Архитектура вычислительных систем
Описание заданий. Общие сведения

ВВЕДЕНИЕ

Понятие архитектуры вычислительной системы (ВС) определенного уровня напрямую связано с программированием для конкретного компьютера. Язык программирования в данном случае выступает связующим звеном между программистом и ВС, предоставляя всю необходимую функциональность. Многоуровневая организация архитектур ВС определяет разные языки взаимодействия с реальным физическим устройством, что, в свою очередь, во многом определяет стиль (парадигму) программирования.

Парадигма (от греческого παράδειγμα – пример, модель, образец) – в философии, социологии исходная концептуальная схема, модель постановки проблем и их решения, методов исследования, господствующих в течение определенного исторического периода в научном сообществе. Смена парадигм представляет собой научную революцию или эволюционный переход.

Парадигма программирования – это парадигма, определяющая некоторый цельный набор идей и рекомендаций, формирующих стиль и технику написания программ. Например, в объектно-ориентированном программировании программист рассматривает программу как набор взаимодействующих объектов, тогда как в функциональном программировании программа представляется в виде цепочки вычисления функций.

Используя определенный язык, программист по сути держит в голове соответствующую данному языку архитектуру ВС, мысленно представляя, каким образом подсистемы (блоки) этой системы будут задействованы в выполнении его программы. Каждый уровень архитектуры предоставляет свою специфическую виртуальную или физическую ВС, знание особенностей которой позволяет писать корректный и эффективный код.

Практические задания по дисциплине «Архитектура вычислительных систем» направлены на изучение и сравнительный анализ особенностей архитектур компьютеров различных уровней, практическое использование методов программирования, учитывающих эти архитектурные особенности.

Цель выполнения заданий

Изучение особенностей архитектур и структур ВС различных уровней организации, методов программирования, присущих этим архитектурам. Анализ взаимосвязи между архитектурами различных уровней организации.

Понимание особенностей структурных решений, поддерживающих различные уровни архитектур ВС.

Особенности выполнения заданий

Для понимания особенностей различных архитектур ВС предлагается выполнение одного и того же варианта практического задания с использованием языков программирования, отражающих конкретную архитектурную специфику. В рамках каждого задания необходимо разработать программу, соответствующую выбранному варианту. Каждое из заданий связано с написанием кода для конкретной заданной архитектуры. Для практического освоения предлагаются следующие архитектуры ВС:

1) архитектура статически типизированного универсального языка программирования, ориентированная на процедурный подход;
2) архитектура статически типизированного универсального языка программирования, ориентированная на объектно-ориентированный подход;
3) архитектура ВС, ориентированная на динамическую типизацию и процедурное программирование;
4) архитектура ВС машинного уровня с программированием на языке ассемблера;
5) архитектура ВС с поддержкой параллельного многопоточного программирования с использованием потоков posix threads.

Следует особо подчеркнуть, что дисциплина «Архитектура ВС» не направлена на изучение программирования. Основная идея курса заключается в получении знаний и навыков в использовании различных подходов к написанию кода на различных уровнях организации вычислительной системы, чтобы в дальнейшем эффективно сочетать эти знания при разработке эффективного программного обеспечения в заданной предметной области.

Требования к инструментальным средствам

К сожалению при массовом выполнении сложно обеспечить проверку и контроль выполнения для разнообразных аппаратных и программных средств. Поэтому при выполнении заданий, из всего возможного разнообразия доступных средств, необходимо использовать:

- вычислительную систему с архитектурой x86-64;
- операционную систему Linux;
- языки программирования C/C++, Python, ассемблер nasm;
- для выполнения заданий также необходимо использовать более простые библиотеки уровня ОС и языка C: stdio.h, stdlib.h, string.h и т.д.

Выбор архитектуры машинного уровня x86-64 обуславливается ее массовым распространением, что позволяет не использовать различные эмуляторы и упрощает непосредственное взаимодействие с компьютером.

Свободно распространяемой ОС Linux вполне достаточно для решения заданий. Помимо возможной непосредственной установки ее можно легко запускать в различных эмулирующих средах. В частности, под ОС Windows можно использовать Windows Subsystem for Linux (WSL). На любой платформе можно также запускать Linux под виртуальной машиной, например, VirtualBox. Описания вариантов установки доступны в сети Интернет. Практически можно использовать любой дистрибутив. При этом достаточно консольной версии.

Компиляторы языков программирования C и C++ имеются практически в любом дистрибутиве Linux. Проще при этом ориентироваться на семейство Gnu Compiler Collection (GCC). Также без проблем в Linux устанавливается Python 3 и nasm.

Состав библиотек определяется тем, что они поддерживаются практически всеми компиляторами C/C++, обеспечивая также работу с языками ассемблера. Помимо это предполагается что использования для изучения архитектур ВС стандартной библиотеки C++ нецелесообразно из-за того, что уровень архитектур повышается до прикладного. Поэтому для создания массивов, векторов и списков предлагается использовать только языковые, а не библиотечные средства.

Предлагаемые задания достаточно простые и не требуют для их написания интегрированных средств разработки. Достаточно текстовых редакторов. В качестве дополнительных инструментов могут опционально пригодиться средства сборки проектов cmake и make. Для сохранения результатов работы и обеспечения их проверки необходимо пользоваться одной из систем контроля версий в сети Интернет (предлагается использовать github).

Порядок выполнения

1. Ознакомиться с описанием задания в соответствии с полученным вариантом.
2. Разработать программу в соответствии с условием задания.
3. Провести отладку и тестирование разработанной программы на заранее подготовленных тестовых наборах данных. Количество тестовых наборов данных – не менее пяти. Число уникальных элементов в тестовых наборах должно варьироваться от нуля до 10000. При необходимости, программа должна правильно обрабатывать переполнение по данным. Тестовые наборы до 20 элементов должны вводиться из заранее подготовленных тестовых файлов. Тестовые данные с большим числом элементов должны порождаться программно с использованием генераторов случайных наборов данных. Дан-

ные формируемые генератом случайных наборов должны поддерживать допустимые значения. Управление вводом данных задается из командной строки.

4. Описать структуру используемой ВС с наложением на нее обобщенной схемы разработанной программы.

5. Зафиксировать для отчета основные характеристики программы, такие как: число интерфейсных модулей (заголовочных файлов) и модулей реализации (фалов с определением программных объектов), общий размер исходных текстов, полученный размер исполняемого кода (если он формируется), время выполнения программы для различных тестовых наборов данных.

6. Провести сравнительный анализ по полученным характеристикам с реализациями, сделанными в других заданиях.

Содержание отчета

Отчет по каждому заданию предоставляется в репозитории системы поддержки версий, посвященном дисциплине, полностью размещаясь в специально созданном отдельном каталоге с номером задани. Он должен содержать:

1. Пояснительную записку, в которой должны быть отражены:
   - описание полученного задания;
   - структурная схема изучаемой архитектуры ВС с размещенной на ней разработанной программы;
   - требуемые метрики, определяющие характеристики программы, для различных тестовых прогонов.
   - данные, демонстрирующие сравнени с характеристиками ранее разработанных программ.
2. Исходные тексты программы, разработанной в ходе выполнения задания, которые должны находиться в своем подкаталоге.
3. Тестовые наборы данных, размещенные в отдельном подкаталоге.
4. Результаты тестовых прогонов, размещенные в отдельном подкаталоге.

1. ВАРИАНТЫ ЗАДАНИЙ

В ходе выполнения каждого задания необходимо написать программу, которая должна быть оформлена в виде консольного приложения, удовлетворяющего следующим требованиям:

1. Запуск программы осуществляется из командной строки, в которой указываются: имя запускаемой программы; имя файла с исходными данными; имя файла с выходными данными.

2. Для каждого программного объекта, загружаемого в контейнер (на основе массива максимальной размерности), исходный файл с тестовым набором должен содержать, признак альтернативы, а также список параметров, необходимых этой альтернативе. Этот список должен быть представлен в формате, удобном для обработки компьютером. При больших данных во входном файле должны быть указаны только параметры для генератора случайных наборов данных, который и заполняет контейнер.

3. В выходной файл необходимо вывести введенные в контейнер данные. Помимо этого необходимо вывести информацию об общем количестве объектов, содержащихся в контейнере. После этого в тот же файл необходимо вывести новые данные в соответствии с результатами, полученными в ходе работы программы. Информация для вывода должна быть представлена в форме, удобной для восприятия пользователем.

4. Программа должна иметь модульную структуру, соответствующую выданному варианту задания.

5. Для представления символьных данных, обрабатываемых в программе предлагается использовать только латиницу. Это обусловлено стремлением упростить их обработку. Дело в том, что в Linux в качестве основной кодировки символов принята UTF-8, которая за пределами кодировки ASCII использует двухбайтовый формат, в том числе и для русских букв. Комментарии, пояснительный текст, не требующий обработки можно писать с использованием как русского, так и английского языков, а также транлитом.

1.1. Выбор варианта задания

Каждый из вариантов собирается из двух независимых компонент: условия задачи и функции обработки данных, загруженных в контейнер. Выбор этих составляющих осуществляется на основе номера варианта задания (от 1 до 336), выданного преподавателем.

Пусть N_{var} – номер варианта задания, div – операция целочисленного деления, mod – операция выделения остатка от целочисленного деления. Тогда номер условия задачи N_{task} вычисляется следующим образом:

$$N_{task} = (N_{var} - 1) \bmod 14 + 1.$$

Номер допоолнительной функции N_{cont} определяется по формуле:

$$N_{func} = ((N_{var} - 1) \operatorname{div} 14) \bmod 25 + 1.$$

1.2. Начальное условие задачи

Условие задачи определяет основное задание для составления программы. Множество различных заданий представлено в таблице 1.

Таблица 1

Условия задач на задания

Обобщенный артефакт, используемый в задании	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив)	Общие для всех альтернатив переменные	Общие для всех альтернатив функции
1. Плоская геометрическая фигура, размещаемые в координатной сетке.	1. Круг (целочисленные координата центра окружности, радиус) 2. Прямоугольник (целочисленные координаты левого верхнего и правого нижнего углов) 3. Треугольник (целочисленные координаты трех углов)	Цвет фигуры (перечислимый тип) = {красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый}	Вычисление площади фигуры (действительное число)
2. Плоская геометрическая фигура, размещаемые в координатной сетке.	1. Круг (целочисленные координата центра окружности, радиус) 2. Прямоугольник (целочисленные координаты левого верхнего и правого нижнего углов) 3. Треугольник (целочисленные координаты трех углов)	Цвет фигуры (перечислимый тип) = {красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый}	Вычисление периметра фигуры (действительное число)
3. Объемная (трехмерная) геометрическая фигура.	1. Шар (целочисленный радиус) 2. Параллелепипед (три целочисленных ребра) 3. Правильный тетраэдр (длина ребра – целое)	Плотность материала фигуры (действительное число)	Вычисление объема (действительное число)
4. Объемная (трехмерная) геометрическая фигура.	1. Шар (целочисленный радиус) 2. Параллелепипед (три целочисленных ребра) 3. Правильный тетраэдр (длина ребра – целое)	Плотность материала фигуры (действительное число)	Вычисление площади поверхности (действительное число)
5. Квадратные матрицы с действительными числами	1. Обычный двумерный массив 2. Диагональная (на основе одномерного массива) 3. Нижняя треугольная матрица (одномерный массив с формулой пересчета)	Размерность – целое число	Вычисление среднего арифметического (действительное число)
6. Пассажирский транспорт	1. Самолеты (дальность полета – целое, грузоподъем-	1. Скорость – целое;	Идеальное время прохо-

	ность – целое) 2. Поезда (количество вагонов – целое) 3. Корабли (водоизмещение – целое; вид судна – перечислимый тип = (лайнер, буксир, танкер)	2. Расстояние между пунктами отправления и назначения – действительное.	ждения пути (действительное число)
7. Фильмы	1. Игровой (режиссер – строка символов) 2. Мультфильм (способ создания – перечислимый тип = рисованный, кукольный, пластилиновый…) 3. Документальный фильм (длительност в минутах – целое)	1. Название фильма – строка символов. 2. Год выхода - целое	Частное от деления года выхода фильма на количество символов в названии (действительное число)
8. Языки программирования	1. Процедурные (наличие, отсутствие абстрактных типов данных – булевская величина) 2. Объектно-ориентированные (наследование: одинарное, множественное, интерфейса – перечислимый тип) 3. Функциональные языки (типизация – перечислимый тип = строгая, динамическая; поддержка «ленивых» вычислений – булевский тип)	Популярность в процентах (TIOBI) — действительное Год создания - целое	Частное от деления года создания на количество символов в названии (действительное число)
9. Тексты, состоящие из цифр и латинских букв, зашифрованные различными способами.	1. Шифрование заменой символов (указатель на массив пар: [текущий символ, замещающий символ]; зашифрованный текст – строка символов) 2. Шифрование циклическим сдвигом кода каждого символа на n (целое число, определяющее сдвиг; зашифрованный текст – строка символов) 3. Шифрование заменой символов на числа (пары: текущий символ, целое число – подстановка при шифровании кода символа в виде короткого целого;	Открытый текст – строка символов.	Частное от деления суммы кодов незашифрованной строки на число символов в этой строке (действительное число)

	зашифрованный текст – целочисленный массив)		
10. Кладезь мудрости.	1. Афоризмы (один из авторов – строка символов) 2. Пословицы и поговорки (страна – строка символов) 3. Загадки (ответ – строка символов)	Содержание кладези мудрости – строка символов	Частное от деления количества знаков препинания в содержательной строке на длину строки (действительное число)
11. Различные числа	1. Комплексные (действительная и мнимая части – пара действительных чисел) 2. Простые дроби (числитель, знаменатель – пара целых чисел) 3. Полярные координаты (угол [радиан] – действительное; координаты конечной точки на плоскости)	–	Приведение каждого значения к действительному числу, эквивалентному записанному. Например, для комплексного числа осуществляется по формуле: sqrt(d^2+i^2)), а для полярных координат - расстояние.
12. Животные	1. Рыбы (место проживания – перечислимый тип: река, море, озеро…) 2. Птицы (отношение к перелету: перелетные, остающиеся на зимовку – булевская величина) 3. Звери (хищники, травоядные, насекомоядные… – перечислимый тип)	1. Название – строка символов, 2. Вес в граммах (целое)	Частное от деления суммы кодов незашифрованной строки на вес (действительное число)
13. Растения	1. Деревья (возраст – длинное целое) 2. Кустарники (месяц цветения – перечислимый тип) 3. Цветы (домашние, садовые, дикие… – перечислимый тип)	Название – строка символов.	Частное от деления числа гласных букв в названии на общую длину названия (действительное число)
14. Автомобильный транспорт	1. Грузовик (грузоподъемность кг – целое) 2. Автобус (пассажировместимость – короткое целое)	1. Емкость топливного бака в литрах (целое)	Максимальное расстояние, которое может пройти авто-

	3. Легковой автомобиль (максимальная скорость – короткое целое)	2. Расход топлива на 100 км в литрах (действительное)	мобиль в км (действительное число)

1.3. Обработка данных в контейнере

После размещения данных в контейнер необходимо осуществить их обработку в соответствии с вариантом задания. Обработанные данные после этого заносятся в отдельный файл результатов. Необходимо реализовать одну из следующих функций.

1. Упорядочить элементы контейнера по возрастанию используя сортировку с помощью прямого включения (Straight Insertion). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
2. Упорядочить элементы контейнера по возрастанию используя сортировку Сортировка с помощью прямого выбора (Straight Selection). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
3. Упорядочить элементы контейнера по возрастанию используя сортировку с помощью прямого обмена или пузырька (Bubble Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
4. Упорядочить элементы контейнера по возрастанию используя шейкерную сортировку (Shaker Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
5. Упорядочить элементы контейнера по возрастанию используя сортировку Шелла (Shell Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
6. Упорядочить элементы контейнера по возрастанию используя сортировку с помощью «дерева» (Heap Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
7. Упорядочить элементы контейнера по возрастанию используя сортировку методом деления пополам (Binary Insertion). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
8. Упорядочить элементы контейнера по возрастанию используя сортировку с помощью разделения (Quick Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

9. Упорядочить элементы контейнера по возрастанию используя сортировку с помощью прямого слияния (Straight Merge). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

10. Упорядочить элементы контейнера по убыванию используя сортировку с помощью прямого включения (Straight Insertion). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

11. Упорядочить элементы контейнера по убыванию используя сортировку Сортировка с помощью прямого выбора (Straight Selection). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

12. Упорядочить элементы контейнера по убыванию используя сортировку с помощью прямого обмена или пузырька (Bubble Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

13. Упорядочить элементы контейнера по убыванию используя шейкерную сортировку (Shaker Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

14. Упорядочить элементы контейнера по убыванию используя сортировку Шелла (Shell Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

15. Упорядочить элементы контейнера по убыванию используя сортировку с помощью «дерева» (Heap Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

16. Упорядочить элементы контейнера по убыванию используя сортировку методом деления пополам (Binary Insertion). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

17. Упорядочить элементы контейнера по убыванию используя сортировку с помощью разделения (Quick Sort). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

18. Упорядочить элементы контейнера по убыванию используя сортировку с помощью прямого слияния (Straight Merge). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

19. Удалить из контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, меньше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции.

20. Удалить из контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, больше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы передвинуть в начало контейнера с сохранением порядка.

21. Удалить из контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, меньше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы передвинуть в начало контейнера с сохранением порядка.

22. Переместить в конец контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, больше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

23. Переместить в начало контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, больше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

24. Переместить в конец контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, меньше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

25. Переместить в начало контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, меньше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

При выполнении лабораторной работы необходима поддержка следующих операций:

• Заполнение контейнера данными, поступающими из входного потока (файла). Полученный элемент должен быть размещен в контейнере любым из способов, выбранных программистом.

• Вывод значений всех элементов в выходной поток (файл). Выводятся параметры элементов, размещенных в контейнере. Порядок вывода элементов определяется порядком размещения элементов в контейнере. Вывод осуществляется в стандартный поток и в файл, указанный в командной строке.

2. ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ

2.1. Проверка задания

Проверка каждого задания осуществляется только один раз, после чего выставляется оценка. Поэтому необходимо внимательное выполнение и прогон тестовых наборов до окончательной сдачи.

Для каждого задания устанавливается срок сдачи (дедлайн). Контроль даты и времени сдачи осуществляется по данным последнего изменения в системе контроля версий. Время московское.

Допускается сдача заданий после дедлайна. При этом полученная оценка снижается на один балл за каждые просроченные сутки. Отрицательные баллы не выставляются. Оценка может быть только обнулена.

Результаты проверки оформляются в течение недели после дедлайна для заданий сданных вовремя. Оценки заданий, сданных с опозданием, могут быть также опубликованы с задержкой. Информацю о том, что окончательная версия задания находится в системе контроля версий необходимо передать преподавателю через установленный канал связи (например, e-mail или указанный чат в одном из месенджеров).

2.1. Оценка задания

Необходимо учитывать ряд требований, определяющих окончательную оценку. Максимальная оценка за работу, выполненную в соответствии с общими требованиями, равна 8 баллам. Для ее получения необходимо соблюсти следующие требования.

1. Оформление отчета в соответствии с предъявляемыми требованиями.
2. Формирование полного тестового покрытия.
3. Прохождение программой всех сформированных тестов и тестов добавленных преподавателем в ходе проверки задания.
4. Результаты выполнения задания выложены вовремя в систему контроля версий.

Оценка снижается в тех случаях когда работа не соответствует предъявляемым требованиям по тем или иным пунктам.

Для повышения оценки до 10 баллов необходимо дополнительно проявить инициативу по следующим направлениям.

1. Выход за пределы требований, предъявляемых к выполнению задания. (разбиение на модули, раздельная компиляция применение систем сборки проектов; расширенные тестовые наборы, учитывающие ввод

некорректных данных и реакция на них; комментарии к коду и проекту и т.д.).
2. Авторская интерпретация демонстрирующая требования к выполнению (более детальное представление структуры используемой ВС, инициативное расширение задания и т. д.).
3. Нестандартные но эффективные решения при написании кода.

Окончательная оценка за задание может являться действительным числом. Все округления предполагается делать при подведении итоговых оценок.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Вирт, Н. Алгоритмы и структуры данных. / Н. Вирт. М.: Мир, 1989. 360 с.
2. Легалов, А.И. Разнорукое программирование. А. И. Легалов. Материал расположен в сети Интернет по адресу: http://www.softcraft.ru/paradigm/dhp/index.shtml
3. Рекомендации о характере и структуре системы оценивания результатов обучения студентов образовательных программ высшего образования в НИУ ВШЭ. - Документ доступен через поисковые системы.