменными?

1.4.7 Что такое перегрузка функции?

1.4.8 Что представляют собой указатели?

1.4.9 В чем разница между адресом, хранимым в указателе, и значением, записанным по этому адресу?

1.4.10 В чем различие между ссылкой и указателем?

2 Лабораторная работа №2. Представление в памяти массивов и матриц

Цель: получить практические навыки в использовании указателей и динамических объектов в языке С, а также создании модульных программ и обеспечение инкапсуляции.

2.1 Задание к лабораторной работе

Сформировать разреженную матрицу целых чисел в соответствии с выбранным вариантом задания (таблица 2.1) и создать модуль доступа к ней, в котором следует обеспечить экономию памяти при размещении данных. Способ индексации выбрать самостоятельно.

Таблица 2.1 – Варианты заданий

№	Назначение
1	Все нулевые элементы размещены на главной диагонали, в первых 3 строках выше диагонали и в последних 3 строках ниже диагонали

Продолжение таблицы 2.1

2	Все нулевые элементы размещены на местах с нечетными индексами строк и столбцов
3	Все нулевые элементы размещены в правой части матрицы
4	Все нулевые элементы размещены в левой и верхней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти)
5	Все нулевые элементы размещены на местах с четными индексами строк и столбцов
6	Все нулевые элементы размещены в верхней и нижней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти)
7	Матрица поделена диагоналями на 4 треугольника, элементы верхнего и нижнего треугольников - нулевые
8	Все нулевые элементы размещены в левой части матрицы
9	Все нулевые элементы размещены в шахматном порядке, начиная со 2-го элемента 1-й строки
10	Все нулевые элементы размещены на главной диагонали и в верхней половине участка выше диагонали
11	Все нулевые элементы размещены в шахматном порядке, начиная с 1-го элемента 1-й строки
12	Все нулевые элементы размещены в столбцах, индексы которых кратны четырем
13	Все нулевые элементы размещены на главной диагонали и в нижней половине участка ниже диагонали
14	Матрица поделена диагоналями на 4 треугольника, элементы левого и правого треугольников нулевые
15	Матрица поделена диагоналями на 4 треугольника, элементы правого и нижнего треугольников нулевые
16	Все нулевые элементы размещены попарно в шахматном порядке (сначала 2 нулевых)
17	Все нулевые элементы размещены выше главной диагонали в нечетных строках и ниже главной диагонали — в четных
18	Все нулевые элементы размещены ниже главной диагонали в нечетных строках и выше главной диагонали — в четных
19	Все нулевые элементы размещены в левой и правой четвертях матрицы (главная и побочная диагонали делят матрицу на четверти)
20	Все нулевые элементы размещены квадратами 2x2 в шахматном порядке

2.2 Общие указания к выполнению лабораторной работы

При выполнении задания следует придерживаться следующих рекомендаций:

а) выполнить постановку задачи, с учетом выбранного варианта;

б) экономное использование памяти предусматривает, что для тех элементов матрицы, в которых наверняка содержатся нули, память выделяться не будет. Поскольку при этом нарушается двумерная структура матрицы, она может быть представлена в памяти как одномерный массив, но при обращении к элементам матрицы пользователь имеет возможность обращаться к элементу по двум индексам;

в) программное изделие должно быть отдельным модулем - файл *lab2.c*, в котором должны размещаться как данные (матрица и вспомогательная информация), так и функции, которые обеспечивают доступ. Внешний доступ к программам и данным модуля возможен только через вызов функций чтения и записи элементов матрицы. Доступные извне элементы программного модуля должны быть описаны в отдельном файле *lab2.h*, который может включаться в программу пользователя оператором препроцессора: *#include <lab2.h>*. Пользователю должен поставляться результат компиляции — файлы *lab2.obj* и *lab2.h*;

г) преобразование 2-компонентного адреса элемента матрицы, которую задает пользователь, в 1-компонентную должно выполняться отдельной функцией (функцией линеаризации), вызов которой возможен только из функций модуля. При этом возможны три метода преобразования адреса:

- при создании матрицы для нее создается также и дескриптор *D[N]* — отдельный массив, каждый элемент которого соответствует одной строке матрицы; дескриптор заполняется значениями, подобранными так, чтобы

$$n = D[x] + y,$$

где *x, y* — координаты пользователя (строка, столбец), *n* — линейная координата;

- линейная координата подсчитывается методом итерации как сумма полезных длин всех строк, предшествующих строке $\underline{x}$, и к ней прибавляется смещение *y*-го полезного элемента относительно начала строки;

- для преобразования подбирается единое арифметическое выражение, которое реализует функцию: *n = f(x,y).*

Первый вариант обеспечивает быстрейший доступ к элементу матрицы, ибо требует наименьших расчетов при каждом доступе, но плата за это — дополнительные затраты памяти на дескриптор. Второй вариант — наихудший по всем показателям, ибо каждый доступ требует выполнения оператора цикла, а это и медленно, и занимает память. Третий вариант может быть компромиссом, он не требует дополнительной памяти и работает быстрее, чем второй. Но выражение для линеаризации будет сложнее, чем в первом варианте, следовательно, и вычисляться будет медленнее. В программном примере, который приводится в приложении Б, полностью реализован именно третий вариант, но существенные фрагменты программного кода для реализации двух других показаны отдельно;

д) выполнить описание логической структуры программы;

е) для проверки функционирования модуля создается программный модуль, который имитирует программу пользователя.

2.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- листинги программ с комментариями;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

2.4 Контрольные вопросы

2.4.1 Что представляет собой программный модуль?
2.4.2 Какие переменные являются статическими?
2.4.3 Что представляет собой массив?
2.4.4 Как осуществляется инициализация массива?
2.4.5 Как объявить массив в динамической памяти?
2.4.6 Как удалить массив из динамической памяти?
2.4.7 Что представляет собой разреженный массив?
2.4.8 Можно ли объединять массивы?
2.4.9 Что представляет собой указатель на массив?
2.4.10 Как используются указатели на функции?

3 Лабораторная работа №3. Исследование механизмов работы с памятью

Цель: приобрести практические навыки использования в операционной системе Windows механизмов работы с памятью.

3.1 Задания к лабораторной работе

3.1.1 Создать две самостоятельные программы (на языке C++) для расчета значений функций *y=sinx* и *y=cosx* в интервале от 0 до 3,14. Создать третью программу, которая бы вызывала одну из программ.

3.1.2 Создать одномерный динамический массив, ввести элементы массива и отсортировать его.

3.1.3 Вычислить суммы элементов каждого столбца матрицы A(5,7) и результат записать в одномерный массив. Для этого создать двумерный динамический массив, задать элементы массива с помощью генератора случайных чисел, создать одномерный динамический массив.

3.1.4 Получить сведения о ресурсах памяти и ее текущем состоянии с помощью функции

GlobalMemorystatus (var meminfo: TmemoryStatus),

где *meminfo* – переменная структурного типа.

Поля структуры и их назначение приведены в таблице 3.1. На форму следует поместить компонент memo и вывести показатели с указанием имени

ресурса. Определить указанные объемы в каждый момент времени. Построить график изменения памяти.

Таблица 3.1 – Поля структуры

Поле	Назначение
dwLength	Размер структуры. Это поле должно быть заполнено до вызова функции *GlobalMemorystatus*
dwMemoryLoad	Процент занятой в данный момент памяти (от 0 до 100)
dwTotalPhys	Объем физической памяти в байтах
dwAvialPhys	Свободный в настоящее время объем физической памяти в байтах
dwTotalPageFile	Объем файла подкачки в байтах
dwAvialPageFile	Свободный в настоящее время объем файла подкачки
dwTotalVirtual	Объем текущего адресного пространства в байтах
dwAvialVirtual	Свободный объем текущего адресного пространства в байтах

3.2 Общие указания к выполнению лабораторной работы

Операционная система Windows может реализовать следующие механизмы работы с памятью:

а) *виртуальную память* для работы с большими массивами объектов или структур;

б) *проецируемые файлы* для операций с большими потоками данных (обычно из файлов) и для совместного использования данных несколькими процессами на одном компьютере;

в) *кучи* – для работы с множеством малых объектов.

Функции, работающие с *виртуальной памятью*, позволяют напрямую резервировать регион адресного пространства, передавать ему физическую память из страничного файла и присваивать любые допустимые атрибуты защиты.

Как и виртуальная память, *проецируемые файлы* позволяют резервировать регион адресного пространства и передавать ему физическую память. Разница между ними заключается в том, что в последнем случае физическая память не выделяется из страничного файла, а берется из файла, уже находящегося на диске. Как только файл спроецирован в память, к нему можно обращаться так, будто он в нее загружен. Проецируемые файлы применяются в следующих случаях:

- для загрузки и выполнения *exe*- и *dll*-файлов;
- для получения доступа к файлу данных, размещенных на диске;
- для разделения данных между несколькими процессами, выполняемыми на одной машине.

В процессе загрузки и выполнения *exe*–файла используется функция *CreateProcess* и выполняются следующие действия:

- отыскивается *exe*-файл, указанный в вызове функции, если файл не найден, новый процесс не создается, а функция возвращает значение *false*;
- если файл найден, создается новый объект ядра «процесс»;
- создается адресное пространство процесса;
- резервируется такая область адресного пространства, чтобы в нее поместился данный *exe*-файл. Желательное расположение этого региона указывается внутри самого *exe*-файла. По умолчанию базовый адрес *exe*-файла 0x00400000;
- система отмечает, что физическая память, связанная с зарезервированным регионом – *exe*-файл на диске, а не страничный файл.

Использование виртуальной памяти не всегда удобно. Например, связанные списки и деревья проще обрабатывать, используя *кучи*. Преимущество динамически распределяемой памяти заключается в том, что она позволяет игнорировать гранулярность выделения памяти и размер страниц. Недостаток этого механизма управления памятью заключается в более медленном выделении и освобождении блоков памяти.

Куча – это область зарезервированного адресного пространства. Первоначально большей его части физическая память не передается. По мере того, как программа занимает эту область под данные, специальный диспетчер, управляющий кучами (*heap manager*), постранично передает ей физическую память из страничного файла. При освобождении блоков в куче диспетчер возвращает системе соответствующие страницы физической памяти.

Для C++-программ существуют два основных способа хранения информации в основной памяти:

- использование *переменных*. Область памяти, предоставляемая переменным, закрепляется за ними во время компиляции и не может быть изменена при выполнении программы;
- использование *системы динамического распределения памяти*. В этом случае память для данных выделяется по мере необходимости из раздела свободной памяти, которая расположена между нашей программой (и ее постоянной областью хранения) и стеком. Этот раздел называется *кучей*.

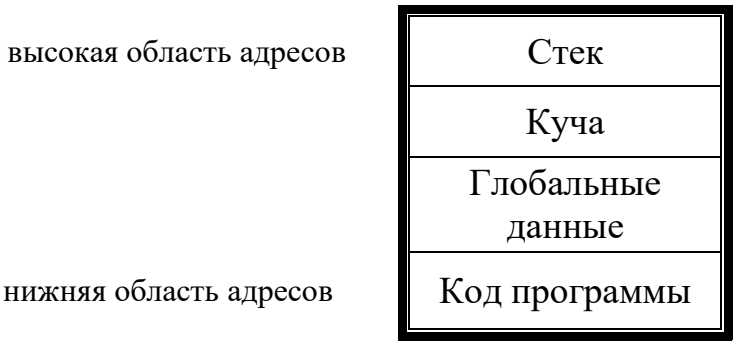

Рисунок 3.1 – Размещение разделов в областях памяти

Система динамического распределения памяти – это средство получения программой некоторой области памяти во время ее выполнения.

Динамическое выделение памяти – это получение программой памяти во время ее выполнения, то есть можно создавать переменные во время ее выполнения и в нужном количестве. Эта система динамического распределения важна для таких структур, как списки, деревья, которые изменяют свой размер по мере использования. Чтобы удовлетворить запрос на динамическое выделение памяти, используется куча.

3.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- листинги программ с комментариями;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

3.4 Контрольные вопросы

3.4.1 Какие механизмы работы с памятью реализует ОС Windows?

3.4.2 С какой целью используется виртуальная память?

3.4.3 В чем особенность использования проецируемых файлов?

3.4.4 Какая функция используется в процессе загрузки и выполнения *exe*–файла?

3.4.5 Какие действия выполняются в результате использования функции *CreateProcess*?

3.4.6 Что представляет собой куча?

3.4.7 Какие способы хранения информации в основной памяти существуют в C++?

3.4.8 Какие операторы C++ служат для выделения и освобождения памяти? Приведите примеры использования.

3.4.9 Как выделяется память под динамический массив? Приведите примеры.

3.4.10 Какую информацию можно получить с помощью функции *GlobalMemorystatus*?

4 Лабораторная работа №4. Распределение ресурсов вычислительной системы между процессами

Цель: получить практические навыки распределения единичных и множественных ресурсов вычислительной системы между процессами, а также построения графов.

4.1 Задания к лабораторной работе

4.1.1 В системе имеются *n* процессов и *m* ресурсов, которые можно предоставить этим процессам. Текущее распределение ресурсов и их максимальное количество, необходимое процессам, принять согласно варианту задания (таблица 4.1).

1) Определить все неизвестные векторы ресурсов, а именно:

а) для четных вариантов - векторы существующих *E* и предоставленных *C* ресурсов, а также матрицу требуемых ресурсов *R*, используя известный вектор доступных ресурсов *A*;

б) для нечетных вариантов - векторы предоставленных *C* и доступных *A* ресурсов, а также матрицу требуемых ресурсов *R*, используя известный вектор существующих ресурсов *E*.

2) Заполнить приведенную в варианте таблицу вручную. Определить оптимальный вариант распределения существующих ресурсов. Построить граф состояния системы.

3) Используя возможности виртуального лабораторного комплекса, а именно: разделы «Построение графа состояния системы», «Распределение разделяемых множественных ресурсов» и «Руководство пользователя» выполнить задание пункта 2.

4) Сравнить полученные результаты. Сделать выводы о состоянии системы и возможных маршрутах ее обслуживания.

Таблица 4.1 – Варианты к заданию 4.1.1

№	Распределение ресурсов и их количество
1	

Процесс	Предоставлено p1, p2, p3, p4	Максимальные требования	Требуется p1, p2, p3, p4	Доступно p1, p2, p3, p4
A	0 3 1 2	2 3 1 7		0 1 0 1
B	2 0 1 4	2 2 6 4		
C	1 2 3 0	2 2 3 1		
D	2 3 2 1	4 3 2 2		
E	0 3 3 2	0 4 3 3		

2	

Процесс	Предоставлено p1, p2, p3, p4	Максимальные требования	Требуется p1, p2, p3, p4	Доступно p1, p2, p3, p4
A	1 1 3 2	0 0 0 0		1 2 3 0
B	2 1 0 3	2 2 1 4		
C	1 0 0 1	2 2 2 1		
D	0 0 0 1	2 3 2 1		
E	1 3 4 0	3 3 4 2		
F	2 3 1 1	4 4 1 2		

Продолжение таблицы 4.1

3

Процесс	Предоставлено p1, p2, p3, p4	Максимальные требования	Требуется p1, p2, p3, p4	Доступно p1, p2, p3, p4
A	0 0 0 3	2 3 4 3		0 0 1 1
B	2 3 0 0	2 3 1 1		
C	1 2 0 1	2 2 3 1		
D	2 4 1 0	4 4 2 2		
E	1 1 1 1	2 2 3 1		

4

Матрица распределенных ресурсов

процесс	ресурс 1	ресурс 2	ресурс 3	ресурс 4
A	1	1	0	1
B	2	0	2	1
C	3	0	2	1
D	1	4	3	1
F	2	3	0	0

Матрица максимальных требований

процесс	ресурс 1	ресурс 2	ресурс 3	ресурс 4
A	2	2	1	1
B	4	4	2	2
C	3	3	2	2
D	2	4	4	3
F	3	4	4	4

5

Матрица распределенных ресурсов

процесс	ресурс 1	ресурс 2	ресурс 3	ресурс 4
A	3	1	1	0
B	2	4	3	3
C	1	1	0	0
D	0	0	2	1
F	2	1	1	1

Матрица максимальных требований

процесс	ресурс 1	ресурс 2	ресурс 3	ресурс 4
A	4	2	2	2
B	4	4	4	4
C	2	2	2	3
D	3	3	2	1
F	3	2	2	3

4.1.2 В системе существует только один ресурс каждого типа. Система состоит из *n* процессов и *m* ресурсов. В некоторый момент времени система соответствует списку, указанному в варианте задания (таблица 4.2).

1) Определить, возможно ли немедленное удовлетворение всех запросов?

Таблица 4.2 – Варианты к заданию 4.1.2

№	Состояние системы в некоторый момент времени
1	**Удерживает** / **Ожидает** Процесс P_1 — - / ресурс R_1 Процесс P_2 — ресурс R_1 / ресурс R_2 Процесс P_3 — ресурс R_2 / ресурс R_3 Процесс P_4 — - / ресурс R_2 Процесс P_5 — ресурс R_3 / ресурс R_1 Процесс P_6 — - / ресурс R_1
2	**Удерживает** / **Ожидает** Процесс P_1 — ресурс R_1 / ресурс R_3 Процесс P_2 — ресурс R_2 / ресурс R_4 Процесс P_3 — ресурс R_4 / ресурс R_3 Процесс P_4 — - / ресурс R_2 Процесс P_5 — ресурс R_3 / ресурс R_4 Процесс P_6 — - / ресурс R_2 Процесс P_7 — - / ресурс R_1
3	**Удерживает** / **Ожидает** Процесс P_1 — ресурс R_3 / ресурс R_1 Процесс P_2 — ресурс R_2 / ресурс R_4 Процесс P_3 — - / ресурс R_3 Процесс P_4 — - / ресурс R_2 Процесс P_5 — - / ресурс R_4 Процесс P_6 — ресурс R_1 / ресурс R_2 Процесс P_7 — ресурс R_4 / ресурс R_1
4	а) процесс A занимает ресурс R и хочет получить ресурсы S, T; б) процесс B ничего не использует, но хочет получить ресурс W; в) процесс C ничего не использует, но хочет получить ресурс V; г) процесс D занимает ресурс V и хочет получить ресурсы S и T; д) процесс E занимает ресурс T и хочет получить ресурсы R, V; е) процесс F занимает ресурс W и хочет получить ресурс S;
5	а) процесс A занимает ресурс R и хочет получить ресурс S; б) процесс B ничего не использует, но хочет получить ресурс V; в) процесс C ничего не использует, но хочет получить ресурс S; г) процесс D занимает ресурс U и хочет получить ресурсы W и T; д) процесс E занимает ресурс S и хочет получить ресурс V; е) процесс F занимает ресурс W и хочет получить ресурс R; ж) процесс G занимает ресурс V и хочет получить ресурс U.

2) В какой последовательности следует удовлетворять запросы процессов для завершения их выполнения при условии одновременного освобождения ресурсов? Выполнить необходимые расчеты вручную. Построить граф состояния системы.

3) С помощью виртуального комплекса, используя разделы «Построение графа состояния системы», «Распределение неразделяемых ресурсов» и руководство пользователя, выполнить задание пункта 2 в двух режимах – при условии одновременного освобождения ресурсов и произвольного освобождения ресурсов.

4) Сравнить полученные результаты. Сделать выводы о состоянии системы и возможных вариантах ее обслуживания.

4.2 Общие указания к выполнению лабораторной работы

Компонент «Построение графа состояния системы» виртуального лабораторного комплекса предназначен для обнаружения тупиковой ситуации методом построения графа состояния системы. Приступая к работе с данным программным продуктом необходимо выполнить следующие действия:

1) Запустить приложение *Kompleks.exe* и в открывшемся окне выбрать нужный компонент.

2) Открывшееся окно функционально можно разделить на три части:
   - область ввода информации для формирования пустых таблиц;
   - область таблиц для задания текущего состояния системы;
   - область построения графа.

3) Установить согласно заданию количество активных процессов и доступных ресурсов системы, при этом будут сформированы две пустые таблицы – таблица «Удерживаемых ресурсов» и «Запрашиваемых ресурсов».

4) В сформированные таблицы необходимо ввести данные о запрашиваемых и ожидаемых ресурсах.

5) После нажатия кнопки «Построить граф», она деактивируется до введения данных о новом состоянии системы, а в рабочем окне появится изображение. В зависимости от состояния системы вершины окрашиваются в красный или зеленый цвет. Кроме того, возможно пересечение линий, поэтому для улучшения восприятия изображение можно отредактировать – перемещать вершины и определять длину стрелок.

6) Для построения нового графа необходимо заново ввести количество активных процессов и доступных ресурсов, после чего нажать кнопку «Установить» и «Построить граф».

В программе предусмотрены исключительные ситуации ввода пользователем неверной информации или отсутствия введенной информации. В случае возникновения пользователю будет выведено сообщение об ошибке.

Для выхода из программы необходимо нажать кнопку «Выход».

4.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;

- снимки экрана с результатами работы;
- выводы по каждому заданию.

4.4 Контрольные вопросы

4.4.1 Что понимают под термином «ресурс»?
4.4.2 Какие способы разделения ресурсов существуют?
4.4.3 Какие ресурсы можно отнести к важнейшим?
4.4.4 Какие типы программных модулей различают? Возможно ли их разделение?
4.4.5 Какие устройства можно отнести к устройствам с прямым или последовательным доступом?
4.4.6 Что относится к информационным ресурсам?
4.4.7 Какова общая схема выделения ресурсов?
4.4.8 При каких условиях процессу может быть выделен ресурс?
4.4.9 В каких случаях ресурс принудительно отбирается у процесса?
4.4.10 Как приоритет процесса может влиять на выделение ресурса?

5 Лабораторная работа №5. Взаимные блокировки потоков и их обнаружение

Цель: получить практические навыки обнаружения взаимоблокировок потоков и освоить способы их устранения.

5.1 Задания к лабораторной работе

5.1.1 Обнаружение взаимоблокировок при наличии неразделяемых ресурсов. Рассматривается простой вариант для случая, когда в системе существует только один ресурс каждого типа.

Пусть система состоит из n процессов и m ресурсов. В некоторый момент времени система соответствует определенному списку (таблица 5.1). Требуется:

а) построить граф ресурсов и процессов, позволяющий установить процессы, которые попали в тупиковую ситуацию;

б) определить заблокирована ли эта система и если да, то какие процессы в этом участвуют;

в) распределить ресурсы по процессам в соответствии с приведенным списком. В случае тупиковой ситуации, определить какой из процессов следует снять и выполнить оптимальное распределение ресурсов;

г) вручную и с помощью виртуального комплекса, используя компоненты «Построение графа состояния системы», «Распределение разделяемых множественных ресурсов» и руководство пользователя, выполнить задание пунктов а и в;

д) сравнить полученные результаты и сделать выводы.

Таблица 5.1 – Варианты к заданию 5.1.1

№	Текущее состояние системы		
1		*Удерживает*	*Ожидает*
	Процесс P_1	-	ресурс R_1
	Процесс P_2	ресурс R_1	ресурс R_2
	Процесс P_3	ресурс R_2	ресурс R_3
	Процесс P_4	-	ресурс R_2
	Процесс P_5	ресурс R_3	ресурс R_1
	Процесс P_6	-	ресурс R_1
2		*Удерживает*	*Ожидает*
	Процесс P_1	ресурс R_1	ресурс R_3
	Процесс P_2	ресурс R_2	ресурс R_4
	Процесс P_3	ресурс R_4	ресурс R_3
	Процесс P_4	-	ресурс R_2
	Процесс P_5	ресурс R_3	ресурс R_4
	Процесс P_6	-	ресурс R_2
	Процесс P_7	-	ресурс R_1
3		*Удерживает*	*Ожидает*
	Процесс P_1	ресурс R_3	ресурс R_1
	Процесс P_2	ресурс R_2	ресурс R_4
	Процесс P_3	-	ресурс R_3
	Процесс P_4	-	ресурс R_2
	Процесс P_5	-	ресурс R_4
	Процесс P_6	ресурс R_1	ресурс R_2
	Процесс P_7	ресурс R_4	ресурс R_1
4		*Удерживает*	*Ожидает*
	Процесс P_1	ресурс R_1	ресурс R_3
	Процесс P_2	-	ресурс R_1
	Процесс P_3	ресурс R_2	ресурс R_3
	Процесс P_4	-	ресурс R_2
	Процесс P_5	ресурс R_3	ресурс R_1
5		*Удерживает*	*Ожидает*
	Процесс P_1	ресурс R_2	ресурс R_1
	Процесс P_2	ресурс R_3	ресурс R_4
	Процесс P_3	ресурс R_1	ресурс R_3
	Процесс P_4	-	ресурс R_2
	Процесс P_5	-	ресурс R_2
	Процесс P_6	ресурс R_4	ресурс R_1

5.1.2 Решить задачи, приведенные ниже, вручную и с помощью виртуального комплекса, используя компоненты «Построение графа состояния системы», «Распределение разделяемых множественных ресурсов» и «Руководство пользователя» (таблица 5.2):

а) *для нечетных вариантов*. В системе имеются *n* процессов и *m* ресурсов, которые можно предоставить этим процессам. Текущее распределение ресурсов и максимальное их количество, необходимое процессам, приведено в таблице 5.2. Определить оптимальный вариант распределения существующих ресурсов.

б) *для четных вариантов.* Имеются *n* процессов и *m* ресурсов, также известны матрица распределенных ресурсов и матрица требований. Вектор существующих ресурсов E. Определить оптимальный вариант распределения существующих ресурсов. Возможно ли возникновение в системе тупиковой ситуации?

Таблица 5.2 – Варианты к заданию 5.1.2

№ 1

Процесс	Предоставлено p1, p2, p3, p4	Максимальные требования	Требуется p1,p2,p3,p4	Доступно после завершения p1, p2, p3, p4
A	0 1 1 0	0 2 1 2		2 1 1 1
B	3 0 0 1	3 3 1 1		
C	2 0 3 4	2 3 5 4		
D	2 3 1 3	3 3 2 3		
E	0 2 3 1	1 2 4 3		

№ 2

Матрица распределенных ресурсов

процесс	ресурс 1	ресурс 2	ресурс 3	Вектор существующих ресурсов E
A	1	1	0	9 5 7
B	3	1	3	
C	4	1	2	
D	1	1	2	

Матрица максимальных требований

процесс	ресурс 1	ресурс 2	ресурс 3
A	3	2	2
B	3	3	3
C	4	1	4
D	4	2	2

№ 3

Процесс	Предоставлено p1, p2, p3, p4	Максимальные требования	Требуется p1,p2,p3,p4	Доступно после завершения p1, p2, p3, p4
A	2 1 1 0	2 2 1 2		1 1 2 2
B	4 1 1 0	4 3 1 0		
C	2 1 1 2	2 3 5 4		
D	3 3 1 3	3 3 2 3		
E	1 2 0 1	1 2 4 3		

Продолжение таблицы 5.2

4

Матрица распределенных ресурсов

процесс	ресурс 1	ресурс 2	ресурс 3	Вектор существующих ресурсов E
A	2	1	1	10 4 6
B	3	0	1	
C	2	0	1	
D	2	1	2	

Матрица максимальных требований

процесс	ресурс 1	ресурс 2	ресурс 3
A	3	3	4
B	4	3	3
C	3	2	4
D	4	4	4

5

Процесс	Предоставлено p1, p2, p3, p4	Максимальные требования	Требуется p1,p2,p3,p4	Доступно после завершения p1, p2, p3, p4
A	1 1 1 1	2 1 1 2		1 1 0 1
B	3 1 1 0	3 1 2 1		
C	1 1 3 4	2 2 4 4		
D	1 1 1 3	2 1 2 3		
E	1 2 0 1	3 2 2 1		

6

Матрица распределенных ресурсов

процесс	ресурс 1	ресурс 2	ресурс 3	Вектор существующих ресурсов E
A	1	2	2	7 7 8
B	1	1	2	
C	2	0	3	
D	2	1	1	

Матрица максимальных требований

процесс	ресурс 1	ресурс 2	ресурс 3
A	2	3	2
B	3	2	2
C	4	1	4
D	3	1	4

5.2 Общие указания к выполнению лабораторной работы

При наличии нескольких разделяемых ресурсов каждого типа возможно использование алгоритма обнаружения взаимоблокировок.

Пусть имеется множество процессов $P=\{P_1, P_2, \ldots, P_n\}$ и множество ресурсов $E=\{E_1, E_2, \ldots, E_m\}$, где n и m - количество процессов и ресурсов

соответственно. В любой момент времени некоторые ресурсы могут быть заняты и, соответственно, недоступны.

Пусть вектор A=(A_1, A_2, …, A_m) – вектор доступных ресурсов. Причем выполняется соотношение A_j<=E_j, где j=1, 2, …, m.

Кроме того, рассматриваются две матрицы:

1) C={c_{ij}}, *i*=1, 2, …n; *j*=1,2,...,m – матрица текущего распределения ресурсов, где c_{ij} – количество ресурсов j-того класса, которые занимает процесс P_i.

2) R={r_{ij}}, *i*=1,2,...n; *j*=1,2,...,m – матрица требуемых (запрашиваемых) ресурсов, где r_{ij} – количество ресурсов j–того класса, которые хочет получить процесс P_i.

Справедливо m соотношений по ресурсам:

$$\sum_{i=1}^{n} c_{ij} + A_j = E_j \text{, где } j=1,2,...,m. \quad (1)$$

Алгоритм обнаружения взаимоблокировок основан на сравнении векторов доступных и требуемых ресурсов. В исходном состоянии все процессы не отмечены. По мере реализации алгоритма на процессы будет ставиться отметка, что они могут закончить свою работу, следовательно, не находятся в тупике. После завершения алгоритма любой немаркированный процесс находится в тупиковой ситуации.

Алгоритм обнаружения тупиков состоит из следующих шагов:

1) Ищется процесс P_i, для которого *i*-строка матрицы R меньше вектора A, то есть R_i<=A, или r_{ij}<A_j, где *j*=1, 2, ..., m.

2) Если такой процесс найден, это значит, что он может завершиться и освободить занятые ресурсы. Найденный процесс отмечается, *i*–я строка матрицы C прибавляется к вектору A, то есть A_j=A_j+c_{ij}, где *j*=1,2,...m, и осуществляется возвращение к шагу 1.

3) Если таких процессов не существует, работа алгоритма заканчивается, а неотмеченные процессы попадают в тупик.

Изложенный алгоритм можно использовать для решения задач обнаружения тупиковых ситуаций и заблокированных процессов в системе с множественными ресурсами.

5.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

5.4 Контрольные вопросы

5.4.1 Что такое «поток»?

5.4.2 В чем разница между потоком и процессом?
5.4.3 В каких случаях возникает взаимоблокировка?
5.4.4 На чем основан алгоритм обнаружения взаимоблокировок?
5.4.5 Какие виды планировщиков Вам известны?
5.4.6 Какие состояния возможны для активных процессов?
5.4.7 Приведите пример пассивных процессов?
5.4.8 Что понимается под мультипрограммированием?
5.4.9 Что понимается под планированием вычислительных процессов?
5.4.10 Что определяет стратегия планирования?

6 Лабораторная работа №6. Распределение процессорного времени в операционных системах

Цель: ознакомиться с основными дисциплинами диспетчеризации и получить практические навыки их использования при распределении процессорного времени.

6.1 Задания к лабораторной работе

Запустить виртуальный комплекс, компонент «Эмуляции работы дисциплин диспетчеризации».

6.1.1 Провести для каждой из четырех основных дисциплин диспетчеризации серии экспериментов по определению оптимальных параметров (частоты выполнения заблокированного процесса для FCFS, оптимального значения кванта времени для RR), изменяя характеристики в окне настроек. Полученные данные экспортировать в Excel, используя соответствующую команду в меню Файл. Построить графики отношения заблокированных и выполненных задач для каждой серии.

6.1.2 При одинаковых параметрах (использовать оптимальные, подобранные в пункте 6.1.1 для каждой из дисциплин диспетчеризации), определить наиболее эффективную дисциплину диспетчеризации для загруженных процессов. Обосновать свой выбор.

6.1.3 Провести серии экспериментов 6.1.1, 6.1.2 для другого набора процессов (можно использовать домашний компьютер). Сравнить результаты.

6.2 Общие указания к выполнению лабораторной работы

Компонент «Эмуляции работы дисциплин диспетчеризации» виртуальной системы предназначен для исследования принципов работы четырех дисциплин диспетчеризации: FCFS, SJN, SRT, RR. Для эмуляции используются копии реальных процессов компьютера, на которой установлена настоящая программа. Приступая к работе, необходимо выполнить следующие действия:

а) запустить приложение *dispatcher.exe*;

б) в главном меню во вкладке «Файл» нажать кнопку «Загрузить» после чего сгенерируется список процессов и их параметров, с помощью которых будет производиться эмуляция;

в) во вкладке «Дисциплина» необходимо выбрать алгоритм дисциплины диспетчеризации, с помощью которой будет производиться эмуляция;

г) нажать клавишу «Запустить» во вкладке «Файл».

После выполнения вышеперечисленных действий запустится процесс эмуляции, а в рабочем окне отобразится информация о ходе выполнения эмуляции. Функционально окно можно разделить на четыре части:

1) Область отображения списка используемых процессов и их параметров.

2) Область построения зависимостей выполнения процессов.

3) Область отображения эмуляции процессора и очередей выполняемых процессов.

4) Графическое отображение маршрутов процессов в ходе эмуляции.

Для смены дисциплины диспетчеризации необходимо нажать кнопку «Остановить» во вкладке «Файл» для остановки процесса эмуляции, а затем - кнопку «Сброс» во вкладке «Файл». После чего выполнить действия для запуска выполнения программы (кроме пункта а). Для изменения параметров эмуляции необходимо нажать кнопку «Параметры» в главном меню. Изменяемые параметры:

– максимальное время выполнения процесса (для всех дисциплин);

– частота выполнения заблокированного процесса (для дисциплины FCFS);

– квант времени (для дисциплины RR).

Для импорта результатов в Excel с целью анализа проведенных экспериментов необходимо выполнить следующие действия:

1) Провести по одному эксперименту для каждой дисциплины диспетчеризации. В каждом эксперименте количество итераций не должно превышать 40.

2) Нажать кнопку «Загрузить в Excel» во вкладке «Файл».

3) Открыть «файл DDbook.xls» в папке, где установлена программа.

Для выхода из программы необходимо остановить процесс эмуляции (кнопка «Остановить») и нажать кнопку «Выход» во вкладке «Файл».

6.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

6.4 Контрольные вопросы

6.4.1 С какой целью в операционных системах используются дисциплины диспетчеризации?

6.4.2 В каких операционных системах используется дисциплина FCFS?

6.4.3 Какими достоинствами обладают рассмотренные основные дисциплины обслуживания?

6.4.4 Какие недостатки невытесняющих дисциплин устранены с помощью дисциплины RR?

6.4.5 Какое влияние на результат использования RR оказывает учет приоритетов?

6.4.6 Что представляет собой частота выполнения заблокированного процесса?

6.4.7 Приведите пример использования дисциплин диспетчеризации в ОС реального времени.

6.4.8 Приведите схемы функционирования дисциплин SJN и SRT.

6.4.9 Как изменится схема функционирования FCFS при использовании одной очереди вместо двух?

6.4.10 Какие стратегии обслуживания используются в Unix-системах?

7 Лабораторная работа №7. Синхронизация потоков

Цель: ознакомиться с основными методами синхронизации потоков в многозадачных системах.

7.1 Задания к лабораторной работе

Самостоятельно согласно выбранному варианту (таблица 7.1) создать 2 потока и выполнить их синхронизацию.

Таблица 7.1 – Варианты к заданию

№	Поток 1	Поток 2
1	В бесконечном цикле рисует спираль	Выводит текст из файла
2	Рассчитывает значение π/2	Рисует окружности заданных радиусов (в цикле, с определенным шагом)
3	Печатает алфавит. Цикл бесконечный	Рисует график показательной функции
4	В бесконечном цикле рисует окружность	Рассчитывает значение функции y=sin(x)
5	Рассчитывает значение функции y=cos(x)	Печатает приветствие. Цикл бесконечный.

7.2 Общие указания к выполнению лабораторной работы

С целью ознакомления использовать виртуальный лабораторный комплекс, компонент «Потоки и синхронизация». Для запуска потока расчета числа π установить галочку в графу «Рассчитать». Для сравнения представлено точное значение числа π. Также предоставлено значение числа проведенных итераций расчета. Чем выше число итераций – тем точнее значение рассчитанной величины π. Остановить поток выполнения расчета числа π можно, убрав галочку из графы «Рассчитать».

Для запуска потока вывода введенного пользователем текста необходимо предварительно ввести текст. Для этого необходимо во вкладке «Файл» главного меню нажать кнопку «Изменить текст». Откроется файл Symbols.txt. Ввести текст, сохранить изменения.

Нажать кнопку «Запустить поток». Число нажатий данной кнопки определяет количество создаваемых потоков.

Для запуска потоков заполнения ProgressBar необходимо предварительно остановить выполнение потоков расчета числа π и вывода текста – нажав кнопку «Рассчитать» и дождавшись завершения вывода текста соответственно. Нажать кнопку «Пуск».

Изменить приоритет выполняемых потоков заполнения ProgressBar. Для этого установить указатель приоритета соответствующего потока на необходимое деление. Указатели приоритета приведены в окне «Приоритет».

Остановить выполнение того или иного потока заполнения ProgressBar установив галочку в соответствующую номеру потока графу «Блокировать поток».

7.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- листинги программ с комментариями;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

7.4 Контрольные вопросы

7.4.1 Что представляет собой «поток»?

7.4.2 Поясните разницу между потоком выполнения и процессом.

7.4.3 В чем состоит суть мультипрограммирования?

7.4.4 В чем заключается основное отличие мультипрограммирования от мультизадачности?

7.4.5 С какой целью в операционных системах используется синхронизация потоков?

7.4.6 Какие механизмы синхронизации Вам известны?

7.4.7 Какие процессы называются кооперативными?

7.4.8 Что представляет собой семафор?

7.4.9 Какие программные средства используются при организации параллельной работы потоков?

7.4.10 Что происходит при блокировании потока?

8 Лабораторная работа №8. Операционная система Windows. Основы разработки командных (пакетных) файлов

Цель: изучить использование основных команд, применяемых при разработке командных файлов; получить практические навыки разработки командных файлов.

8.1 Задания к лабораторной работе

Используя правила создания командных файлов, разработать командный файл с учетом условий, предложенных в выбранном варианте (таблица 8.1). Указать параметры запуска командного файла на выполнение.

Таблица 8.1 – Варианты задания

№	Задание
1	Используя формальные параметры, создать командный файл, который позволяет выполнить следующие функции: - найти файл, указанный пользователем; - в зависимости от результата: а) если файл найден, вывести его содержимое на экран; б) в противном случае – выдать соответствующее сообщение пользователю и завершить работу.
2	Используя переменные окружения, создать командный файл, который позволяет выполнить следующие функции: - найти файл, указанный пользователем; - в зависимости от результата: а) если файл найден, уничтожить его; б) в противном случае – выдать соответствующее сообщение пользователю и завершить работу.
3	Используя формальные параметры, создать командный файл, который позволяет выполнить следующие функции: - найти файл, по указанному контексту; - в зависимости от результата: а) если файл найден, вывести информацию о нем на экран; б) в противном случае – выдать соответствующее сообщение пользователю и завершить работу.

Продолжение таблицы 8.1

4	Используя переменные окружения, создать командный файл, который позволяет выполнить следующие функции: - создать каталог с именем, указанным пользователем; - используя цикл, создать в этом каталоге несколько текстовых файлов;
5	Используя переменные окружения, создать командный файл, который позволяет выполнить следующие функции: - найти файл, по контексту, указанному пользователем; - в зависимости от результата: а) если файл найден, изменить его атрибуты; б) в противном случае выдать соответствующее сообщение пользователю и завершить работу.
6	Используя формальные параметры, создать командный файл, который позволяет выполнить следующие функции: - найти файл, по контексту, указанному пользователем; - в зависимости от результата: а) если файл найден, скопировать его; б) в противном случае выдать соответствующее сообщение пользователю и завершить работу.
7	Используя переменные окружения, создать командный файл, который позволяет выполнить следующие функции: - запросить у пользователя и установить значение целой переменной d; - определить кратность значения переменной числу 5; - вывести результат на экран.
8	Используя переменные окружения, создать командный файл, который позволяет выполнить следующие функции: - найти файлы, по шаблону имени, указанному пользователем; - если файлы найдены, вывести информацию о файлах и установить для них атрибуты «скрытый» и «только для чтения», в противном случае выдать соответствующее сообщение пользователю и завершить работу;
9	Используя переменные окружения, создать командный файл, который позволяет запросить у пользователя и установить значение целой переменной a; определить кратность значения переменной числу 3; вывести результат на экран.
10	Используя переменные окружения, создать командный файл, который позволяет выполнить следующие функции: - проверить существование файла, указанного пользователем; - если файл не найден, создать текстовый файл с этим же именем и установить для него атрибут «только для чтения»; в противном случае выдать соответствующее сообщение пользователю и завершить работу.

8.2 Общие указания к выполнению лабораторной работы

Командный файл создается по обычным правилам, но расширением должно быть только одно сочетание - *bat* (сокращение от *batch* – пачка). Выполнение командного файла может быть прекращено командами *Ctrl+Break* или *Ctrl+C*. Командный файл выполняется командным процессором строка за строкой. Из одного командного файла можно вызвать другой командный файл командой *Call* (с возвратом) или обычной командой вызова (без возврата). Перед выполнением очередной строки командного файла ее значение выводится на экран. Вывод любой строки командного файла на экран подавляется, если строка начинается с символа @.

Для построения командных файлов используются специальные внутренние команды операционной системы (таблица 8.2), а также внешние команды, например, *find, sort, mode, more*.

Таблица 8.2 - Команды командных (пакетных) файлов

Команда	Значение
call	Вызов одного пакетного файла из другого
echo	Вывод сообщений и переключение режима отображения команд на экране
exit	Завершение программы cmd.exe (интерпретатора командных строк)
for	Запуск указанной команды для каждого из файлов в наборе
goto	Передача управления в отмеченную строку пакетного файла
if	Оператор условного выполнения команд в пакетном файле
pause	Приостановка выполнения пакетного файла и вывод сообщения
rem	Помещение комментариев в пакетные файлы
set local	Начало локальных изменений среды для пакетного файла
shift	Изменение содержимого (сдвиг) подставляемых параметров для пакетного файла

Для обращения к переменным окружения их имена следует заключать в знаки %. Формальные параметры, включаемые в строки командного файла, имеют вид %1, и так далее до %9. Фактические значения вводятся в строке вызова командного файла.

Подробную информацию о командах можно получить, набрав в командной строке *help имя_команды* или *имя_команды / ?*

8.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание способов решения задач;
- листинги программ с комментариями;

- снимки экрана с результатами работы;
- выводы по каждому заданию.

8.4 Контрольные вопросы

8.4.1 Каковы особенности использования команды echo?

8.4.2 С какой целью применяется команда set?

8.4.3 Каковы особенности использования команды find?

8.4.4 Что может использоваться в качестве условия при реализации команды if?

8.4.5 Как выполнить проверку идентичности двух символьных строк?

8.4.6 Какая команда позволяет изменить привычную последовательность команд выполнения операторов командного файла?

8.4.7 Допустимо ли совместное использование команд if и goto?

8.4.8 Какой формат команды for используется для строк командных файлов?

8.4.9 Что представляет собой переменная окружения?

8.4.10 Какое условие используется для проверки наличия файла?

Приложение А

Листинг программы и комментарии к лабораторной работе № 1

```
#include <stdio.h>
#include <conio.h>
#define N 80
/*******************************************************/
int substr_mas(char src[N], char dest[N], int num, int len)
{
int i, j;
/* проверка случая 4*/
if((num<0)||(len<=0))
{
dest[0]=0; return 0;
}
/* выход на символ num*/
for (i=0; i<=num; i++)
/* проверка случая 3*/
if(src[i]=='\0')
{
dest[0]=0; return 0;
}
/* перезапись символов*/
for (i--, j=0; j<len; j++, i++)
{
dest[j]=src[i];
/* проверка случая 2*/
if (dest[j]=='\0') return 1;
}
/* запись признака конца в выходную строку*/
dest[j]='\0';
return 1;
}
/*******************************************************/
/* функция выделения подстроки*/
/* адресная арифметика*/
/*******************************************************/
int substr_ptr(char *src, char *dest, int num, int len)
{
/* проверка случая 4*/
if((num<0)||(len<=0)) return dest[0]=0;
/* выход на символ num или на конец строки*/
while (num-- && *src++);
```

```
/* проверка случая 3*/
if(!num) return dest[0]=0;
/* перезапись символов*/
while(len-- && *src) *dest++=*src++;
/* запись признака конца в выходную строку*/
*dest=0;
return 1;
}
/*********************************************************/
main()
{
char ss[N], dd[N];
int n,l;
clrscr();
printf("Enter your symbols:\n");
gets(ss);
printf("begin=");
scanf("%d",&n);
printf("length=");
scanf("%d",&l);
printf("Arrays:\n");
if(substr_mas(ss,dd,n,l)) printf(">>%s<<\n>>%s<<\n",ss,dd);
else printf("Error! >>%s<<\n",dd);
printf("Address arithmetic:\n");
if(substr_ptr(ss,dd,n,l)) printf(">>%s<<\n>>%s<<\n",ss,dd);
else printf("Error! >>%s<<\n",dd);
getch();
}
```

Приложение Б

Листинг программы и комментарии к лабораторной работе № 2

```
/************************ файл lab2.h ***********************/
/* Описание функций и внешних переменных файла lab2.c*/
extern int L2_RESULT;
/* Глобальная переменная – флаг ошибки*/
/* Выделение памяти под матрицу*/
int creat_matr (int N);
/* Чтение элемента матрицы по заданным координатам*/
int read_matr(int x, int y);
/* Запись элемента в матрицу по заданным координатам*/
int write_matr(int x, int y, int value);
/* Уничтожение матрицы*/
int close_matr(void);
/************************ конец файла lab2.h ******************/

/************************** файл lab2.c **********************/
/* В файле определены функции и переменные для обработки матрицы,
   заполненной нулями ниже главной диагонали*/
#include <alloc.h>
static int NN;                  /* размерность матрицы*/
static int SIZE;                /* размер памяти*/
static int *m_addr=NULL;        /* адрес сжатой матрицы*/
static int lin(int,int);        /* описание функции линеаризации*/
static char ch_coord(int, int); /* описание функции проверки*/
int L2_RESULT;                  /* внешняя переменная – флаг ошибки*/
/************************************************************/
/* Выделение памяти под сжатую матрицу*/
int creat_matr(int N)
/* N – размер матрицы*/
{
NN=N;
SIZE=N*(N-1)/2+N;
if((m_addr=(int *)malloc(SIZE*sizeof(int))) == NULL)
return L2_RESULT=-1;
else
return L2_RESULT=0;
/* Возвращает 0, если выделение памяти прошло успешно, иначе -1 */
}
/************************************************************/
/* Уничтожение матрицы (освобождение памяти*/
int close_matr(void)
```

```
{ if (m_addr!=NULL)
{
free(m_addr);
m_addr=NULL;
return L2_RESULT=0;
/* Возвращает 0, если освобождение памяти прошло успешно, иначе -1*/
}
else return L2_RESULT=-1;
}
/*************************************************************/
/* Чтение элемента матрицы по заданным координатам*/
int read_matr(int x, int y)
/* x, y – координаты (строка, столбец)*/
{
if(ch_coord(x,y)) return 0;
/* Если координаты попадают в нулевой участок, возвращается 0,
иначе - применяется функция линеаризации */
return(x>y) ? 0 : m_addr[lin(x,y)];
/* Проверка успешности чтения по переменной L2_RESULT:
0 – без ошибок, -1 – ошибка  */
}
/*************************************************************/
/* Запись элемента матрицы по заданным координатам*/
int write_matr(int x,int y,int value)
/* x, y – координаты, value – записываемое значение*/
{
if(ch_coord(x,y)) return 0;
/* Если координаты попадают в нулевой участок, записи нет,
 иначе - применяется функция линеаризации*/
if(x>y) return 0;
else return m_addr[lin(x,y)]=value;
/* Проверка успешности записи по переменной L2_RESULT */
}
/*************************************************************/
/*Преобразование 2-мерных координат в линейные (вариант 3)*/
static int lin(int x,int y)
{
int n;
n=NN-x;
return SIZE-n*(n-1)/2-n+y-x;
}
/*************************************************************/
/* Проверка корректности обращения*/
static char ch_coord(int x,int y)
```

```
{
if((m_addr==NULL) || (x>SIZE) || (y>SIZE) || (x<0) || (y<0))
/* Если матрица не размещена в памяти, или заданные координаты
выходят за пределы матрицы */
return L2_RESULT=-1;
return L2_RESULT=0;
}
/********************** конец файла lab2.c *********************/

/************************** файл main2.c ***********************/
/*Программа пользователя*/
#include " lab2.h"
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
main()
{int R,i,j,m;          /*размерность, номера строки и столбца,
                       значения элемента*/
int op;                /* операция*/
clrscr();
printf("Введите размерность матрицы >");
scanf("%d",R);
/* Создание матрицы*/
if (creat_matr(R))
{ printf("Ошибка создания матрицы \n");
exit(0);}
/* Заполнение матрицы*/
for(m=j=0; j<R; j++)
for(i=0; i<R; i++)
write_matr(i,j,++m);
while(1)
/* Вывод матрицы на экран*/
{ clrscr();
for(j=0; j<R; j++)
{ for(i=0; i<R; i++)
printf("%3d",read_matr(i,j));
printf("\n");}
printf("0 – выход \n1 – чтение \n2 - запись\n>");
scanf("%d",&op);
switch (op)
{ case 0:
if(close_matr()) printf("Ошибка при уничтожении\n");
else printf("Матрица уничтожена \n");
exit (0);
```

```
case 1: case 2:
printf("Введите номер строки>");
scanf("%d",&j);
printf("Введите номер столбца>");
scanf("%d",&i);
if (op==2)
{printf("Введите значение элемента>");
scanf("%d",&m);
write_matr(j,i,m);
if(L2_RESULT<0) printf("Ошибка записи\n");}
else
{ m=read_matr(j,i);
if(L2_RESULT<0) printf("Ошибка записи\n");
else printf("Считано: %d\n",m);}
printf("Нажмите клавишу\n");
getch();
break;
}
}
}
/********************** конец файла main2.c ********************/
```

Вариант, который обеспечивает быстрейший доступ к элементу матрицы, однако требующий дополнительных затрат памяти на дескриптор, реализуется при следующих условиях:

- к общим статическим переменным добавляется переменная

```
static int *D;   /* адрес дескриптора*/
```

- в функцию *create_matr* добавляется блок

```
{int i, s;
D=(int *)malloc(N*sizeof(int));
for (D[0]=0, s=NN-1, i=1; i<NN; i++)
D[i]=D[i-1]+s--;}
```

- функция *lin* изменяется на функцию вида

```
static int lin(int x, int y)
{return D[x]+y;}
```

Наихудший по всем показателям вариант, при котором каждый доступ требует выполнения оператора цикла, что значительно замедляет программу и занимает больше памяти, реализуется при изменении функции *lin* на функцию вида

```
static int lin(int x, int y)
{int s;
for (s=j=0; j<x; j++)
s+=NN-j;
return s+y-x;}
```

Список литературы

1 Молчанов А.Ю. Системное программное обеспечение. Лабораторный практикум. - СПб., 2011.

2 Гордеев В.А. Операционные системы. – СПб.: Питер, 2011.

3 Фельдман С.К. Системное программирование на персональном компьютере. - М.: ЗАО «Новый издательский мир», 2007.

4 Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. – СПб.: Питер, 2008.

5 Финогенов К.Г. Самоучитель по системным функциям MS-DOS. - М.: М.: Горячая линия - Телеком, 2001.

6 Страуструп Б. Язык программирования C++. – М., 2012.

7 Павловская Т.А. C/C++. Структурное программирование. – СПб., 2010.

8 Немцова Т.И. Программирование на языке высокого уровня. Программирование на языке C++. - М.: «Форум», 2012.

Содержание

Сводный план 2014г., поз. 119

Наталья Валерьевна Сябина

СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ПРОГРАММИРОВАНИЕ

Методические указания по выполнению лабораторных работ
для студентов специальности 5В070200 – Автоматизация и управление

Редактор Л.Т.Сластихина
Специалист по стандартизации Н.К.Молдабекова

Подписано в печать __.__.__.
Тираж 100 экз.
Объем 2,2 уч.-изд. л.

Формат 60х84 1/16
Бумага типографская №1
Заказ _____. Цена 1100 тг.

Копировально-множительное бюро
некоммерческого акционерного общества
«Алматинский университет энергетики и связи»
050013 Алматы, Байтурсынова, 126