Разделяй и властвуй!
Латинское изречение

# *Лекция 6.*
# АРХИТЕКТУРА ПРОГРАММНОГО СРЕДСТВА

*Понятие архитектуры и задачи ее описания. Основные классы архитектур программных средств. Взаимодействие между подсистемами и архитектурные функции. Контроль архитектуры программных средств.*

## 6.1. Понятие архитектуры программного средства.

*Архитектура* ПС - это его строение как оно видно (или должно быть видно) из-вне его, т.е. представление ПС как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы. Разработка архитектуры является первым этапом борьбы со сложностью ПС, на котором реализуется принцип выделения относительно независимых компонент.

Основные задачи разработки архитектуры ПС:

- Выделение программных подсистем и отображение на них внешних функций (заданных во внешнем описании) ПС;
- определение способов взаимодействия между выделенными программными подсистемами.

С учетом принимаемых на этом этапе решений производится дальнейшая конкретизация и функциональных спецификаций.

## 6.2. Основные классы архитектур программных средств.

Различают следующие основные классы архитектур программных средств [6.1]:

- цельная программа;
- комплекс автономно выполняемых программ;
- слоистая программная система;
- коллектив параллельно выполняемых программ.

*Цельная программа* представляет вырожденный случай архитектуры ПС: в состав ПС входит только одна программа. Такую архитектуру выбирают обычно в том случае, когда ПС должно выполнять одну какую-либо ярко выраженную функцию и ее реализация не представляется слишком сложной. Естественно, что такая архитектура не требует какого-либо описания (кроме фиксации класса архитектуры), так как отображение внешних функций на эту программу тривиально, а определять способ взаимодействия не требуется (в силу отсутствия какого-либо внешнего взаимодействия программы, кроме как взаимодействия ее

с пользователем, а последнее описывается в документации по применению ПС).

*Комплекс автономно выполняемых программ* состоит из набора программ, такого, что:

- любая из этих программ может быть активизирована (запущена) пользователем;
- при выполнении активизированной программы другие программы этого набора не могут быть активизированы до тех пор, пока не закончит выполнение активизированная программа;
- все программы этого набора применятся к одной и той же информационной среде.

Таким образом, программы этого набора по управлению не взаимодействуют - взаимодействие между ними осуществляется только через общую информационную среду.

*Слоистая программная система* состоит из некоторой упорядоченной совокупности программных подсистем, называемых *слоями*, такой, что:

- на каждом слое ничего не известно о свойствах (и даже существовании) последующих (более высоких) слоев;
- каждый слой может взаимодействовать по управлению (обращаться к компонентам) с непосредственно предшествующим (более низким) слоем через заранее определенный интерфейс, ничего не зная о внутреннем строении всех предшествующих слоев;
- каждый слой располагает определенными ресурсами, которые он либо скрывает от других слоев, либо предоставляет непосредственно последующему слою (через указанный интерфейс) некоторые их абстракции.

Таким образом, в слоистой программной системе каждый слой может реализовать некоторую абстракцию данных. Связи между слоями ограничены передачей значений параметров обращения каждого слоя к смежному снизу слою и выдачей результатов этого обращения от нижнего слоя верхнему. Недопустимо использование глобальных данных несколькими слоями.

В качестве примера рассмотрим использование такой архитектуры для построения операционной системы. Такую архитектуру применил Дейкстра при построении операционной системы THE [6.2]. Эта операционная система состоит из четырех слоев (см. рис. 6.1). На нулевом слое производится обработка всех прерываний и выделение центрального процессора программам (процессам) в пакетном режиме. Только этот уровень осведомлен о мультипрограммных аспектах системы. На первом слое

осуществляется управление страничной организацией памяти. Всем вышестоящим слоям предоставляется виртуальная непрерывная (не страничная) память. На втором слое осуществляется связь с консолью (пультом управления) оператора. Только этот слой знает технические характеристики консоли. На третьем слое осуществляется буферизация входных и выходных потоков данных и реализуются так называемые абстрактные каналы ввода и вывода, так что прикладные программы не знают технических характеристик устройств ввода и вывода.

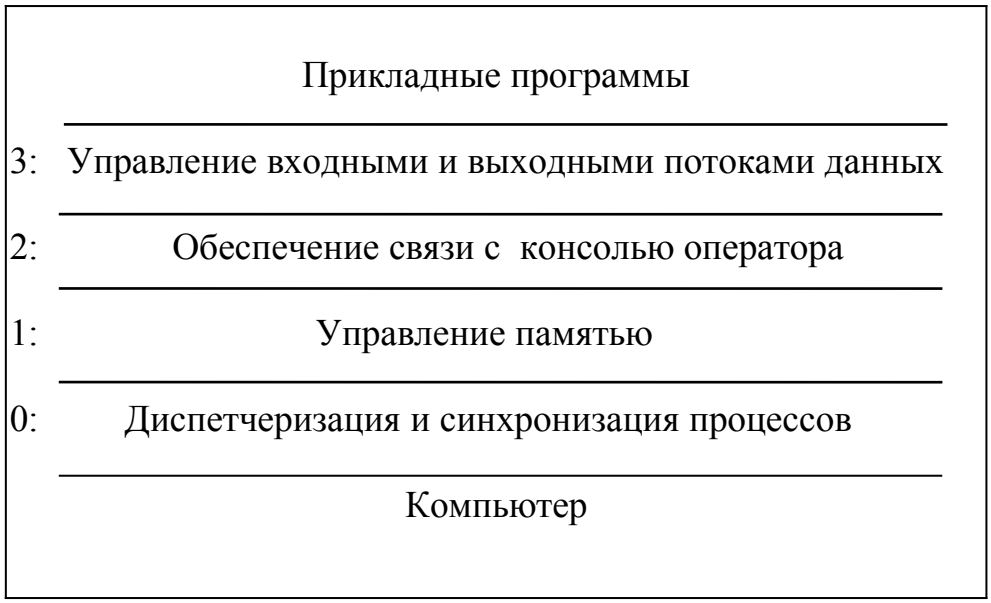


Рис. 6.1. Архитектура операционной системы THE.

*Коллектив параллельно действующих программ* представляет собой набор программ, способных взаимодействовать между собой, находясь одновременно в стадии выполнения. Это означает, что такие программы, во-первых, вызваны в оперативную память, активизированы и могут попеременно разделять по времени один или несколько центральных процессоров, а во-вторых, осуществлять между собой динамические (в процессе выполнения) взаимодействия, на базе которых производиться их синхронизация. Обычно взаимодействие между такими процессами производится путем передачи друг другу некоторых сообщений.

Простейшей разновидностью такой архитектуры является конвейер. Возможности для организации конвейера имеются, например, в операционной системе UNIX [6.3]. *Конвейер* представляет собой последовательность программ, в которой стандартный вывод каждой программы, кроме самой последней, связан со стандартным вводом следующей программы этой последовательности (см. рис. 6.2). Конвейер обрабатывает

некоторый поток сообщений. Каждое сообщение этого потока поступает на ввод первой программе, которая переработанное сообщение передает следующей программе, а сама начинает обработку очередного сообщения потока. Таким же образом действует каждая программа конвейера: получив сообщение от предшествующей программы и, обработав его, она передает переработанное сообщение следующей программе и приступает к обработке следующего сообщения. Последняя программа конвейера выводит результат работы всего конвейера (результирующее сообщение). Таким образом, в конвейере, состоящим из n программ, может одновременно находиться в обработке до n сообщений. Конечно, в силу того, что разные программы конвейера могут затратить на обработку очередных сообщений разные отрезки времени, необходимо обеспечить каким-либо образом синхронизацию этих процессов (некоторые процессы могут находиться в стадии ожидания либо возможности передать переработанное сообщение, либо возможности получить очередное сообщение).

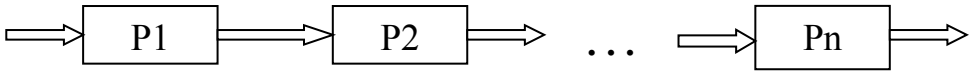


Рис. 6.2. Конвейер параллельно действующих программ.

В общем случае коллектив параллельно действующих программ может быть организован в систему с портами сообщений. *Порт сообщений* представляет собой программную подсистему, обслуживающую некоторую очередь сообщений: она может принимать на хранение от программы какое-либо сообщение, ставя его в очередь, и может выдавать очередное сообщение другой программе по ее требованию. Сообщение, переданное какой-либо программой некоторому порту, уже не будет принадлежать этой программе (и использовать ее ресурсы), но оно не будет принадлежать и никакой другой программе, пока в порядке очереди не будет передано какой-либо программе по ее запросу. Таким образом, программа, передающая сообщение не будет находиться в стадии ожидания пока программа, принимающая это сообщение, не будет готова его обрабатывать (если только не будет переполнен принимающий порт).

Пример программной системы с портами сообщений приведен на рис. 6.3. Порт U может рассматриваться как порт вводных сообщений для представленного на этом рисунке коллектива параллельно действующих программ, а порт W - как порт выводных сообщений для этого коллектива программ.

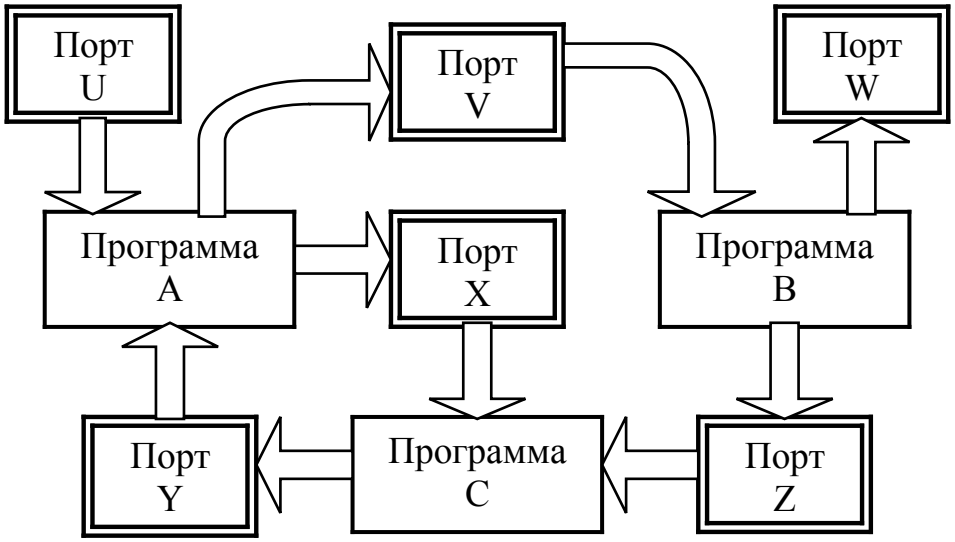


Рис. 6.3. Пример программной системы с портами сообщений.

Программные системы с портами сообщений могут быть как *жесткой* конфигурации, так и *гибкой* конфигурации. В системах с портами жесткой конфигурации каждая программа жестко связывается с одним или с несколькими входными портами. Для передачи сообщения такая программа должна явно указать адрес передачи: имя программы и имя ее входного порта. В этом случае при изменении конфигурации системы придется корректировать используемые программы: изменять адреса передач сообщений. В системах с портами гибкой конфигурации с каждой программой связаны как входные, так и выходные *виртуальные* порты. Перед запуском такой системы программы на основании информации, задаваемой пользователем, должна производиться ее предварительная настройка с помощью специальной программной компоненты, осуществляющей совмещение каждого выходного виртуального порта одной программы с каким-либо входным виртуальным портом другой. Тем самым при изменении конфигурации системы в этом случае не требуется какой-либо корректировки используемых программ - необходимые изменения должны быть отражены в информации для настройки. Однако в этом случае требуется иметь специальную программную компоненту, осуществляющую настройку системы.

### 6.3. Архитектурные функции.

Для обеспечения взаимодействия между подсистемами в ряде случаев не требуется создавать какие-либо дополнительные программные компоненты (помимо реализации внешних функций) -

для этого может быть достаточно заранее фиксированных соглашений и стандартных возможностей базового программного обеспечения (операционной системы). Так в комплексе автономно выполняемых программ для обеспечения взаимодействия достаточно описания (спецификации) общей внешней информационной среды и возможностей операционной системы для запуска программ. В слоистой программной системе может оказаться достаточным спецификации выделенных программных слоев и обычный аппарат обращения к процедурам. В программном конвейере взаимодействие между программами также может обеспечивать операционная система (как это имеет место в операционной системе UNIX).

Однако в ряде случаев для обеспечения взаимодействия между программными подсистемами может потребоваться создание дополнительных программных компонент. Так для управления работой комплекса автономно выполняемых программ часто создают специализированный командный интерпретатор, более удобный (в данной предметной области) для подготовки требуемой внешней информационной среды и запуска требуемой программы, чем базовый командный интерпретатор используемой операционной системы. В слоистых программных системах может быть создан особый аппарат обращения к процедурам слоя (например, обеспечивающий параллельное выполнение этих процедур). В коллективе параллельно действующих программ для управления портами сообщений требуется специальная программная подсистема. Такие программные компоненты реализуют не внешние функции ПС, а функции, возникшие в результате разработки архитектуры этого ПС. В связи с этим такие функции мы будем называть *архитектурными*.

### 6.4. Контроль архитектуры программных средств.

Для контроля архитектуры ПС используется смежный контроль и ручная имитация.

Смежный контроль архитектуры ПС сверху - это ее контроль разработчиками внешнего описания: разработчиками спецификации качества и разработчиками функциональной спецификации. Смежный контроль архитектуры ПС снизу - это ее контроль потенциальными разработчиками программных подсистем, входящих в состав ПС в соответствии с разработанной архитектурой.

Ручная имитация архитектуры ПС производится аналогично ручной имитации функциональной спецификации, только целью этого контроля является проверка взаимодействия между

программными подсистемами. Так же как и в случае ручной имитации функциональной спецификации ПС должны быть сначала подготовлены тесты. Затем группа разработчиков должна для каждого такого теста имитировать работу каждой программной подсистемы, входящей в состав ПС. При этом работу каждой подсистемы имитирует один какой-либо разработчик (не автор архитектуры), тщательно выполняя все взаимодействия этой подсистемы с другими подсистемами (точнее, с разработчиками, их имитирующими) в соответствии с разработанной архитектурой ПС. Тем самым обеспечивается имитационное функционирование ПС в целом в рамках проверяемой архитектуры.

**Упражнения к лекции 6.**

*6.1. Что такое архитектура программного средства?*

*6.2. Что такое архитектурная функция?*

**Литература к лекции 6.**

6.1. Г. Майерс. Надежность программного обеспечения. - М.: Мир, 1980. - С. 78-91.

6.2. E.W. Dijkstra. The Structure of the THE-Multiprogramming // Communications of the ACM. - 1968, 11(5). - Pp. 341-346.

6.3. М. Кристиан. Введение в операционную систему UNIX. - М.: Финансы и статистика, 1985. - С. 46-49.