МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЁВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»

ВОПРОСЫ, ЗАДАНИЯ И УПРАЖНЕНИЯ ПО КУРСУ «ОПЕРАЦИОННЫЕ СИСТЕМЫ»

Утверждено Редакционно-издательским советом университета в качестве лабораторного практикума

С А М А Р А
Издательство СГАУ
2012

УДК 004.451(075)
ББК 22.161

Составитель ***С.В. Востокин***

Рецензент д-р техн. наук, проф. Ю. М. З а б о л о т н о в

Вопросы, задания и упражнения по курсу «Операционные системы»: лабораторный практикум / сост. *С.В. Востокин.*– Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2012. – 32 с.

Приведены краткие сведения по языку программирования Си, вопросы для самопроверки, задания к лабораторным работам, задания с элементами исследования, а также примеры задач и список экзаменационных вопросов по курсу «Операционные системы».

Лабораторный практикум предназначен для проведения практических, лабораторных и самостоятельных занятий по курсу «Операционные системы» со студентами очной и заочной форм обучения по специальностям 230100, 010300, 010400, 010900.

УДК 004.451(075)
ББК 22.161

1. ПОРЯДОК ПРОВЕДЕНИЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Цель лабораторного практикума - изучить язык системного программирования Си в качестве второго языка; получить практические навыки управления памятью, организации файлового ввода-вывода, многопоточного программирования и управления процессами средствами программного интерфейса Win32.

Практикум предусматривает выполнение восьми лабораторных работ и устный отчет по ним, отчет по теории языка программирования Си и итоговый письменный отчет. В параграфах 2 и 3 данного пособия представлен список вопросов с краткими ответами, а также задачи для самопроверки при подготовке к отчету по теоретическим сведениям о языке Си.

Лабораторный практикум выполняется индивидуально или группой из двух студентов. Задания по лабораторному практикуму приведены в параграфах 4 – 6. Отчет по каждой лабораторной работе принимается индивидуально. График выполнения лабораторных работ и отчетов по ним согласуется с руководителем практики. В случае успешной сдачи первых трех лабораторных работ и отчета по теоретическим сведениям, по желанию, возможен выбор задания с элементами исследования. Список тем заданий приведен в параграфе 7 данного пособия.

В параграфе 8 содержится список экзаменационных вопросов, охватывающих обязательный для изучения в рамках курса «Операционные системы» теоретический материал. Параграф 9 содержит примеры экзаменационных задач по данному курсу.

2. НЕКОТОРЫЕ СВЕДЕНИЯ О ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИ

Какие классы памяти используются в языке Си

В языке Си используются автоматический и статический класс памяти. Автоматические объекты локальны в блоке, при выходе из него значения таких объектов теряются, так как они размещаются во фрейме вызова в стеке. Статические объекты могут быть локальными в блоке или располагаться вне блоков, но в обоих случаях их значения сохраняются после выхода из блока (или функции) до повторного входа в него. Они размещаются в памяти, инициализируемой при загрузке программы по фиксированному адресу. Объект (переменная) - часть памяти с двумя главными характеристиками: класс памяти (время жизни памяти, связанной с объектом), тип (род значений, хранящихся в объекте).

Какие правила определяют область видимости имен

Существуют два вида областей видимости имен: первая область – это лексическая область идентификатора, область в тексте программы, где имеют смысл все его характеристики; вторая область – это область, ассоциируемая с объектами и функциями, имеющими внешние связи, устанавливаемые между идентификаторами из раздельно компилируемых единиц трансляции.

Лексическая область видимости определяется по следующим принципам. Идентификатор объекта (функции) во внешнем объявлении виден от места объявления и до конца единицы трансляции. Параметр функции – в пределах всего тела функции. Переменная внутри функции – от места объявления до конца тела функции. Идентификатор в блоке – от места объявления и до конца блока. Метка – во всем теле функции, где встречается эта метка.

Если идентификатор явно объявлен в начале блока (в том числе тела функции), то любое объявление того же идентификатора, находящееся снаружи этого блока, временно перестает действовать вплоть до конца блока.

Как синтаксически выражается принадлежность переменной к классу памяти

Возможны следующие варианты выражения принадлежности переменной к классу памяти. Если переменная объявлена внутри блока, а спецификатор не задан - это автоматический объект. Внутри блока со спецификатором **`auto`** - автоматический объект. Внутри блока со спецификатором **`register`** – автоматический объект, который по возможности располагается в регистре машины. Внутри блока со спецификатором **`static`** – статический объект. Вне всех блоков при незаданном спецификаторе - статический глобальный объект. Вне всех блоков со спецификатором **`extern`** - статический глобальный объект (атрибут внешней связи). Вне всех блоков со спецификатором **`static`** - статический локальный (в пределах модуля) объект (атрибут внутренней связи).

Как можно определить константу (литерал) целого типа

Константа считается восьмеричной, если начинается с **`O`** (цифры ноль); шестнадцатеричной, если начинается с символов **`Ox`** или **`OX`**, и десятичной во всех остальных случаях. Если добавить суффикс **`u`** или **`U`**, то она будет беззнаковой; суффикс **`1`** или **`L`** – типа long. Тип

константы определяется по следующим правилам. Берется первый из перечисленных типов, который годится для ее представления:

- десятичная без суффикса - **int**, **long int**, **unsigned long int**;
- восьмеричная или шестнадцатеричная без суффикса - **int**, **unsigned int**, **long int**, **unsigned long int**;
- с суффиксом **U** - **unsigned int**, **unsigned long int**;
- с суффиксом **L** - **long int**, **unsigned long int**;
- с суффиксом **UL** - **unsigned long int**.

Что такое escape-символы

Escape - символы (escape – последовательности) - это коды текстовых знаков, которые в данной программе имеют специальное управляющее назначение и потому не могут быть записаны напрямую. Они используются, например, при необходимости добавить в строковую константу символа двойных кавычек, который является управляющим и обозначает конец объявления строки.

Таблица 1. Escape – символы, определенные в языке Си

описание символа	обозначение ASCII	запись в Си
новая строка	**NL или LF**	**\n**
горизонтальная табуляция	**HT**	**\t**
вертикальная табуляция	**VT**	**\v**
возврат на шаг	**BS**	**\b**
возврат каретки	**CR**	**\r**
перевод страницы	**FF**	**\f**
сигнал звонка	**BEL**	**\a**
обратная наклонная черта	****	****
знак вопроса	**?**	**\?**
одиночная кавычка	**'**	**\'**
двойная кавычка	**"**	**\"**
восьмеричный код	**Ooo**	**\ooo**
шестнадцатеричный код	**Hh**	**\xhh**

Как определяется константа (литерал) символьного типа, как такая константа будет храниться в памяти

Константа символьного типа заключается в одиночные кавычки (**'x'**). Один символ будет храниться как численное значение этого символа в кодировке, принятой на данной машине (например, ASCII). Хранение нескольких символов зависит от реализации.

Символьная константа принимает тип **char** или **wchar__t**. Последний тип определяет расширенную константу и используется для расширенного набора символов, который не может быть охвачен типом **char**. Чтобы ввести расширенную константу, нужно записать перед ней букву **L**: **L'x'**.

Как определяется строковый литерал. Как поступают, если он не умещается на одной строке

Строковый литерал определяется как последовательность символов, заключающихся в двойные кавычки (**" "**). Строка имеет тип «массив символов» и память класса static. Размер строки определяется нулевым символом **'\0'**, который обозначает ее конец.

Объединить несколько строк в один строковый литерал можно двумя способами: либо написав в конце строки обратную наклонную черту (поскольку символ **** и следующий за ним символ новой строки выбрасываются препроцессором), либо закрыв двойные кавычки в конце одной строки и вновь открыв их на другой.

Как определить размер типа

Размер типа определяется с помощью оператора **sizeof «тип»**, который возвращает размер памяти, выделяемый под объект данного типа в байтах.

Что такое тип void, как он используется

Тип **void** обозначает отсутствие значения, а поэтому применять его можно там, где значение не требуется. Например, как тип возвращаемого значения функции он явным образом подчеркивает факт того, что результат функции отбрасывается.

Указатель на тип **void** можно присваивать, передавать в качестве параметра и результата функции, но операции косвенного обращения и адресной арифметики с ним недопустимы. Кроме того, он играет роль обобщенного указателя: указатель на любой тип данных можно привести к типу **void***, а затем обратно без потери данных.

Это используется в функциях, работающих с любым типом данных. Например, WINAPI – функция вторичного потока

```
DWORD WINAPI ThreadFunc(LPVOID);
```

получает в качестве параметра обобщенный указатель, который затем можно явно привести к нужному типу в теле функции.

Как можно вернуть результат, вычисленный функцией

Результат, вычисленный функцией, можно вернуть либо через инструкцию **`return`** (предварительно указав в определении функции тип возвращаемого значения), либо передав в функцию указатель на объект (и тем самым получить возможность работать с ним, а не с его локальной копией).

Какие операции применимы к указателям

В языке Си можно производить следующие операции с указателями:

- присваивание значения указателя другому указателю того же типа;
- сложение и вычитание указателя и целого;
- вычитание и сравнение двух указателей, указывающих на элементы одного и того же массива;
- присваивание указателю нуля и сравнение указателя с нулем.

В язык C++, помимо этого, добавлена возможность присваивания значения одного указателя другому, даже если эти указатели имеют различный тип. При этом производится неявное преобразование типа.

Напишите функцию, обменивающую значения двух переменных типа `int`

Функция может выглядеть следующим образом.

```
void swap(int *a, int *b){
   int v=0; v=*a; *a=*b; *b=v;
}
```

Переменные передаются в функцию по ссылке. Обмен производится через временную вспомогательную переменную. Вызов имеет вид: **`swap(&x,&y)`**.

Как осуществляется доступ к элементам массива

К i-тому элементу массива можно обратиться либо явно по номеру элемента: **`a[i]`**, либо с использованием адресной арифметики: **`*(a+i)`**. Последний вариант возможен, поскольку переменная массива есть указатель на его нулевой элемент.

Для чего используется ключевое слово `typedef`

Ключевое слово **`typedef`** используется для задания произвольных имен (typedef – имен) существующим типам взамен или в дополнение к стандартным.

Например, после

```
typedef long Blockno, *Blockptr;
typedef struct {double r, theta;} Complex;
```

допустимы следующие объявления:

```
Blockno b;
extern Blockptr bp;
Complex z, *zp;
```

Переменная **b** принадлежит типу **long**, **bp** – типу «указатель на **long**»; **z** – это структура заданного вида, а **zp** принадлежит типу «указатель на такую структуру».

Перечислите инструкции выбора языка Си, как они работают

1) Инструкция **if-else** – принятие решения:

```
if (выражение) инструкция1 else инструкция2;
```

2) Инструкция **else-if** - многоступенчатое принятие решения:

```
if (выражение)
   инструкция
else if (выражение)
   инструкция
else if (выражение)
   инструкция
else
   инструкция;
```

3) Переключатель **switch**:

```
switch (выражение) {
   case константное выражение:
        инструкции
        break;/*если требуется выйти из switch*/
   case константное выражение:
        инструкции
        break;/*если требуется выйти из switch*/
   default: инструкции
}
```

Особенностью **switch** является то, что для выполнения только одной ветви case необходимо вставлять в нее **break**, иначе инструкции продолжают выполняться последовательно и возможен проход по нескольким вариантам выбора.

Так как в местах, где по синтаксису полагается одна инструкция, иногда возникает необходимость выполнить несколько,

предусматривается возможность задания составной инструкции (которую также называют блоком). Тело определения функции есть составная инструкция.

Перечислите инструкции циклов языка Си, как они работают

1) Цикл с предусловием **while**:

```
while(выражение)
   инструкция;
```

2) Цикл с постусловием **do-while**:

```
do
   инструкция
while(выражение);
```

Является аналогом **repeat-until** в Паскале (инструкция будет выполнена хотя бы один раз). Единственное отличие: постусловие является условием продолжения, а не завершения цикла.

3) Цикл **for**:

```
for(выражение1;выражение2;выражение3)инструкция;
```

Относится к циклам с предусловием, поскольку аналогичен следующему коду:

```
выражение1;
while (выражение2) {инструкция; выражение3;}
```

Отличие от цикла **for** из Паскаля в том, что в Си индекс и его предельное значение могут изменяться внутри цикла, и значение индекса после выхода из цикла всегда определено. Кроме того, все три компоненты цикла могут быть произвольными выражениями, поэтому применение **for**-циклов не ограничивается только случаем арифметической прогрессии.

Где может понадобиться инструкция goto

Наиболее типичный случай применения инструкции **goto** – прерывание обработки в некоторой глубоко вложенной структуре и выход сразу из двух или большего числа вложенных циклов. Инструкция **break** здесь не поможет, так как она обеспечит выход только из самого внутреннего цикла. В качестве примера можно взять следующую конструкцию:

```
for(...)
for(...){
   /* если бедствие, уйти на ошибку */
   if (disaster) goto error;
}
error: /* обработка ошибки */
```

Другой пример: необходимо определить, есть ли в двух массивах совпадающие элементы. При использовании цикла **for** инструкция **goto** позволяет избежать дополнительных проверок условий:

```
for(i=0;i<n;i++)
for(j=0;j<m;j++)if(a[i]==b[i]) goto found;
/* нет одинаковых элементов */
found:
/* обнаружено совпадение:a[i]==b[i] */
```

В других случаях следует избегать использования инструкции безусловного перехода по метке, так как это ухудшает читаемость программы.

Как передать массив в качестве аргумента функции

Массив в качестве аргумента функции можно передавать только по ссылке, например:

```
void sorting(int *a,int n);
```

или

```
void sorting(int a[],int n);
```

с последующим вызовом

```
sorting(a,n); .
```

Если требуется передать массив по значению, то необходимо заключить его в какую-либо структуру и передать через нее, поскольку структуры, в отличие от массивов, можно передавать по значению.

Как передать функцию в функцию, как выглядят объявления и вызов переданной функции

В языке Си функции нельзя определять внутри других функций, а сама функция является не переменной, а константой (адресом первой машинной инструкции в коде функции). Однако можно определить указатель на функцию и работать с ним как с обычной переменной, в том числе и передавать в качестве параметра функции.

Примером может служить функция из обобщенного алгоритма сортировки:

```
void qsort(void*lineptr[],
   int left, int right,
   int (*comp)(void*,void*)
);
```

В качестве своих аргументов функция **qsort** ожидает массив указателей, два целых значения и функцию с двумя аргументами типа указатель на **void**. Внутри тела функции **qsort** мы сможем работать с функцией **comp** как с обычной переменной, использовав ее, например, в проверке условия **if**:

```
if((*comp)(v[i],v[left])<0)/*перестановка*/ .
```

Как производится сборка программы из файлов. Перечислите основные стадии этого процесса

Сборка программы из файлов осуществляется следующим образом. В первую очередь исходный код обрабатывается препроцессором, который включает нужные файлы и осуществляет макроподстановки. Полученный код компилируется в объектный файл. Заключительный этап проводится линковщиком, который устанавливает в объектном файле все внешние связи. Конечный результат зависит от типа создаваемой программы (библиотека, исполняемый файл и другие возможные варианты).

Также следует отметить, что проводить полную сборку из-за одного незначительного изменения было бы неэффективно, поэтому язык Си разбивает исходный код на единицы компиляции и при повторной сборке перекомпилирует только те единицы, которые изменились.

Как работает препроцессор языка Си, как исполняются директивы #include, #define, #ifdef (#ifndef)

Препроцессор языка Си работает на первом шаге компиляции и осуществляет:

1) Включение файла.

Любая строка вида

```
#include "имя-файла"
```

или

```
#include <имя-файла>
```

заменяется содержимым файла с именем «имя-файла».

2) Макроподстановку.

```
#define имя замещающий текст
```

Во всех местах, где встречается лексема «имя», вместо нее будет помещен замещающий текст.

3) Условную компиляцию.

```
#if !defined(HDR)
#define HDR
/* здесь содержимое hdr.h */
#endif
```

Условная компиляция – это выборочное включение того или иного текста программы в зависимости от значения условия, вычисляемого во время компиляции (схоже с инструкциями if-else). Она позволяет управлять ходом препроцессирования.

Само препроцессирование проистекает в нескольких логически последовательных фазах (в отдельных реализациях некоторые фазы объединены):

1) Трехзнаковые последовательности заменяются их эквивалентами. Между строками вставляются символы новой строки, если того требует операционная система.
2) Выбрасываются пары символов, состоящие из обратной наклонной черты с последующим символом новой строки, тем самым осуществляется «склеивание» строк.
3) Комментарии заменяются единичными пробелами. Затем выполняются директивы препроцессора и макроподстановки.
4) Escape-последовательности в символьных константах и строковых литералах заменяются на символы, которые они обозначают. Соседние строковые литералы конкатенируются.

Результат препроцессирования транслируется в объектный код. Затем устанавливаются связи с другими программами и библиотеками посредством сбора необходимых программ и данных и соединения ссылок на внешние функции и объекты с их определениями.

3. ЗАДАЧИ НА ЗНАНИЕ ОСНОВ ЯЗЫКА СИ

1. Имеется переменная типа double. Напишите выражение для вычисления первой цифры справа от десятичной точки, считая, что значение переменной записано в формате с фиксированной точкой.

2. Имеется переменная типа double. Напишите выражение для вычисления второй цифры слева от десятичной точки в представлении значения этой переменной в формате с фиксированной точкой. Результат запишите в переменную типа char. Если такой цифры в представлении числа нет, то присвойте переменной значение '_'.
3. Найдите все простые числа от 2 до 1000.
4. Найдите все простые делители числа в диапазоне от 2 до 1000.
5. Запишите код для вычисления произведения двух матриц.
6. Напишите функцию, вычисляющую максимальный элемент в массиве. Запишите различные способы передачи массива в функцию и доступа к элементам в самой функции.
7. Создайте в динамической памяти и заполните значением 1.0 нижнетреугольную матрицу размером 10x10, выделив память только под используемые элементы.
8. Напишите функцию, выполняющую сортировку произвольного массива. Критерий упорядочения элементов передать как параметр функции.

4. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ №№1, 2

Цель лабораторных работ №№ 1, 2: изучение синтаксиса языка Си. В лабораторной работе №1 реализуется алгоритм решения выбранной задачи. В лабораторной работе №2 решение оформляется в виде функций, выполняющих ввод данных пользователя с клавиатуры в консольном режиме, обработку данных согласно заданию и вывод результата.

Вариант №1. Дан двумерный целочисленный массив A(2,10). Известно, что среди его элементов два и только два равны между собой. Напечатать их индексы. Не брать для сравнения одну и ту же пару элементов дважды.

Вариант №2. Составить программу вывода всех трехзначных десятичных чисел, сумма цифр которых равна данному целому числу M.

Вариант №3. В массиве A(N) каждый элемент равен 0, 1 или 2. Переставить элементы массива так, чтобы сначала располагались все нули, затем все двойки и, наконец, все единицы.

Вариант №4. Напечатать все простые числа, не превосходящие заданное число M.

Вариант №5. В написанном выражении ((((1?2)?3)?4)?5)?6 вместо каждого знака «?» вставить знак одной из четырех арифметических операций +, –, *, / так, чтобы результат вычислений равнялся 35. При делении дробная часть в частном отбрасывается. Достаточно найти одно решение.

Вариант №6. Дан одномерный массив. Все его элементы, не равные нулю, переписать, сохраняя их порядок в начало массива, а нулевые элементы – в конец массива.

Вариант №7. Натуральное число называется совершенным, если оно равно сумме всех своих собственных делителей, включая 1. Напечатать все совершенные числа, меньшие, чем заданное M.

Вариант №8. Заданы три числа D, M, Y, которые обозначают число, месяц и год. Найти номер N этого дня с начала года.

Вариант №9. Последовательность чисел определяется следующим образом: первое число – произвольное натуральное число, кратное 3; любое следующее число в последовательности равно сумме кубов всех цифр предыдущего числа. Любая такая последовательность становится постоянной, равной 153. Напишите программу, проверяющую это утверждение.

Вариант №10. Дан одномерный массив положительных вещественных чисел. Преобразовать этот массив следующим образом. Сначала обнуляется минимальный элемент. Затем – максимальный элемент из оставшихся элементов. Далее – минимальный из оставшихся, и так до тех пор, пока не останется единственный элемент. Вывести на экран значение и индекс оставшегося элемента.

5. ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №3

Цель лабораторной работы №3: изучение методов работы с динамической памятью средствами программного интерфейса Win32. В первой части работы пишется программа с функциями создания динамического массива требуемого размера, обработки его согласно заданию, вывода и очистки с использованием функций библиотеки времени исполнения C/C++. Во второй части работы вызовы библиотеки времени исполнения заменяются вызовами программного интерфейса Win32.

Вариант №1. Найти наибольший и наименьший элементы в динамическом массиве размерностью NxM.

Вариант №2. Необходимо каждый элемент строки разделить на сумму элементов строки в динамическом массиве размерностью NxM.

Вариант №3. Необходимо каждый элемент строки разделить на наибольший элемент строки в динамическом массиве размерностью NxM.

Вариант №4. По динамическому массиву из M вещественных чисел необходимо рассчитать выборочное среднее и выборочную дисперсию.

Вариант №5. Динамический массив размерности MxN необходимо дополнить (M+1)-й строкой и (N+1)-м столбцом, в которых записать суммы элементов соответствующих строк и столбцов. В элементе M+1,N+1 должна храниться сумма всех элементов массива.

Вариант №6. В динамическом массиве размерности MxN необходимо в каждой строке найти элемент с наименьшим значением, а затем среди этих чисел найти наибольшее. На экран вывести индексы этого элемента.

Вариант №7. Транспонировать матрицу, размещенную в динамическом массиве размерности NxN, не используя дополнительного массива.

Вариант №8. В динамическом массиве размерности MxN необходимо найти номер строки и номер столбца, в которых находится наименьший элемент.

Вариант №9. Создать динамический массив из M строк по N символов каждая. Необходимо вывести только те строки, которые являются палиндромами, то есть читаются одинаково слева направо и справа налево.

Вариант №10. Реализовать код для перемножения двух матриц, размещенных в динамических массивах размерности N.

6. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ №4-8

Цель лабораторной работы №4: изучение методов работы с типами данных, определяемых пользователем на языке Си. Требуется реализовать в виде отдельной единицы компиляции (модуля) набор функций и объявлений данных, необходимых для работы с указанным в задании типом данных. В отдельном модуле пишется код для тестирования функций модуля.

Цель лабораторной работы №5: изучение функций ввода/вывода в программном интерфейсе Win32. Интерфейс модуля для работы с заданной структурой данных из задания №4 расширяется функциями для сохранения структуры данных на диске целиком и восстановления структуры данных из сохраненного файла.

Цель лабораторной работы №6: изучение методов работы с динамически подключаемыми библиотеками в программном интерфейсе Win32. Из модуля для работы с заданным типом данных, реализованным в задании №5, строится динамически подключаемая библиотека. Тестирующий код выполняет подключение библиотеки с использованием явного (парой вызовов LoadLibrary/GetProcAddress) и неявного (конфигурированием проекта) связывания.

Цель лабораторной работы №7: изучение методов написания многопоточных приложений и синхронизации потоков в программном интерфейсе Win32. В библиотеку функций для работы с заданной структурой данных, реализованную в задании №5 или №6, добавляется следующая функциональность. Вызов функции для добавления элемента в структуру выполняется в одном потоке, обработка вызова с действительным помещением элементов в нее – в другом потоке. Передача аргументов вызова осуществляется через буфер в памяти, доступ к которому синхронизируется. При каждом добавлении элемента в структуру данных происходит ее сохранение на диск целиком, как в задании №5. Тестирующая программа демонстрирует корректность записи элементов путем чтения файла на диске и печати его содержимого по окончании добавления.

Цель лабораторной работы №8: изучение методов работы с процессами в программном интерфейсе Win32. Задание выполняется по схеме задания №7 за исключением того, что поток, осуществляющий фактическое добавление элементов в структуру данных, реализуется в дочернем процессе.

В заданиях рассматриваются следующие абстрактные типы данных.

Ассоциативный массив (словарь) – абстрактный тип данных (интерфейс к хранилищу данных), позволяющий хранить пары вида (ключ, значение) и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу.

Очередь - структура данных с дисциплиной доступа к элементам «первый пришёл - первый вышел». Добавление элемента возможно лишь в конец очереди, выборка - только из начала очереди, при этом выбранный элемент из очереди удаляется.

Стек - структура данных, в которой доступ к элементам организован по принципу «последним пришёл - первым вышел». Добавление элемента возможно только в вершину стека. Удаление элемента тоже возможно только из вершины стека, при этом второй сверху элемент становится верхним.

Дэк - двусвязная (двухсторонняя) очередь, «очередь с двумя концами» - структура данных, в которой элементы можно добавлять и удалять как в начало, так и в конец.

Связный список -структура данных, состоящая из узлов, каждый из которых содержит как собственные данные, так и одну или две ссылки на следующий и/или предыдущий узел списка.

Хэш-таблица – структура данных, реализующая ассоциативный массив. При реализации по методу цепочек представляет собой массив указателей на списки. Индекс массива – это хэш-код ключа из пары (ключ, значение). Список, соответствующий элементу массива, состоит из элементов (ключ, значение), имеющих одинаковый хэш-код, равный значению индекса этого списка в массиве.

Вариант №1. Требуется реализовать структуру данных «ассоциативный массив», используя динамический массив.

Вариант №2. Требуется реализовать структуру данных «ассоциативный массив», используя связный список.

Вариант №3. Требуется реализовать структуру данных «ассоциативный массив», используя бинарное дерево.

Вариант №4. Требуется реализовать структуру данных «ассоциативный массив», используя хэш-таблицу, построенную по методу цепочек.

Вариант №5. Реализовать очередь на основе связного списка.

Вариант №6. Реализовать очередь на основе динамического массива.

Вариант №7. Реализовать стек на основе связного списка.

Вариант №8. Реализовать стек на основе динамического массива.

Вариант №9. Реализовать дек на основе связного списка.

Вариант №10. Реализовать дек на основе динамического массива.

7. ТЕМЫ ЗАДАНИЙ С ЭЛЕМЕНТАМИ ИССЛЕДОВАНИЯ

Приведенные ниже темы заданий посвящены разработке системных средств автоматизации многопоточного, параллельного и распределенного программирования. Работа по темам заданий выполняется в рамках проекта «ГрафПлюс» [4]. Средства автоматизации представляют собой надстройку над интегрированной средой программирования MS Visual Studio (или аналогичной). Их работа основана на использовании препроцессора и каркасной библиотеки (фреймворка), построенных в терминах специально разработанной модели программирования. Исследования связаны с проектированием компонент данной системы и ее применением. Подробная информация содержится на сайте проекта по адресу http://graphplus.ssau.ru/. В работе над заданиями используются сведения из следующих дисциплин: теория формальных грамматик, методы параллельных вычислений, численные методы, распределенные алгоритмы, методы защиты информации и др. Задания направлены на развитие навыков программирования на языках: C/C++, C#, Java, Objective C, JavaScript, PHP и др.; языках разметки: XML, HTML, CSS; с использованием API и библиотек: Win32, pthread, TBB, Concurrency Runtime и др.

Выполнение задания с элементами исследования включает в себя разработку технического задания по выбранной теме и согласование его с руководителем, проектирование и программную реализацию. По результатам разработки оформляется бумажный отчет и аннотированный листинг в электронном виде.

1. Реализация паттерна управления вычислениями «портфель задач». Рассмотреть, по выбору, высокопроизводительные системы с общей памятью, распределенной памятью или комбинированные архитектуры. Язык программирования C/C++.
2. Реализация паттернов управления вычислениями для метода переменных направлений. Рассмотреть, по выбору, высокопроизводительные системы с общей памятью, распределенной памятью, вычисления на графических ускорителях или комбинированные архитектуры. Язык программирования C/C++.

3. Имитационное моделирование распределенного алгоритма метода переменных направлений. Требуется реализовать монитор дискретно-событийного моделирования для данного численного метода и провести эксперименты с моделью. Использовать язык программирования C++ для монитора моделирования и язык по выбору для подсистемы визуализации.
4. Разработка и исследование статических методов балансировки нагрузки для численных методов, основанных на паттерне «конвейер». Требуется реализовать монитор дискретно-событийного моделирования для данного численного метода и провести эксперименты с моделью. Язык программирования C++ для монитора моделирования и язык по выбору для подсистемы визуализации.
5. Реализация паттерна управления вычислениями «конвейер». Рассмотреть, по выбору, высокопроизводительные системы с общей памятью, распределенной памятью или комбинированные архитектуры. Язык программирования C/C++.
6. Разработка SAX-парсера. Требуется написать библиотечный модуль SAX-парсера и тесты для него на языке C++. Необходимо изучить спецификацию XML, определить подмножество языка XML, достаточное для описания форматов языка системы, формализовать анализируемый язык методами теории формальных грамматик.
7. Сопряжение с системами программирования Java, .NET и API браузеров. Требуется изучить и документировать методы вызова исполняемого кода из перечисленных сред исполнения. Рассмотреть случаи вызова как процессов, так и методов в подгружаемых динамических библиотеках. Язык программирования подключаемого кода – C++.
8. API графического редактора для браузера. Изучить имеющиеся библиотеки, адаптировать или разработать заново библиотеки для визуализации файлов системы в web-браузере. Языки реализации – HTML5 и JavaScript.
9. Перенос генератора кода на платформы Java, .NET, C (Objective-C). Требуется изучить API многопоточных вычислений в целевой системе программирования и механизм исполнения системы. Используется язык C++ и язык выбранной платформы.

10. Графический редактор IDE. Реализовать редактор файлов описания модулей языка в системе программирования по выбору.
11. Интеграция IDE с системами программирования. Рассмотреть методы интеграции с IDE Visual Studio, Eclipse и другими по выбору.
12. Сайт исследовательского проекта. Требуется выполнить проектирование и рассмотреть возможности хостинга сайта. Цель сайта – организовать взаимодействие внутри группы разработчиков. Допускается интеграция с существующими порталами аналогичного назначения.
13. Система управления запуском заданий на суперкомпьютере. Цель разработки: автоматизация загрузки и выгрузки файлов проекта на суперкомпьютер, компиляция и запуск, наблюдение за статусом запущенного задания. Необходимо создать для пользователя иллюзию локального исполнения программы.
14. Разработка библиотеки паттернов для параллельных алгоритмов в общей памяти. Задание предусматривает нагрузочное тестирование, сравнение с библиотеками TBB и Concurrency Runtime.
15. Разработка и исследование алгоритмов планирования многопоточных вычислений для SMP-систем. Задание предусматривает нагрузочное тестирование, сравнение с библиотеками TBB и Concurrency Runtime. Требуется реализовать разные методы связывания потоков с процессами, рассмотреть методы оптимизации планировщика.
16. Распределенные объекты. Требуется реализовать взаимодействие между процессами через сокеты и MPI, продумать совмещение разрабатываемой системы с планировщиками потоков разных типов.
17. Разработка алгоритмов планирования для динамических сетей процессов. Требуется реализовать на языке C++ алгоритмы планирования, позволяющие перестраивать сеть взаимодействующих процессов в темпе вычислений для SMP-систем.
18. Разработка алгоритмов планирования для систем с распределенной памятью и стационарных сетей

взаимодействующих процессов. Для программ, в которых конфигурация сети процессов определяется перед запуском вычислений, требуется прозрачно для пользователя реализовать исполнение в распределенных кластерных архитектурах. Язык программирования C++.

19. Разработка алгоритмов балансировки для стационарных сетей процессов и гетерогенных систем с распределенной памятью. Реализовать имитационную модель алгоритма балансировки.
20. Дополнить реализацию задания 18 алгоритмом балансировки, разработанным в задании 19. Язык программирования C++.
21. Применение системы в технологиях Web2.0. Требуется переделать генератор кода системы для построения программ на языке JavaScript с целью реализации асинхронных запросов с web-страницы на сервер в рамках технологии AJAX.
22. Разработка библиотеки паттернов для распределенных вычислений. Описать типовые коммуникационные топологии и сравнительное нагрузочное тестирование библиотеки на примерах простых численных методов. Язык программирования C++.
23. Разработка библиотеки паттернов для искусственных нейронных сетей. Рассмотреть известные фреймворки для автоматизации программирования нейронных сетей, методы обобщения в данной предметной области. В дизайне библиотеки необходимо также учитывать методы обучения сетей. Провести нагрузочное тестирование. Предпочтительный язык программирования C++.
24. Разработка библиотеки паттернов для визуализации числовых данных. Разработать библиотеку, пригодную как для десктопных, так и для web-интерфейсов.
25. Разработка паттернов для управления транзакциями. Рассмотреть варианты распределенных и вложенных транзакций. Реализовать алгоритмы транзакций можно «вручную» или средствами программной среды по выбору.
26. Разработка алгоритмов планирования для имитационного моделирования динамических сетей из взаимодействующих процессов. Требуется реализовать среду для моделирования

физических процессов, систем массового обслуживания на основе модели взаимодействующих процессов.

27. Реализация функции сохранения контрольных точек в SMP-планировщиках. Требуется изучить методы отказоустойчивости на основе формирования контрольных точек и реализовать подходящий для планировщика алгоритм. Далее требуется оценить накладные расходы на формирование контрольных точек.
28. Реализация функции сохранения контрольных точек в распределенных алгоритмах. Требуется изучить методы отказоустойчивости на основе формирования контрольных точек и реализовать подходящий для планировщика алгоритм. Далее требуется оценить накладные расходы на формирование контрольных точек. В задании используется конкретный распределенный алгоритм численного метода по выбору.
29. Реализация управляющих алгоритмов реального времени. В отличие от задания 26, требуется реализовать среду для управления физическим (или игровым) процессом на основе данной модели программирования системы.
30. Автоматизированное тестирование программ. Исследовать методы перебора трасс исполнения программ, а также записи истории исполнения для локализации места ошибки. Реализовать соответствующий планировщик. Предпочтительный язык программирования C++.

8. ВОПРОСЫ ПО ТЕОРЕТИЧЕСКОМУ КУРСУ

1. Определение операционной системы и ее функции (виртуальная машина, управление ресурсами, задачи управления ресурсами).
2. Классификация операционных систем (многозадачность, число пользователей, тип многозадачности, многопоточная обработка, особенности управления памятью, критерии эффективности многозадачных ОС, сетевые функции, аппаратные средства, архитектура).
3. Функциональные требования, предъявляемые к операционным системам, и способы их реализации (расширяемость, переносимость, надежность, совместимость, безопасность, производительность).

4. История разработки операционных систем, поколения ЭВМ и операционных систем (лампы – коммутационные панели, транзисторы – пакетные системы, интегральные схемы – многозадачность, СБИС – персональные компьютеры).
5. Основные архитектуры операционных систем (монолитные, многоуровневые, микроядерные, объектно-ориентированные, виртуальные машины).
6. Абстракция процесса, управление процессами в многозадачной операционной системе (определение процесса, диаграмма состояния, контекст, дескриптор, квантование, приоритетное планирование, нити).
7. Функциональные возможности многозадачности в ОС Windows (способы использования многозадачности, решаемые задачи).
8. Планировщик ОС Windows (класс и уровень приоритета, переключение контекста, «неготовые» потоки, динамический приоритет).
9. Эффект инверсии приоритетов (пример, способы преодоления).
10. Мультипроцессорная обработка в ОС Windows (термины, вызовы API, назначение).
11. Эффект гонки (пример, способ преодоления).
12. Средства синхронизации в режиме пользователя в ОС Windows (interlocked-функции, объект «критическая секция»).
13. Задача о критической секции. Алгоритм Питерсона для двух процессов (условия задачи, объяснение принципа алгоритма).
14. Эффект отталкивания, или голодания (использование модификатора volatile, пример возникновения).
15. Эффект ложного разделения переменных (пример влияния кэш-линий).
16. Управление объектами ядра в ОС Windows (описатель объекта, таблица описателей объектов процесса, создание, наследование, именование, дублирование).
17. Средства синхронизации в режиме ядра в ОС Windows (события, семафоры, мьютексы).
18. Эффект взаимоблокировки или возникновения тупика (определение, условия возникновения, моделирование).

19. Стратегия «обнаружение-устранение» для борьбы с взаимоблокировками (с использованием графов Холта, матриц распределения ресурсов).
20. Стратегия избегания блокировок. Алгоритм банкира (диаграмма траекторий ресурсов, алгоритм банкира для одного вида ресурсов).
21. Стратегии предотвращения блокировок (исключение условий в определении блокировок).
22. Методы управления памятью без использования внешней памяти (фиксированные, динамические и перемещаемые разделы).
23. Методы управления памятью с использованием внешней памяти (сегментный, страничный, сегментно-страничный способ).
24. Свопинг. Кэширование (назначение, принцип работы механизма).
25. Реализация сегментного механизма управления памятью в процессорах семейства x86.
26. Реализация страничного механизма управления памятью в процессорах семейства x86 (размер и основные поля структур данных, особенности реализации).
27. Средства ОС Windows для управления виртуальной памятью процесса (VirtualAlloc, структурированная обработка исключений, файлы, отображаемые в память)
28. Архитектура программного обеспечения ввода-вывода (средства ввода – вывода, архитектура программного стека, функции слоев).
29. Общие принципы организации файловых систем и файлов (идентификация файлов в файловых системах, логическая и физическая организация файлов).
30. Введение в распределенные операционные системы (классификация моделей распределенных систем, классификация основных алгоритмов).

9. ЭКЗАМЕНАЦИОННЫЕ ЗАДАЧИ

1. Дана функция, записанная на языке Си, в которой пропущены некоторые фрагменты. Требуется восстановить пропущенные фрагменты таким образом, чтобы получившийся код был синтаксически и семантически корректным.

а)
```
void func(){
   /* объявления и инициализация пропущены */
   x=*y;
   /* вывод пропущен */
}
```
б)
```
void func(){
   /* объявления и инициализация пропущены */
   *x=y;
   /* вывод пропущен */
}
```
в)
```
void func()
{
   /* объявления и инициализация пропущены */
   x=&y;
   /* вывод пропущен */
}
```
2. Объясните смысл выражений с указателями: воспользовавшись графическим представлением переменных, значений и указателей, изобразите состояние памяти после вычисления выражений в каждом случае.

а) `int x[5],*p; p=x+1;`

б) `int x=1; int*y=&x;`

в) `int x;int y[3]={1,2,3}; x=*(y+2);`

г) `int **x,*y,z=0;x=&y;y=&z;`

д) `int r1[2]={11,12};int r2[2]={21,22}; int *m[2]={r1,r2};`

е) `char* s="123";char c=*(s+3);`

3. Напишите код на языке Си для выделения памяти под двумерный массив чисел типа **int** в динамической памяти (куче). Далее запишите код функции, на вход которой подаются указатель на этот массив, индексы элемента в массиве. Функция возвращает значение элемента массива, соответствующее переданным индексам. Индексация элементов - с нуля. Варианты размещения:

а) построчное размещение элементов в памяти;

б) постолбцовое размещение элементов в памяти;
в) древовидное размещение (массив указателей на одномерные массивы).

4. Имеется программа на языке Си

```
/* программа в одном файле */
#include <stdio.h>
int x=123;
int main(int,char**){printf("%d\n",x)}
```

Разделите программу на два файла таким образом, чтобы определение переменной **x** содержалось в одном файле, а ее использование (вывод на печать) – в другом.

5. Имеется два фрагмента кода на языке Си:

а) `int fun1(){ int x=0; x++; return x; }`

б) `int fun2(){ static int x=0; x++; return x; }`

В чем различие функций? Приведите контекст вызова, демонстрирующий разницу между ними.

6. Имеется два фрагмента кода на языке Си:

а) `int fun1(){/*реализация пропущена*/}`

б) `static int fun2(){/*реализация пропущена*/}`

В чем различие функций? Приведите пример, демонстрирующий разницу между ними.

7. С использованием модификаторов **`__declspec(dllimport)`** и **`__declspec(dllexport)`** напишите минимальный код приложения, использующего код произвольной функции, импортируемой из динамической библиотеки:

а) случай неявного связывания (выполняемого загрузчиком при анализе заголовка исполняемого файла);

б) случай явного связывания (вызовами **`LoadLibrary`** и **`GetProcAddress`**).

8. Имеется фрагмент кода (по листингу из отчета), создающий новый поток исполнения. Переделайте код таким образом, чтобы при создании потока в параметр функции потока записывалось целое число. Выведите это число на печать из функции потока.

9. Напишите код с использованием Win32 API функций, реализующий простейшую конструкцию параллельного программирования «разветвление-слияние». Как можно организовать ожидание завершения вторичного потока?

10. Объясните работу программы по листингу из Вашего отчета по лабораторной работе №8.

10. ЛИТЕРАТУРА

1. Рихтер, Дж. Windows для профессионалов. Создание эффективных WIN32-приложений с учетом специфики 64-разрядной версии Windows [Текст] / Дж. Рихтер. – М.: Питер, Русская Редакция, 2001. – 752 с.
2. Керниган, Б.У. Язык программирования С [Текст] / Б.У. Керниган, Д.М. Ритчи. - М.: Вильямс, 2007. – 304 с.
3. Вирт, Н. Алгоритмы и структуры данных [Текст] / Н. Вирт. – СПб.: Невский Диалект, 2001. – 351 с.
4. Востокин, С.В. Графическая объектная модель параллельных процессов и ее применение в задачах численного моделирования [Текст] / С.В. Востокин. - Самара: Изд-во Самарского научного центра РАН, 2007. – 286 с.

СОДЕРЖАНИЕ

Учебное издание

ВОПРОСЫ, ЗАДАНИЯ И УПРАЖНЕНИЯ
ПО КУРСУ
«ОПЕРАЦИОННЫЕ СИСТЕМЫ»

Лабораторный практикум

Составитель ***Востокин Сергей Владимирович***

Редактор Ю.Н. Литвинова
Доверстка А.В. Ярославцева

Подписано в печать 29.10.2012. Формат 60x84 1/16.
Бумага офсетная. Печать офсетная. Печ. л. 2,0.
Тираж 100 экз. Заказ . Арт. – М16/2012.

Самарский государственный аэрокосмический университет.
443086 Самара, Московское шоссе, 34.

Изд-во Самарского государственного аэрокосмического университета.
443086 Самара, Московское шоссе, 34.

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК