**Введение в архитектуру ЭВМ**

# Лекция 13. Абстракции в программировании. Роль языка ассемблера

## План лекции

- Общие положения
- Примеры абстракций
  - Ввод, вывод
  - Дисковая память
  - Графика
  - Сетевые протоколы
- «Закон» протекающих абстракций (Leaky Abstraction)
- Цель изучения языка ассемблера
- Когда используется языка ассемблера

Абстракция (лат. `abstractio` — отвлечение, исключение) — результат процесса абстрагирования - мысленного выделение неких сторон вещи из ее целостности; нечто, взятое отдельно от вещи, с которой оно неразрывно связано, т. е. отвлечение от каких-то элементов вещи.

Наиболее развитой системой абстракций обладает математика. Степень отвлечённости обсуждаемого понятия называется уровнем абстракции. В зависимости от целей и задач можно рассуждать об одном и том же объекте на разных уровнях абстракции.

Уровень абстракции — один из способов сокрытия деталей реализации определенного набора функциональных возможностей.

**Применяется для управления сложностью проектируемой системы при декомпозиции, когда система представляется в виде иерархии уровней абстракции. Активно применяется в информатике и программировании.**

**Абстракция является фундаментальным понятием в областях информатики и разработки программного обеспечения.**

**Рис.1. ЭВМ «Урал-1», два уровня абстракции.**

**Абстракция в информатике также тесно связана с абстракцией в математике из-за их общего акцента на построение абстракций в качестве объектов.**

**NBNB. Абстракции скрывают детали реализации нижнего уровня, уменьшая сложность процесса проектирования.**

**Примеры абстракций**

**Ввод, вывод**

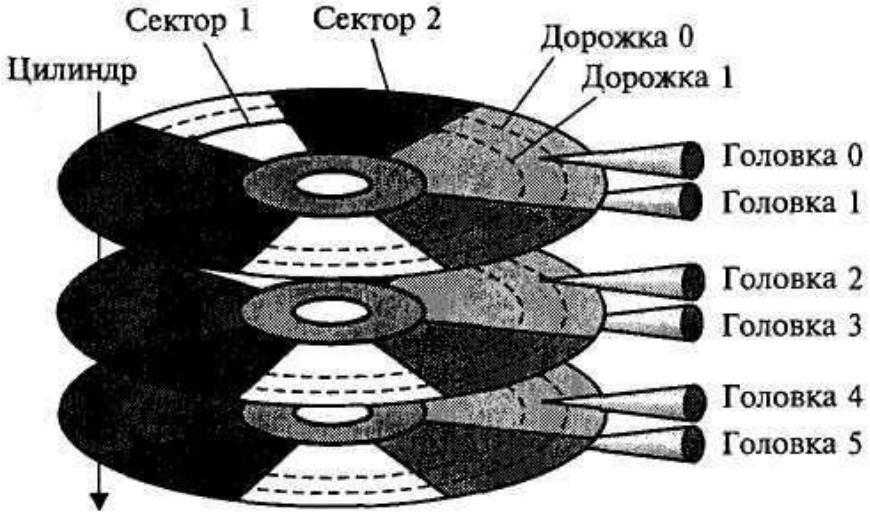
**В ОС Unix большинство операций ввода-вывода рассматриваются как потоки байтов, считываемые или записываемые на устройство. Модель потока байтов используется для ввода-вывода в файл, сокет или компьютерный терминал, чтобы обеспечить независимость от устройства ввода-вывода.**

**Для чтения и записи в устройство на уровне приложения программа вызывает функцию открытия устройства, которое может соответствовать реальному устройству, например, терминалу или виртуальному устройству, например, сокету или файлу в файловой системе.**

**Физические характеристики устройства передаются ОС, которая, в свою очередь, предоставляет абстрактный интерфейс, позволяющий программисту считывать и записывать байты в устройство. ОС затем выполняет действия, необходимые для работы с конкретным устройством.**

**NBNB.** **Со́кет (англ. `socket` — разъём, розетка) — название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения.**

**Дисковая память**



Разбиение жесткого диска на дорожки и секторы

**ОС абстрагируется от тонкостей адресации на жестком диске (дорожки, сектора, цилиндры), чтобы для прикладной программы диск выглядел как набор файлов переменного размера. Опираясь на эту абстракцию, программисты могут создавать файлы, записывать и читать данные, не зная устройства и физической организации жесткого диска.**

**Графика**

**Большинство графических библиотек, например OpenGL, предоставляют в качестве интерфейса абстрактную графическую модель. Библиотека отвечает за трансляцию команд программы в специальные команды устройства, необходимые для прорисовки графических элементов и объектов.**

**Специальные команды для графопостроителя отличаются от команд для монитора, но графическая библиотека скрывает зависящие от устройства детали реализации, предоставляя**

**абстрактный интерфейс, содержащий набор примитивов, общеупотребительных для рисования графических объектов.**

**Сетевые протоколы**

**Классическим примером многоуровневой инженерной системы абстракций является модель взаимодействия открытых систем OSI (Open Systems Interconnection model) ISO имеющая семь уровней. Это концептуальная модель, которая определяет различные уровни взаимодействия систем и функции, выполняемые каждым уровнем при этом взаимодействии. Каждый промежуточный уровень предоставляет несколько сервисов для вышестоящего уровня и получает несколько сервисов от нижележащего уровня.**

**На практике наиболее распространена концептуальная сетевая модель Интернет (Internet Protocol Suite), которая не полностью соответствует модели OSI и разработана и поддерживается технологической комиссией Интернет IETF (Internet Engineering Task Force). В этой модели сетевой уровень - IP предоставляет вышележащему уровню — TCP сервис максимально старательной (best effort) доставки адресату пакетов — дейтаграмм. В свою очередь TCP предоставляет вышележащему прикладному уровню сервис надежной доставки пакетов — сегментов.**

**Таблица 1. Уровни абстракции современных ЭВМ**

| Имя уровня | Представление | Объекты | Операции |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **Прикладные системы сверхвысокого уровня.**<br>**- СУБД (`MySQL`)**<br>**- графические редакторы (`GIMP`)**<br>**- программные каркасы (`frameworks`):**<br>**- `Qt` - кроссссплатфрм;**<br>**- `Flask` — веб разработка.**<br>**- `...`** | **Пакет прикладных программ, установка пакетным менеджером (`zypper`)** | **Определены в прикладной системе.** | **Определены в прикладной системе.** |
| **Прикладные программы. Библиотеки на языках программирования.** | `API` **и реализации библиотек (`glibc`).** | **ЦП, ОП, ВУ, линии связи, файлы, программы.** | **Определены в библиотеках, связь модулей через `ABI`.** |
| **Имя уровня** | **Представление** | **Объекты** | **Операции** |
| **Операционная система. Управление.** | **Модули ОС от поставщика.** | **ЦП, ОП, ВУ, линии связи, файлы, программы.** | **Управление объектами ОС. Язык команд (`shell`) или графический интерфейс.** |
| **Операционная система. Системные вызовы.** | **Модули ОС от поставщика.** | **ЦП, ОП, ВУ, линии связи, файлы, программы.** | **Управление объектами ОС. Вызовы из языков** |

| | | | |
|---|---|---|---|
| | | | программирования. |
| **Имя уровня** | **Представление** | **Объекты** | **Операции** |
| Операционная система. Ядро. | Модули ОС от поставщика. | ЦП, ОП, ВУ, линии связи, файлы, программы. | Управление объектами ОС. |
| Языки высокого уровня (С). | Текст в файле. | Типы данных языка. | Определены в языке. |
| Язык ассемблера. | Символьное, легко читаемое представление ЯКЦП в файле. | Типы данных ЦП и сопроцессоров. | Определены в системе команд. |
| **Имя уровня** | **Представление** | **Объекты** | **Операции** |
| Язык команд ЦП. | Двоичные цифровые коды в ОП. | Типы данных ЦП и сопроцессоров. | Определены в системе команд. |
| Для инженеров по ЭВМ | | | |
| Язык микропрограмм. | Цифровые коды в ПЗУ. | Регистры, вентили, сигналы, | Над элементами логических |

| | | | |
|---|---|---|---|
| | | тактовые импульсы. | схем. |
| Физически реализованные дискретные электронные логические схемы. | СБИС. | Цифровые электрические сигналы. | Преобразование сигналов по жесткой логике схем. |

Микропрограммы состоят из серий микроинструкций, часто называемых элементарными операциями. Микроинструкции управляют дискретными электронными логическими схемами процессором на самом низком уровне. Условный пример последовательности микроинструкций, реализующих микропрограмму команды сложения ЯКЦП:

1. Подсоединить Регистр 1 ко входу «А» арифметическо-логического устройства (АЛУ).
2. Подсоединить Регистр 7 ко входу «Б» АЛУ.
3. Настроить АЛУ на выполнение операции сложения.
4. Установить разряд переноса АЛУ в ноль.
5. Сохранить результат операции в Регистр 8.
6. Обновить «коды состояния» из флагов АЛУ («Отрицательное», «Ноль», «Переполнение», «Перенос»).

«Закон» протекающих абстракций (Leaky Abstraction)

(leaky — протекающий, неплотный, дырявый, негерметичный).

**Закон Дырявых Абстракций, Автор: Джоэл Сполски, Переводчик: Семён Хавкин, Редактор: Маргарита Исаева, 23 марта 2000**

**http://www.grafik.freehand8.ru/057/glava-1-zakon-dyrjavyh-abstrakcij.htm**

**Выдержки из перевода**

**Есть в Интернете инженерное волшебство, на работу которого мы с вами полагаемся каждый день. Оно заключено в сетевом протоколе TCP, одном из основных кирпичей, из которых выстроен Интернет.**

**TCP — способ пересылки данных, который считается надёжным. Это значит: если вы с его помощью отсылаете сообщение по сети, оно обязательно прибудет на место в неискажённом виде. Волшебство же состоит в том, что TCP основан на ненадежном протоколе IP. Иными словами, TCP обязуется работать надёжно, используя лишь ненадёжные детали.**

**Обрыв сетевого кабеля не позволит доставить данные, а подключение к перегруженному каналу приведет к потере настолько большого числа пакетов IP, что работа TCP будет недопустимо замедлена.**

**Вот это я и называю протекающей (имеющей дыры) абстракцией. TCP пытается абстрагироваться от ненадёжной сети полностью, но иногда эта сеть все-таки просвечивает сквозь дыры в абстракции, так что** `абстракция не всегда защищает от необходимости иметь дело с глубокими подробностями.`

**В абстракциях обнаруживаются дыры. В одних немного, в других целая куча. Эти дыры постоянно просвечивают, протекают, абстракции не срабатывают. Вот ещё примеры.**

### Язык SQL

**Язык SQL был создан, чтобы абстрагироваться от процедурных шагов, нужных для запросов к базе данных. Вместо этого он позволяет описать, что именно запрашивается, и пусть база данных сама догадается, какие процедурные шаги для этого нужны.**

**Однако бывает, что некоторые запросы SQL выполняются в тысячи раз медленнее, чем другие, логически им эквивалентные. Известный пример: некоторые сервера SQL значительно быстрее отрабатывают запрос `where a=b and b=c and a=c`, чем `where a=b and b=c`, хотя результат, конечно, тот же самый.**

**Программисту на SQL вроде бы и не следует заботиться о процедуре, только о спецификациях. Но иногда абстракция протекает, что приводит к страшным потерям в производительности, так что приходится лезть во внутренности планировщика запросов и смотреть, что там не так, и как заставить его работать эффективнее.**

### Сетевые файловые системы

**Хотя сетевые библиотеки, вроде NFS и SMB, позволяют работать с файлами реально расположенными на удаленных ЭВМ, как с расположенными на своей, иногда связь становится очень медленной или просто падает, и файл на удаленной ЭВМ перестаёт**

**прикидываться местным; а ведь программисту надо писать код так, чтобы и в этой ситуации всё работало. Значит, в абстракции "всё равно, где лежит этот файл" есть дыра.**

**Вот пример для системных администраторов ОС `Unix`. Если домашние директории лежат на дисках, подмонтированных по `NFS` (одна абстракция), а пользователи создают файл `.forward` для автоматической пересылки почты по другому адресу (вторая абстракция), и сервер `NFS` падает, а почта всё прибывает, то никуда она не перешлётся, поскольку файл `.forward` будет недоступен. Так сквозь дырку в абстракции письма могут «просыпаться на пол».**

**А единственный компетентный способ залатать эти дыры - выучить, как работают абстракции, и какие подробности они скрывают.**

**Итак, абстракции экономят наше рабочее время, но не экономят учебное время.**

**Десять лет назад можно было мечтать, что на сегодняшний день новые компьютерные концепции облегчат труд программиста. И правда: созданные за эти годы абстракции позволяют работать с проектами на порядки более сложными, чем десять или пятнадцать лет назад, типа программирования `GUI` и сетевого программирования.**

**Но хотя замечательные инструменты, вроде современных объектных языков визуальных форм, позволяют сделать много и очень быстро, вдруг в один злосчастный день приходится искать течь в абстракции, и на это уходит пара недель.**

**И наконец!**

---

**Еще раз: абстракции экономят наше рабочее время, но не экономят учебное время.**

**Отсюда парадоксальное следствие: в то время как инструментарий программиста забирается на всё более высокие уровни сложности со всё более развитыми абстракциями, подготовить высококвалифицированного программиста становится всё труднее.**

**Во время мой первой стажировки в Microsoft я писал строковые библиотеки для Макинтоша. Типичное задание: написать версию функции strcat, которая возвращает указатель на конец новой строки. Несколько строчек кода на C. Всё, что я делал, пришло прямо со страниц , одной тоненькой книжки про язык C.**

**Сегодня же для работы над CityDesk'ом мне нужно знать Вижуал Бэйсик, COM, ATL, C++, InnoSetup, внутренности Эксплорера, регулярные выражения (RegExp), DOM, HTML, CSS и XML. Всё это инструменты более высокого уровня по сравнению со старым K&R, а всё ж таки мне и его надо знать, не то беда.**

---

**Цель изучения языка ассемблера**

**Язык ассемблера по-прежнему преподается в большинстве компьютерных и электронных инженерных программ. Хотя сегодня немногие программисты регулярно работают с ассемблером как инструментом, лежащие в его основе концепции остаются важными.**

**Такие фундаментальные темы, как двоичная арифметика, выделение памяти, обработка стека, кодирование набора символов,**

**обработка прерываний и дизайн компилятора, было бы трудно изучить подробно без понимания того, как компьютер работает на аппаратном уровне.**

**Поскольку поведение компьютера в основном определяется его набором инструкций, логическим способом изучения таких понятий является изучение языка ассемблера. Большинство современных компьютеров имеют аналогичные наборы инструкций. Поэтому изучение одного языка ассемблера достаточно для того, чтобы:**

I. **изучить основные понятия;**
II. **распознать ситуации, когда использование языка ассемблера может быть уместным;**
III. **увидеть, насколько эффективный исполняемый код может быть создан из языков высокого уровня.**

**NB. Знание языка ассемблера обеспечивает хорошее понимание уровней абстракции.**

**Когда используется языка ассемблера**

- **Написание кода для систем со старыми процессорами, которые имеют ограниченные возможности языка высокого уровня.**
- **Код, который должен взаимодействовать непосредственно с аппаратным обеспечением, напр**
**имер в драйверах устройств и обработчиках прерываний.**
- **Во встроенных процессорах или DSP (Digital signal processor) — когда прерывания, возникающие с высокой частотой, требуют наименьшего числа циклов обработки на одно прерывание,**

**например прерывание, которое происходит 1000 или 10000 раз в секунду.**

**• Программы, которые должны использовать специфические для процессора инструкции, не реализованные в компиляторе. Распространенным примером является инструкция побитового вращения, лежащая в основе многих алгоритмов шифрования, а также запрос четности байта или 4-битного переноса сложения.**

**• Требуется автономный исполняемый файл компактного размера, который должен выполняться без обращения к компонентам времени выполнения или библиотекам, связанным с языком высокого уровня. Примеры включают в себя встроенное программное обеспечение для телефонов, автомобильных топливных систем и систем зажигания, систем управления кондиционером, систем безопасности и датчиков.**

**• Программы, создающие векторизованные функции для программ на языках более высокого уровня, таких как C. В языке более высокого уровня этому иногда помогают внутренние функции компилятора, которые прямо отображаются в мнемонику SIMD (single instruction, multiple data - одиночный поток команд, множественный поток данных), но тем не менее приводят к ассемблированию вида SISD для данного векторного процессора.**

**• Программы реального времени, такие как моделирование, пилотажно-навигационные системы и медицинское оборудование. Например, в системе fly-by-wire (система управления летательным аппаратом, обеспечивающая передачу управляющих сигналов от органов управления в кабине экипажа (например, от ручки управления самолётом, педалей руля направления) к исполнительным**

приводам аэродинамических поверхностей) телеметрия должна интерпретироваться и обрабатываться в жестких временных рамках.

Такие системы должны устранять источники непредсказуемых задержек, которые могут быть вызваны интерпретируемыми языками, автоматической сборкой мусора, операциями подкачки или упреждающей многозадачностью. Однако некоторые языки более высокого уровня включают компоненты времени выполнения и интерфейсы операционной системы могут привести к таким задержкам. Выбор ассемблера или языков более низкого уровня для таких систем дает программистам большую видимость и контроль над деталями обработки.

- Криптографические алгоритмы, которые всегда должны занимать строго одинаковое время для выполнения, предотвращая временные атаки.
- Компьютерные вирусы, загрузчики, некоторые драйверы устройств или другие элементы, очень близкие к аппаратному обеспечению или низкоуровневой операционной системе.
- Тренажеры набора инструкций для контроля, трассировки и отладки, где дополнительные накладные расходы сведены к минимуму.
- Ситуации, когда не существует языка высокого уровня, на новом или специализированном процессоре, для которого отсутствует кросс-компилятор.
- Реверс-инжиниринг и модификация программных файлов, таких как:
  - существующие двоичные файлы, которые могут быть или не быть изначально написаны на языке высокого уровня, например

при попытке воссоздать программы, для которых исходный код недоступен или был потерян, или при взломе защиты от копирования несвободного программного обеспечения.

- Видео игры (также называемые взломом ROM), который возможен несколькими способами. Наиболее широко используемым методом является изменение программного кода на уровне языка ассемблера.

• Язык ассемблера обычно используется в загрузочном коде системы, низкоуровневом коде, который инициализирует и тестирует аппаратное обеспечение системы перед загрузкой операционной системы и часто хранится в ПЗУ, например BIOS)

• Некоторые компиляторы сначала переводят высокоуровневые языки в ассемблер перед полной компиляцией, что позволяет просматривать ассемблерный код в целях отладки и оптимизации.

• Язык ассемблера полезен в обратном реконструировании программ. Многие программы распространяются только в виде машинного кода, который легко перевести на язык ассемблера с помощью дизассемблера. Такие инструменты, как интерактивный дизассемблер, широко используются для этой цели. Этот метод применяется хакерами для взлома коммерческого программного обеспечения, а конкурентами — для производства программного обеспечения с аналогичными результатами от конкурирующих компаний.

• Ассемблеры могут использоваться для генерации блоков данных, без каких-либо языковых накладных расходов высокого уровня, из форматированного и комментированного исходного кода, который будет использоваться другим кодом.