САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ

САРТАСОВ СТАНИСЛАВ ЮРЬЕВИЧ

ПРАКТИКУМ НА ЭВМ

МЕТОДИЧЕСКОЕ ПОСОБИЕ

Дисциплина [003576] «Практикум на ЭВМ»
по направлению 09.03.04 Программная инженерия
учебный план рег. № 19/5080/1

САНКТ-ПЕТЕРБУРГ
2019

Содержание

Введение

В это методическое пособие входят практические задания, которые автор предлагает студентам направления 09.03.04 «Программная инженерия» в рамках двухлетнего курса «Практикум на ЭВМ». Первый семестр посвящен базовым понятиям программирования (например, передаче параметра по ссылке и по значению, списковым структурам, косвенной адресации, вызовом подпрограмм через стек и т.д.), обзору различных алгоритмов в информатике и обучению структурной парадигме программирования. Во втором семестре изучается объектно-ориентированная парадигма программирования и основы объектно-ориентированного дизайна. В третьем семестре в фокусе оказываются параллельные алгоритмы обработки данных и механизмы синхронизации многопоточных программ. Наконец, в четвертом семестре все полученные знания в рамках данного курса и курсов [003574] «Основы программирования», [003590] «Введение в программную инженерию», [003617] «Разработка программного обеспечения» собираются воедино в семестровом проекте, параллельно с которым изучаются различные аспекты архитектуры программных систем. В Приложении 2 приведены примерные списки вопросов к теоретическому зачету в конце каждого семестра для большей ясности, какой объем знаний студент должен освоить по завершении курса.

Задания первого семестра предполагается выполнять на языке программирования C. Изначально задания были ориентированы на операционные системы семейства Windows и со второго семестра на язык программирования C#, но в ходе работы оказалось, что все задания можно выполнять в операционных системах семейства Linux, а объектно-ориентированную часть курса – на язые Java, для которой можно найти эквивалентные по функциональности библиотеки (например, MPJ вместо MPI.NET).

При составлении практических заданий и курса в целом автор целенаправленно ориентировался на структурную и объектно-ориентированную парадигмы программирования, так как функциональная парадигма изучается в курсе [003589] «Функциональное программирование».

Некоторая часть задач первого семестра была взята или вдохновлена задачами на сайте Project Euler (https://projecteuler.net). За задачу №6 первого семестра автор благодарит Я.А. Кириленко, а за идею задачи №1 третьего семестра – Н.А. Пенкрата, так как вместе с освоением API файберов студенты получают полезный опыт отладки без использования отладчика.

Salve Lector, & ςφαλματα si quae occurrerint, excusa.

1. Задачи I семестра

В течение семестра студентам необходимо выполнить задачи с 1 по 12. Задачи 13* и 14** - это задачи повышенной сложности, и их правильное решение засчитывается вместо любой другой задачи из первых двенадцати.

1. Вычислить произведение длин своих имени, фамилии и отчества. Вывести на экран двоичное представление следующих величин в указанных форматах данных:

А) отрицательного 32-битного целого, модуль которого равен найденному произведению;

Б) положительного числа с плавающей запятой единичной точности по стандарту IEEE 754, модуль которого равен найденному произведению;

В) отрицательного числа с плавающей запятой двойной точности по стандарту IEEE 754, модуль которого равен найденному произведению.

2. Пифагоровой тройкой называют тройку натуральных чисел (x,y,z), удовлетворяющих условию $x^2+y^2=z^2$. Пифагорова тройка называется примитивной, если числа, её составляющие, взаимно просты. Для введённых пользователем трёх чисел определить, являются ли они пифагоровой тройкой, и если да, являются ли они также простой пифагоровой тройкой. Порядок, в котором числа вводятся, произвольный.

3. Определить, можно ли, исходя из трёх введённых пользователем чисел, построить невырожденный треугольник с соответствующими сторонами. Если возможно, определить его углы в градусах, минутах и секундах с точностью до секунды. Предусмотреть ввод пользователем чисел с дробной частью.

4. Числами Мерсенна называются числа вида 2^N-1, где N – натуральное число. Вывести на экран все простые числа Мерсенна на отрезке $[1; 2^{31}-1]$.

5. Цепная дробь – конечное или бесконечное выражение вида

$$[a_0; a_1, a_2, a_3, \cdots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}}$$

где a_0 – целое, остальные a_i – натуральные числа. Любое вещественное число представимо в виде цепной дроби, при этом рациональные числа представляются конечной цепной дробью, иррациональные – бесконечной. Квадратный корень целого числа, не являющегося квадратом другого целого, является иррациональным числом. Цепная дробь квадратного корня обладает таким свойством, что начиная со следующего за некоторым a_i $a_j=a_{j-i}$: $a_{i+1}=a_1$, $a_{i+2}=a_2$ и т.д. i называют периодом данной цепной дроби. Для введённого пользователем числа, не являющегося квадратом целого, вывести период i и последовательность $[a_0; a_1 \dots a_i]$.

6. Написать программу для сортировки строк в текстовом файле с использованием механизма файлов, отображаемых на память (memory mapped files). Строки в файле оканчиваются по правилам операционной системы (\n или \r\n). В качестве компаратора можно использовать любой адекватный поставленной задаче. Для примера достаточно больших текстовых файлов можно брать файлы с геномами из https://www.ncbi.nlm.nih.gov/genome.

7. Создать структуру данных для хэш-таблицы и определить для неё следующие операции:

- Вставка нового ключа и значения. При этом при достижении некоторого условия (например, слишком длинный список для одной из корзин относительно общего числа элементов в хэш-таблице) должна происходить перебалансировка хэш-таблицы.
- Поиск значения по ключу. В случае, если ключ не найден, возвращается некоторое заранее определённое значение.
- Удаление ключа и значения. Если ключ не найден, ничего не происходит. Обратная перебалансировка не требуется.

8. Требуется написать программу, получающую из командной строки следующие данные:

- Имя входного файла.

- Тип фильтра.
- Имя выходного файла.

Входной файл представляет из себя 24-битный или 32-битный BMP-файл без сжатия, но, возможно, с палитрой. Требуется считать из него данные об изображении и применить к нему один из следующих фильтров согласно переданному параметру:

- Усредняющий фильтр 3x3.
- Усредняющий фильтр Гаусса 3x3 или 5x5. Допустимо использование других размеров фильтра, если будут правильно подобраны параметры гауссианы.
- Фильтр Собеля по X. Допустимо использовать вместо него фильтр Щарра.
- Фильтр Собеля по Y. Допустимо использовать вместо него фильтр Щарра.
- Перевод изображения из цветного в оттенки серого.

Полученное изображение необходимо сохранить по указанному пути выходного файла в формате BMP. Полученный файл должен открываться стандартными средствами просмотра изображений.

Возможный пример командной строки для использования программы: MyInstagram C:\Temp\1.bmp SobelX C:\Temp\2.bmp

9. Реализовать набор из следующих функций и показать их работоспособность:

- `void* myMalloc(size_t size)` – аналог функции malloc;
- `void myFree(void* ptr)` – аналог функции free;
- `void* myRealloc(void* ptr)` – аналог функции realloc;
- `void init()` – вспомогательная функция, инициализирующая необходимые структуры данных;

В функции `init()` происходит выделение большой области динамической памяти штатными средствами. Выделение памяти в реализуемых функциях должно происходить в этой области. За пределами функции `init()` нельзя использовать функции malloc, realloc и free.

10. В Англии в обращении находятся монеты следующего достоинства: 1 пенс, 2 пенса, 5 пенсов, 10 пенсов, 20 пенсов, 50 пенсов, 1 фунт (100 пенсов) и 2 фунта (200 пенсов). Пользователь вводит натуральное число, обозначающее некоторую сумму денег в пенсах. Вывести на экран количество способов, которыми эту сумму можно набрать, пользуясь любым количеством любых английских монет.

11. Цифровым корнем называется число в десятичной системе счисления, полученное из цифр исходного числа путём их сложения и повторения этого процесса над полученной суммой до тех пор, пока не будет получено число, меньшее 10. Например, цифровой корень 467 равен 8.

Известно, что составное число можно разложить на множители различными способами, например

24 = 2x2x2x3 = 2x3x4 = 2x2x6 = 4x6 = 3x8 = 2x12 = 24.

В данном случае умножение на единицу не рассматривается.

Назовём суммой цифровых корней сумму цифровых корней отдельных множителей в разложении составного числа. Например, для 24:

Разложение	Сумма цифровых корней
2x2x2x3	9
2x3x4	9
2x2x6	10
4x6	10
3x8	11
2x12	5
24	6

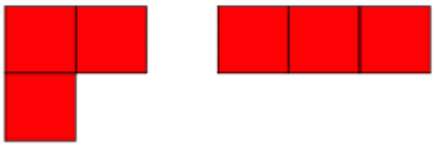
В этом случае максимальная сумма цифровых корней равна 11.

Обозначим максимальную сумму цифровых корней среди всех разложений числа n на множители как MDRS(n).

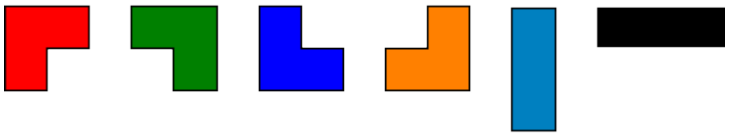
Вычислить $\sum_{n=2}^{999999} MDRS(n)$.

12. Вычислить с помощью алгоритмов длинной арифметики значение числа 3^{5000} и представить его в шестнадцатеричной системе счисления.

13*. По аналогии с домино тромино – форма, состоящая из трёх квадратов с общими сторонами:

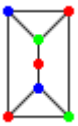
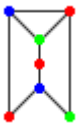
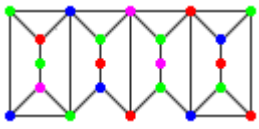


Всего возможных ориентаций тромино 6:



Любую прямоугольную область размера n на m можно заполнить без просветов тромино, если nxm делится на 3. Будем считать, что заполнения, полученные отражением и поворотом – это разные заполнения. Область размером 2x9 можно заполнить 41 различным способом.

Найти число способов, которым можно заполнить область 9x12. Подсказка: в ответе 17 десятичных знаков.

14**. Рассмотрим графы, построенные из элементов A:  и B:  таким образом, что элементы, входящие в состав итогового графа, «склеиваются» вертикальными рёбрами. Пример такого графа:  .

Будем называть конфигурацией типа (a,b,c) граф, построенный из *a* элементов A и *b* элементов B, вершины которого покрашены не более чем *c* цветами таким образом, что никакие две соседние вершины не покрашены в один цвет. Например, вышеприведённый граф – это конфигурация типа (2,2,6), а если точнее – (2,2,c) для любого c ≥ 4.

Обозначим как N(a,b,c) число конфигураций типа (a,b,c). Известно, что:

- N(1,0,3) = 24;
- N(0,2,4) = 92928;
- N(2,2,3) = 20736.

Найти последние 8 цифр N(25,75,1984).

2. Задачи II семестра

1. Переписать консольное приложение по обработке изображений различными фильтрами на C#. Нельзя использовать класс System.Drawing.Bitmap.

2. Реализовать на выбор одну из иерархий объектов:

- Создать абстрактный класс "Танк" с некоторым набором характеристик на ваше усмотрение (вооружение, броня, страна-производитель и т.д.), а также методом или свойством позволяющим получить полную, агрегированную информацию о модели. Реализовать описание нескольких конкретных моделей автомобилей.
- Создать абстрактный класс "Мороженое" с некоторым набором набором характеристик на ваше усмотрение (тип (сливочное, фруктовый лёд), вкус, количество шариков (если применимо) и т.д.), а также методом или свойством позволяющим получить рецепт его приготовления на основе этих характеристик. Реализовать описание нескольких конкретных тортов.

Вынести базовый класс и реализации в отдельные библиотеки. В основной программе выводить в консоль требуемую информацию через методы или свойства абстрактного класса.

3. Реализовать иерархию классов для одной из игр и написать для неё ботов, реализующих как минимум 2 разные стратегии на усмотрение автора и имеющих некоторую стартовую сумму денег. Показать, сколько в среднем денег остаётся у каждого бота после 40 ставок. К решению приложить диаграмму классов UML.

- Блэкджек с базовым вариантом правил и 8 замешанными колодами (https://en.wikipedia.org/wiki/Blackjack) .
- Баккара в варинате Punto banco и 8 замешанными колодами (https://en.wikipedia.org/wiki/Baccarat_(card_game)).
- Рулетка с 1 зеро и ставками на цвет, четность, дюжину и конкретное число (https://en.wikipedia.org/wiki/Roulette).

4. Написать с использованием дженериков класс, реализующий одну из перечисленных коллекций с возможностью добавления, удаления и поиска по ней. Возможно, в реализации потребуется сделать более, чем один класс.

- Двусвязный список.
- Хэш-таблица.
- Динамический массив.

5. Написать приложение для чата. Клиенты устанавливают связь друг с другом напрямую с помощью сокетов. При этом подключение третьего клиента к одному из двух других уже подключенных клиентов приводит к тому, что три клиента объединяются в единое информационное пространство, и сообщение от одного клиента видно всем остальным. Число подключающихся таким образом клиентов не ограничено. Предусмотреть в полном объёме сопутствующую обработку ошибок. Для сетевого взаимодействия использовать класс Socket (т.е. использование классов TcpClient, TcpListener и UdpClient запрещено). Допускается использование как консоли, так и графических технологий (вроде WinForms или WPF) для создания графического интерфейса. Для опроса сокета на получение сообщения можно использовать класс System.Threading.Timer или отдельный поток. Предусмотреть возможность выхода из программы и освобождение ресурсов.

6. Написать приложение, использующее две технологии пользовательского интерфейса (например, WinForms, WPF, JavaFX), которые на экранной форме строят алгебраическую кривую не ниже второго порядка из нескольких заранее определённых (оси координат и сама кривая). Как минимум у одной кривой порядка N коэффициенты как при X^N, так и при Y^N должны быть ненулевыми. Список кривых отображается в комбобоксе. Реализовать возможность масштабирования показываемого изображения. Классы, реализующие связанную с кривыми математику, следует вынести в отдельную общую библиотеку. При использовании WPF отображение кривой должно использовать механизм привязки (binding).

7. С помощью слабых ссылок реализовать хэш-таблицу, которая хранит объекты не менее задаваемого интервала времени, например, минуты. Интервал задаётся параметром конструктора. При необходимости предусмотреть освобождение ресурсов.

8. Реализовать программу, которая ищет по заданному пути библиотеки с плагинами, удовлетворяющими некоторому интерфейсу (определяется в отдельной библиотеке), загружает их и создаёт по одному экземпляру каждого класса, реализующего этот интерфейс.

9. bash – это одна из наиболее популярных командных оболочек под UNIX. Необходимо реализовать свою версию этой программы с урезанной функциональностью. Она должна поддерживать следующие команды:

echo – вывести на экран аргумент(-ы);

exit – выйти из интерпретатора;

pwd – вывести на экран текущий рабочий каталог (название и список файлов);

cat [FILENAME] – показать на экране содержимое файла

wc [FILENAME] – показать на экране количество строк, слов и байт в файле

Команда, не распознанная как одна из приведённых выше, приводит к попытке запуска механизмами операционной системы (вроде Process.Start() в .NET).

Программа также должна поддерживать следующие возможности:

оператор $ - присваивание и использование локальных переменных сессии (например, $PATH, $a=4)

оператор | - конвейерная обработка команд. Результат выполнения одной команды становится входом для другой.

10. Настроить IoC-контейнер по своему выбору, с его помощью инициализировать классы 3 задачи и запустить игру.

3. Задачи III семестра

1. Файберы – это потоки внутри потока, которые надо переключать вручную (https://msdn.microsoft.com/ru-ru/library/windows/desktop/ms682661%28v=vs.85%29.aspx). Пример их реализации доступен в приложении 1. ProcessManagerFramework.cs содержит модель процесса операционной системы, где периоды работы сменяются периодами операций ввода-вывода, и при этом априори невозможно узнать, какова будет длительность этих интервалов.

В операционных системах Windows 3.1 и ранних MacOS не было понятия вытесняющей многозадачности. Вместо этого управление отдавалось другому процессу добровольно и вручную – например, когда процесс находился в ожидании операции ввода-вывода. Это реализовано в модели процесса.

Требуется:

- Создать несколько экземпляров класса Process;
- Сделать две реализации метода ProcessManager.Switch – с приоритетным и бесприоритетным алгоритмом диспетчеризации;
- По завершению всех процессов удалить все файберы кроме основного и корректно выйти из программы.

2. Реализовать указанный параллельный алгоритм (назначается преподавателем) с применением MPI.NET:

- чёт-нечётная сортировка;
- быстрая сортировка;
- быстрая сортировка с использованием регулярного набора образцов;
- алгоритм Флойда;
- алгоритм Прима.

Исходные массивы и графы хранятся в файлах. Результаты работы также записываются в файл. Примеры таких файлов приложены в соответствующих папках в Git. Все числа в файлах представлены в ASCII в десятичной системе счисления.

Размер массива для сортировки – не менее 1000000 элементов. В файл массив записывается в одну строку с пробелом между элементами.

Число вершин V в графе не менее 5000, число рёбер – не менее 1000000 и не более V2/2. В файле с исходным графом данные представлены следующим образом:

- 1 строка – число вершин в графе;
- Последующие строки – описание рёбер в виде троек {индекс_вершины индекс_вершины вес_ребра}. Индекс первой вершины строго меньше индекса второй вершины.

Результат алгоритма Прима – файл с числом вершин в первой строке и весом полученного остовного дерева во второй. Результат алгоритма Флойда – записанная в файл построчно с пробелом между элементами матрица путей, где для каждого элемента индекс строки – начальная вершина пути, индекс столбца – конечная вершина пути.

Предусмотреть корректное завершение работы отдельных процессов.

3. Реализовать решение упрощённой задачи «производитель-потребитель» (буфер не имеет верхней границы) с указанным преподавателем средством синхронизации (атомарные операции, мьютексы, семафоры, мониторы):

- Производители – объекты, кладущие некоторые объекты (например, числа, строки или более сложные объекты-заявки) нестатическими методами в экземпляр класса List<T>;
- Потребители – объекты, извлекающие заявки из экземпляра List<T> в нестатических методах;
- Между двумя последовательными добавлениями у одного и того же производителя или двумя последовательными изъятиями у одного и того же потребителя вставляется пауза (например, с помощью Thread.Sleep);
- Количество производителей и потребителей задаётся константами;
- При запуске программы создаются производители и потребители. Они прекращают работу по нажатию произвольной клавиши. При этом завершение работы производителей и потребителей должно быть корректно реализовано (Thread.Kill таковым не является).

4. Реализовать объект ThreadPool, реализующий паттерн «пул потоков». Число потоков задаётся константой в классе пула. Добавление задачи осуществляется с помощью нестатического метода класса пула Enqueue(Action a). Класс должен быть унаследован от интерфейса IDisposable и корректно освобождать ресурсы при вызове метода Dispose().

5. С помощью стандартной или своей версии концепции Future сделать две различных по схеме распараллеливания реализации интерфейса IVectorLengthComputer для подсчёта модуля вектора, представленного массивом целых:

```
public interface IVectorLengthComputer
{
   int ComputeLength(int[] a);
}
```

В отдельном файле приложить оценки ускорения и эффективности каждой схемы при работе на N процессорах. Для набора формул можно пользоваться Microsoft Equation 3.0, MathType 5, TeX или иные средства.

6. Деканат решил облегчить себе жизнь и заказал матмеху разработку системы, в которую преподавателями и студентами заносится информация о зачётах у последних. Вам поручено реализовать ядро этой системы, удовлетворяющей следующим критериям:

- Зачёты не дифференцированы, либо они есть, либо их нет.
- Зачёт однозначно идентифицируется парой (идентификатор_студента, идентификатор_курса). Оба идентификатора – длинные целые (64 бита) без дополнительных ограничений.
- Общее число пользователей системы – несколько тысяч.

Система должна поддерживать одновременную и непротиворечивую работу с ней нескольких пользователей.

```
public interface IExamSystem
{
   public void Add(long studentId, long courseId);
   public void Remove(long studentId, long courseId);
   public bool Contains(long studentId, long courseId);
}
```

Предложить две различные реализации указанного интерфейса с различными подходами к организации взаимодействия между потоками. Сравнить их быстродействие из соотношения, что 90% всех вызовов – Contains, 9% - Add, 1% - Remove. Использование библиотечных коллекций для организации конкурентного доступа не допускается. Не допускаются реализации с помощью всеобъемлющих высокоуровневых способов вроде:

```
public void Add(long studentId, long courseId)
{
   lock(this)
   {
     …
   }
}
```

7. Написать клиент-серверное приложение с клиентской частью на WPF, WinForms или HTML.

Клиентская часть:

- Получает от сервера список известных ему фильтров;
- Позволяет загружать изображение из файла и отображать его на форме;
- Позволяет отправлять изображение на сервер для применения к нему фильтра;
- Во время ожидания ответа от сервера интерфейс остаётся отзывчивым (то есть, например, можно нажимать на кнопки);
- Во время ожидания ответа от сервера есть возможность отменить предыдущее задание;
- Во время ожидания ответа от сервера отображает текущую степень готовности на контроле вроде ProgressBar.

Серверная часть:

- При запуске загружает список доступных фильтров из конфигурационного файла;
- Выдаёт список доступных фильтров на запрос клиентского приложения;
- Обрабатывает получаемые от клиентских приложений изображения выбранным фильтром и возвращает результат;
- Каждую секунду возвращает клиентскому приложению процент готовности текущего задания.

Провести нагрузочное тестирование серверного приложения, выложить приложение или скрипт для тестирования в Git и оформить результаты отдельным файлом:

- Построить график распределения времени выполнения запросов при фиксированном размере изображения в отсутствие другой нагрузки и при двух заданных уровнях нагрузки, исчисляемых в запросах в секунду.

- Построить графики среднего и медианного времени выполнения запросов при различном общем числе пикселов в изображении в отсутствие другой нагрузки и при двух заданных уровнях нагрузки, исчисляемых в запросах в секунду.
- Найти число клиентов, приводящее к отказу от обслуживания, для некоторого фиксированного размера изображения.

4. Организация семестрового проекта в IV семестре

В IV семестре студентам предлагается реализовать программный проект, начиная с требований и заканчивая его релизом (без сопроводительной релизной документации), на тему, которую они выбирают совместно с преподавателем. Допускается командная работа при условии, что каждый участник будет при презентации проекта рассказывать, что он конкретно сделал в рамках проекта. Студентам рекомендуется с самого начала пользоваться системой контроля версий, и история коммитов помогает подтвердить их слова о распределении ответственности в процессе разработки. Тема проекта не должна быть слишком легкой, а глубина проработки деталей не должна быть чрезмерной. Примеры тем проектов:

- Игра в жанре Tower Defense.
- Мобильное приложение для отслеживания доходов и расходов.
- Игра платформер с адаптивным искусственным интеллектом.
- Программа для управления квадрокоптером.
- Мобильный клиент для социальной сети VKontakte.
- Программа для чтения файлов в формате FB2.
- Бот для Telegram, интегрирующийся с eBay.

Разработка проекта включает в себя следующие этапы:

- Формулировка темы проекта – что студент хочет сделать в сфободной форме. После этого этапа тему менять нельзя.
- Требования к проекту, оформленные в виде IEEE Software Requirements Specification, UML-диаграммы прецедентов и их описания или набора пользовательских историй (User stories).
- Диаграмма классов – какой видится структура классов проектов в начале разработки в нотации UML.
- Демонстрация кода проекта
- Демонстрация готового проекта.
- Фактическая диаграмма классов – структура классов проекта в том виде, в каком она получилась в результате разработки.
- Список известных ошибок и недоработок проекта.

Для стимулирования работы студентов в течение семестра для каждого из этапов назначаются две даты – полудедлайна, к которому студент должен продемонстрировать хотя бы минимальное начало работ по данному этапу, и дедлайна, к которому этап должен быть завершен. Нарушение каждой из этих дат без предупреждения и/или уважительной причины влечет за собой дополнительный вопрос на зачете.

Пример календаря на 2019 год приводится ниже.

Этапы	Полудедлайн	Дедлайн
Тема проекта	25.02.2019	04.03.2019
Требования	11.03.2019	18.03.2019
Диаграмма классов проектная	01.04.2019	08.04.2019
Демонстрация кода проекта	29.04.2019	27.05.2019
Демонстрация готового проекта	-	27.05.2019
Диаграмма классов фактическая	-	27.05.2019
Известные ошибки и недоработки	-	27.05.2019

Рекомендуется вести разработку на объектно-ориентированном языке.

Приложение 1. Листинги для задачи по файберам

Текст обертки файберов под .NET Framework был найден автором в Интернете, и к ней был добавлен класс ProcessManagerFramework. Автор благодарит Швагер Е.А. и Башкирова А.А. за портирование этой обертки под JNA.

Версия для C#:

UnmanagedFiberAPI.cs

```
using System.Runtime.InteropServices;

namespace Fibers
{
    internal static class UnmanagedFiberAPI
    {
        public delegate uint LPFIBER_START_ROUTINE(uint param);

        [DllImport("Kernel32.dll")]
        public static extern uint ConvertThreadToFiber(uint lpParameter);

        [DllImport("Kernel32.dll")]
        public static extern void SwitchToFiber(uint lpFiber);

        [DllImport("Kernel32.dll")]
        public static extern void DeleteFiber(uint lpFiber);

        [DllImport("Kernel32.dll")]
        public static extern uint CreateFiber(uint dwStackSize,
           LPFIBER_START_ROUTINE lpStartAddress, uint lpParameter);
    }
}
```

Fiber.cs

```
using System;

namespace Fibers
{
    public class Fiber
    {

        /// <summary>
        /// The fiber action delegate.
        /// </summary>
        private Action action;

        /// <summary>
```

```
/// Gets the fiber identifier.
/// </summary>
public uint Id { get; private set; }

/// <summary>
/// Gets the id of the primary fiber.
/// </summary>
/// <remarks>If the Id is 0 then this means that there is no
            primary Id on the fiber.</remarks>
public static uint PrimaryId { get; private set; }

/// <summary>
/// Gets the flag identifing the primary fiber (a fiber that can
            run other fibers).
/// </summary>
public bool IsPrimary { get; private set; }

/// <summary>
/// Initializes a new instance of the <see cref="Fiber"/> class.
/// </summary>
/// <param name='action'>Action.</param>
public Fiber(Action action)
{
    InnerCreate(action);
}

/// <summary>
/// Deletes the current fiber.
/// </summary>
/// <remarks>This method should only be used in the fiber action
            that's executing.</remarks>
public void Delete()
{
    UnmanagedFiberAPI.DeleteFiber(Id);
}

/// <summary>
/// Deletes the fiber with the specified fiber id.
/// </summary>
/// <param name='fiberId'>fiber id.</param>
public static void Delete(uint fiberId)
{
    UnmanagedFiberAPI.DeleteFiber(fiberId);
}

/// <summary>
/// Switches the execution context to the next fiber.
/// </summary>
/// <param name='fiberId'>Fiber id.</param>
public static void Switch(uint fiberId)
{
    // for debug only and to show that indeed it works! Remove
            this line!!!
    Console.WriteLine(string.Format("Fiber [{0}] Switch",
            fiberId));

    UnmanagedFiberAPI.SwitchToFiber(fiberId);
}

/// <summary>
/// Creates the fiber.
/// </summary>
```

```
        /// <remarks>This method is responsible for the *actual* fiber
                creation.</remarks>
        /// <param name='action'>Fiber action.</param>
        private void InnerCreate(Action action)
        {
            this.action = action;

            if (PrimaryId == 0)
            {
                PrimaryId = UnmanagedFiberAPI.ConvertThreadToFiber(0);
                IsPrimary = true;
            }

            UnmanagedFiberAPI.LPFIBER_START_ROUTINE lpFiber =
                    FiberRunnerProc;
            Id = UnmanagedFiberAPI.CreateFiber(100500, lpFiber, 0);
        }

        /// <summary>
        /// Fiber method that executes the fiber action.
        /// </summary>
        /// <param name='lpParam'>Lp parameter.</param>
        /// <returns>fiber status code.</returns>
        private uint FiberRunnerProc(uint lpParam)
        {
            uint status = 0;

            try
            {
                action();
            }
            catch (Exception)
            {
                status = 1;
                throw;
            }
            finally
            {
                if (status == 1)
                    UnmanagedFiberAPI.DeleteFiber((uint)Id);
            }

            return status;
        }
    }
}
```

ProcessManagerFramework.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Diagnostics;
using System.Threading;

namespace ProcessManager
{
    public static class ProcessManager
    {
        public static void Switch(bool fiberFinished)
        {
            // a place for fiber magic
        }
    }
```

```
public class Process
{
    private static readonly Random Rng = new Random();

    private const int LongPauseBoundary = 10000;

    private const int ShortPauseBoundary = 100;

    private const int WorkBoundary = 1000;

    private const int IntervalsAmountBoundary = 10;
    private const int PriorityLevelsNumber = 10;

    private readonly List<int> _workIntervals = new List<int>();
    private readonly List<int> _pauseIntervals = new List<int>();

    public Process()
    {
        int amount = Rng.Next(IntervalsAmountBoundary);

        for (int i = 0; i < amount; i++)
        {
            _workIntervals.Add(Rng.Next(WorkBoundary));
            _pauseIntervals.Add(Rng.Next(
                    Rng.NextDouble() > 0.9
                        ? LongPauseBoundary
                        : ShortPauseBoundary));
        }

              Priority = Rng.Next(PriorityLevelsNumber);
    }

    public void Run()
    {
        for (int i = 0; i < _workIntervals.Count; i++)
        {
            Thread.Sleep(_workIntervals[i]); // work emulation
            DateTime pauseBeginTime = DateTime.Now;
            do
            {
                ProcessManager.Switch(false);
            } while ((DateTime.Now -
                 pauseBeginTime).TotalMilliseconds <
                 _pauseIntervals[i]); // I/O emulation
        }
        ProcessManager.Switch(true);
    }

      public int Priority
      {
             get; private set;
      }

    public int TotalDuration
    {
        get
        {
            return ActiveDuration + _pauseIntervals.Sum();
        }
    }

    public int ActiveDuration
```

```
        {
            get
            {
                return _workIntervals.Sum();
            }
        }
    }
}
```

Версия для Java:

UnmanagedFiberAPI.java:

```
package com.lab;

import com.sun.jna.Library;
import com.sun.jna.Native;

public class UnmanagedFiberAPI {

    public interface Kernel32 extends Library {
        int ConvertThreadToFiber(int lpParameter);

        void SwitchToFiber(int lpFiber);

        void DeleteFiber(int lpFiber);

        int CreateFiber(int dwStackSize, EventCallbackInterface
                    lpStartAddress, int lpParameter);
    }

    public static Kernel32 kernel32 = (Kernel32)
                    Native.loadLibrary("kernel32", Kernel32.class);

}
```

Fiber.java

```
package com.lab;

public class Fiber {

    /// The fiber action delegate.
    private Runnable action;

    /// Gets the fiber identifier.
    private int id;

    public int getId() {
        return id;
    }

    /// Gets the id of the primary fiber.
    /// <remarks>If the id is 0 then this means that there is no primary
                    id on the fiber.</remarks>
    private static int primaryId;

    public static int getPrimaryId() {
        return primaryId;
    }
```

```
/// Gets the flag identifing the primary fiber (a fiber that can run
                other fibers).
public boolean isPrimary;

public boolean isPrimary() {
    return isPrimary;
}

/// Initializes a new instance of the <see cref="Fiber"/> class.
/// <param name='action'>Action.</param>
public Fiber(Runnable action) throws InterruptedException {
    innerCreate(action);
}

/// Deletes the current fiber.
/// <remarks>This method should only be used in the fiber action
                that's executing.</remarks>
public void delete() {
    UnmanagedFiberAPI.kernel32.DeleteFiber(id);
}

/// Deletes the fiber with the specified fiber id.
/// <param name='fiberId'>fiber id.</param>
public static void delete(int fiberId) {
    UnmanagedFiberAPI.kernel32.DeleteFiber(fiberId);
}

/// Switches the execution context to the next fiber.
/// <param name='fiberId'>Fiber id.</param>
public static void fiberSwitch(int fiberId) {
    // for debug only and to show that indeed it works! Remove this
                line!!!
    //System.out.println("Fiber [" + fiberId + "] Switch");

    UnmanagedFiberAPI.kernel32.SwitchToFiber(fiberId);
}

/// Creates the fiber.
/// <remarks>This method is responsible for the *actual* fiber
                creation.</remarks>
/// <param name='action'>Fiber action.</param>
private void innerCreate(Runnable action) throws InterruptedException
                {
    this.action = action;

    if (primaryId == 0) {
        primaryId =
                UnmanagedFiberAPI.kernel32.ConvertThreadToFiber(0);
        isPrimary = true;
    }
    EventCallbackInterface lpFiber = new EventCallbackInterface() {
        @Override
        public int callback(int param) throws InterruptedException {
            return fiberRunnerProc(param);
        }
    };

    id = UnmanagedFiberAPI.kernel32.CreateFiber(10050, lpFiber, 0);
}

/// Fiber method that executes the fiber action.
/// <param name='lpParam'>Lp parameter.</param>
/// <returns>fiber status code.</returns>
```

```
    private int fiberRunnerProc(int param) throws InterruptedException {
        int status = 0;

        try {
            action.run();
        } catch (Exception e) {
            status = 1;
            e.printStackTrace();
        } finally {
            if (status == 1) {
                UnmanagedFiberAPI.kernel32.DeleteFiber(id);
            }
        }
        return status;
    }
}
```

Process.java

```
package com.lab;

import java.util.ArrayList;
import java.util.Random;

public class Process {

    private static final int longPauseBoundary = 10000;
    private static final int shortPauseBoundary = 100;
    private static final int workBoundary = 1000;
    private static final int intervalsAmountBoundary = 10;
    private static final int priorityLevelsNumber = 10;

    private final ArrayList<Integer> workIntervals = new ArrayList<>();
    private final ArrayList<Integer> pauseIntervals = new ArrayList<>();

    private int priority;
    private int totalDuration;
    private int activeDuration;

    public ArrayList<Integer> getWorkIntervals() {
        return workIntervals;
    }

    public int getPriority() {
        return priority;
    }

    public void setPriority(int priority) {
        this.priority = priority;
    }

    public int getTotalDuration() {
        return activeDuration + sumOfElements(pauseIntervals);
    }

    public int getActiveDuration() {
        return sumOfElements(workIntervals);
    }

    public Process() {
        Random random = new Random();
        int amount = random.nextInt(intervalsAmountBoundary);

        for (int i = 0; i < amount; i++) {
```

```
                workIntervals.add(random.nextInt(workBoundary));
                pauseIntervals.add(random.nextInt(
                        random.nextDouble() > 0.9
                                ? longPauseBoundary
                                : shortPauseBoundary));
            }
            priority = random.nextInt(priorityLevelsNumber);
        }

        public void run() throws InterruptedException {
            synchronized (this) {
                for (int i = 0; i < workIntervals.size(); i++) {
                    Thread.sleep(workIntervals.get(i)); // work emulation

                    long pauseBeginTime = System.currentTimeMillis();
                    do {
                        ProcessManager.processManagerSwitch(false);
                    }
                    while ((System.currentTimeMillis() - pauseBeginTime) <
                         pauseIntervals.get(i)); // I/O emulation*/
                }

                ProcessManager.processManagerSwitch(true);
            }
        }

        private int sumOfElements(ArrayList<Integer> elements) {
            int sum = 0;
            for (int element : elements) {
                sum += element;
            }
            return sum;
        }
    }
```

EventCallbackInterface.java

```
package com.lab;

import com.sun.jna.Callback;

public interface EventCallbackInterface extends Callback {
    int callback(int param) throws InterruptedException;
}
```

Приложение 2. Примерные списки вопросов к теоретическому зачету

I семестр:

1. Гарвардская, фон Неймановская и гибридная архитектуры ЭВМ.
2. Разрядность процессора и операционной системы. Машинное слово. Порядки следования байтов.
3. Прямое, обратное и дополнительное представление целых чисел в памяти компьютера.
4. Представление вещественных чисел по IEEE 754.
5. Физическая и виртуальная память. Сегментно-страничная организация памяти ЭВМ.
6. Понятие кэша центрального процессора. Влияние кэша на быстродействие программ.

7. Понятие указателя в языке C. Понятие косвенной адресации. Операции с указателями.
8. Стек и динамическая память. Операции с динамической памятью.
9. Сложность алгоритмов, основные нотации. Сортировка Шелла.
10. Быстрая сортировка.
11. Сети сортировки, битоническая сортировка. Radix-сортировка (LSD и MSD).
12. Бинарные деревья поиска. АВЛ-дерево.
13. BR-дерево.
14. B-дерево.
15. R-дерево.
16. Пирамида (куча), основные операции.
17. Алгоритм heapsort. Приоритетные очереди.
18. Декартово дерево.
19. Система непересекающихся множеств.
20. Динамическое программирование. Основные этапы и примеры алгоритмов.
21. Понятие о жадных алгоритмах.
22. Основные понятия теории графов. Обходы.
23. Алгоритм Дейкстры для нахождения минимальных путей.
24. Алгоритм Краскала для построения минимального остовного дерева.
25. Раскраска графов. Понятие хроматического многочлена.
26. Основные понятия теории автоматов. Принцип работы автоматов Мили и Мура.
27. Длинная арифметика. Сложение. Умножение в столбик.
28. Алгоритм умножения Карацубы.
29. Алгоритм умножения Тоома-Кука.
30. Алгоритм деления в длинной арифметике.

II семестр:

1. Понятие класса и экземпляра класса.
2. Понятие поля. Понятие метода.
3. Модификаторы видимости.
4. Жизненный цикл объекта.
5. Понятие свойства в C#. Аксессоры.
6. Ключевое слово static. Ключевое слово this.
7. Понятие наследования. Наследование в языке C#.
8. Понятие интерфейса. Понятие абстрактного класса.
9. Диаграмма классов UML.
10. Понятие инкапсуляции.
11. Принципы SOLID: S, O.
12. Принципы SOLID: L.
13. Принципы SOLID: I, D.
14. Параметрический и ad hoc полиморфизм.
15. Обобщённое программирование на примере дженериков.
16. Понятие делегата. Понятие события.
17. Понятие рефлексии.
18. IoC-контейнеры.

III семестр:

1. Таксономия Флинна и типы параллелизма. Процессы, потоки, файберы. Диспетчеризация в операционных системах.
2. Закон Амдала. Закон Густафсона-Барсиса.
3. Операция merge-split. Распараллеливание пузырьковой сортировки.
4. Распараллеливание quicksort.
5. Параллельный алгоритм Флойда.
6. Параллельный алгоритм Прима.
7. MPI: основные понятия и операции.
8. Понятие взаимного исключения и критической секции. Deadlock, livelock, голодание. Барьер памяти.
9. Алгоритм Петерсона. Доказательство корректности.
10. Алгоритм Лэмпорта. Понятие консенсуса, теорема о консенсусе (без доказательства).
11. Понятие блокирующего алгоритма, lock-free, wait-free. Проблема ABA.
12. Атомарные операции.
13. Понятие спинлока. Примеры реализации спинлоков по принципу Test-And-Set.
14. Неявная и явная реализация спинлоков с помощью списков.
15. Семафоры и мьютексы. Решение задачи производителей-потребителей с ограниченным буфером на семафорах и мьютексах.
16. Мониторы и переменные условия.
17. Реэнтрантный спинлок. Read-Write Lock.
18. Понятие барьерной синхронизации. Простая барьерная синхронизация.
19. Иерархическая барьерная синхронизация.
20. Высокоуровневая и детализированная синхронизация для односвязного сортированного списка.
21. Оптимистическая синхронизация для односвязного сортированного списка
22. Ленивая синхронизация для односвязного сортированного списка.
23. Неблокирующая синхронизация для односвязного сортированного списка.
24. Алгоритмы синхронизации хэш-таблицы на списках.
25. Неблокирующая синхронизация хэш-таблицы на списках.
26. Алгоритмы синхронизации хэш-таблицы с открытой адресацией.
27. Понятие Future. Ключевые слова async и await в языке C#.
28. Понятие о пуле потоков. Дисциплины обработки задач Work Stealing и Work Sharing.
29. Понятие списка с пропусками. Ленивая синхронизация для списка с пропусками.
30. Базовые понятия GPGPU и технологии NVIDIA CUDA.

IV семестр:

1. Понятие архитектуры программной системы. Подходы к архитектурным решениям (или их отсутствие) в различных методологиях разработки.
2. Понятие паттерна в архитектуре программных систем. Примеры.
3. Понятие требования. Список заинтересованных лиц. Подходы к сбору требований. Участники.
4. Спецификация программной системы согласно IEEE 830.
5. Оформление требований в виде прецедентов.

6. Элементы UML. Диаграмма прецедентов.
7. Пользовательские истории (User stories).
8. Многослойная архитектура. Состав слоёв, распределение ответственностей.
9. Сценарии транзакции. Описание и область применимости.
10. Модель предметной области. Описание и область применимости.
11. Сервисно-ориентированная архитектура. Описание и область применимости.
12. Гексагональная архитектура. Основные решаемые задачи и пути их решения.
13. Плагинная архитектура. Основные решаемые задачи и пути их решения.
14. Архитектура «фильтры и трубы». Основные решаемые задачи и пути их решения.
15. Пиринговая архитектура. Основные решаемые задачи и пути их решения.
16. Шина сервисов. Основные решаемые задачи и пути их решения.
17. Инфраструктурные способы организации сервера приложения.
18. Виртуальные машины и контейнерная виртуализация. Границы применимости.
19. «Зеленое» программное обеспечение и понятие энергетической эффективности программных систем.
20. Определение и практики археологии программного обеспечения.
21. Понятие рефакторинга.

Список литературы

1. Керниган Б., Ритчи Д. Язык программирования Си. - М.: Вильямс, 2017. - 288 с.;
2. Кнут Д. Искусство программирования. Том 1. Основные алгоритмы. - М.: Вильямс, 2017. - 720 с.
3. Кнут Д. Искусство программирования. Том 2. Получисленные алгоритмы. - М.: Вильямс, 2017. - 832 с.
4. Кнут Д. Искусство программирования. Том 3. Сортировка и поиск. - М.: Вильямс, 2017. - 824 с.
5. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#. - 4-е изд. - СПб.: Питер, 2017. - 896 с.
6. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Гамма, Хелм, Джонсон, Влиссидес, - СПб.: Питер, 2016. - 366 с.
7. Shavit N., Herlihy M. The Art of Multiprocessor Programming. – 2-е изд. - Burlington, MA: Morgan Kaufmann Publishers, 2012. - 552 с.
8. Куликов Е.К., Макаров А.А. Элементы параллельного программирования: учебное пособие. – СПб: ООО «Политехника Сервис», 2018. – 60 с.