Современные системы программирования

Состав системы программирования

- Текстовый редактор
- Компилятор
- Компоновщик
- Библиотеки прикладных программ
- Загрузчик
- Отладчик

Взаимосвязь технических средств, используемых при разработке программы

- Подготовка текстов исходной программы
- Подать данные в виде текста исходной программы на вход компилятора
- Получить от компилятора результаты его работы в виде набора объектных файлов
- Подать весь набор полученных объектных файлов на вход компоновщика
- Получить от компоновщика единый исполняемый файл программы и подготовить его к выполнению с помощью загрузчика
- Послать программу на выполнение, при необходимости использовать отладчик для проверки правильности выполнения программы

Программа MAKE (язык MAKEFILE)

```
# Microsoft Developer Studio Generated NMAKE File
CFG=CiMain - Win32 Debug
!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF
CPP=cl.exe
RSC=rc.exe
OUTDIR=.\Debug
INTDIR=.\Debug
# Begin Custom Macros
OutDir=.\Debug
# End Custom Macros
ALL : "$(OUTDIR)\CiMain.exe"
CLEAN :
 -@erase "$(INTDIR)\CiMain.obj"
 -@erase "$(INTDIR)\vc60.idb"
```

```
 -@erase "$(INTDIR)\vc60.pdb"
 -@erase "$(OUTDIR)\CiMain.exe"
 -@erase "$(OUTDIR)\CiMain.ilk"
 -@erase "$(OUTDIR)\CiMain.pdb"
"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
CPP_PROJ=/nologo /MLd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_CONSOLE" /D
"_MBCS" /Fp"$(INTDIR)\CiMain.pch" /YX /Fo"$(INTDIR)\\" /Fd"$(INTDIR)\\" /FD /GZ /c
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)\CiMain.bsc"
BSC32_SBRS= \
LINK32=link.exe
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib /nologo
/subsystem:console /incremental:yes /pdb:"$(OUTDIR)\CiMain.pdb" /debug
/machine:I386 /out:"$(OUTDIR)\CiMain.exe" /pdbtype:sept
LINK32_OBJS= \
 "$(INTDIR)\CiMain.obj"
"$(OUTDIR)\CiMain.exe" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<< $(LINK32_FLAGS) $(LINK32_OBJS) <<

.c{$(INTDIR)}.obj::
   $(CPP).@<< $(CPP_PROJ) $< <<

.cpp{$(INTDIR)}.obj::
   $(CPP) @<< $(CPP_PROJ) $< <<

!IF "$(NO_EXTERNAL_DEPS)" != "1"
!IF EXISTS("CiMain.dep")
!INCLUDE "CiMain.dep"
!ELSE
!MESSAGE Warning: cannot find "CiMain.dep"
!ENDIF
!ENDIF
SOURCE=.\CiMain.cpp
"$(INTDIR)\CiMain.obj" : $(SOURCE) "$(INTDIR)"
```

Интегрированные среды разработки

- Turbo Pascal (Borland International)
- GUI (Graphical User Interface)
- API (Application Programming Interface)

Структура современной системы программирования

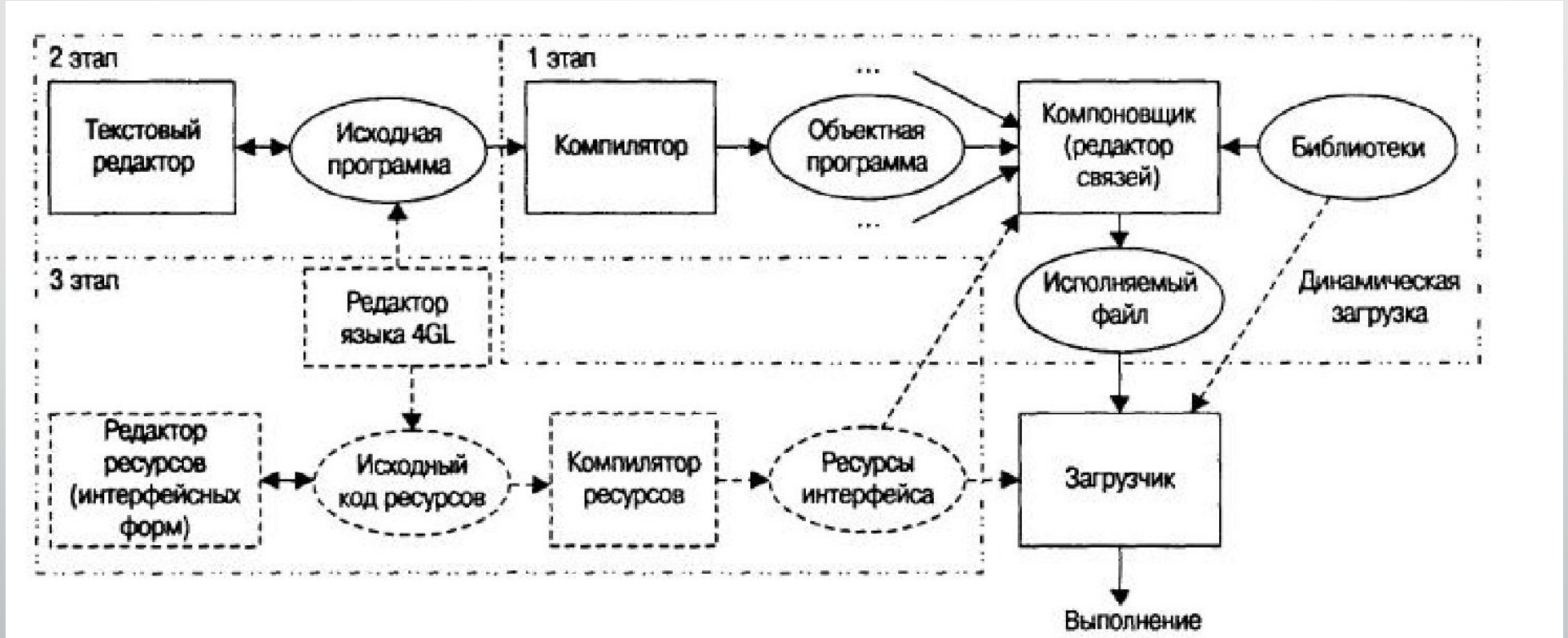

Рис. 6.1. Общая структура и этапы развития систем программирования

Принципы функционирования систем программирования
Функции текстовых редакторов в системах программирования

- Лексический анализ «на лету»
- Системы гиперссылок подсказок и справок (пояснение, вариант кода)
- Справка по семантике и синтаксису входного языка
- Руководство по работе с самой системой программирования
- Справка о функциях библиотек системы программировани

Компилятор
Компоновщик

- Загрузчики и отладчики
- Трансляция адресов. Настраивающий загрузчик
- Динамические загрузчики

Отладчик

- Последовательное пошаговое выполнение результирующей программы на основе шагов по машинным командам или по операторам входного языка;
- Выполнение результирующей программы до достижения ею одной из заданных точек останова;
- Выполнение результирующей программы до наступления некоторых заданных условий, связанных с данными или адресами, выполняемыми этой программой;
- Просмотр содержимого областей памяти, занятых командами или данными результирующей программы.

Дальнейшее развитие отладчиков

- Появление интегрированных средств разработки;
- Появление возможностей аппаратной поддержки средств отладки.

Библиотеки подпрограмм

- Turbo Pascal : TPU – Turbo Pascal Units
- Fortran
- Cobol

Статические библиотеки подпрограмм

Недостатки статических библиотек:

- При наличии ошибки в библиотеке она будет проявляться во всех программах, ее использующих;
- Т.к. объектный код статических библиотек встраивается в исполняемый файл, то это ведет к увеличению объема результирующей программы.

Динамические библиотеки подпрограмм

Варианты загрузки динамических библиотек:

- Сразу же при загрузке программы;
- При непосредственном обращении к ее функциям.

Преимущества динамических библиотек – они не требуют включать в результирующую программу объектный код часто используемых функций.

Недостатки динамических библиотек – результирующие программы связаны с объектным кодом, непосредственно не входящим в их состав.

Ресурсы пользовательского интерфейса

Множество данных, обеспечивающих внешний вид интерфейса пользователя результирующей программы, не связанных напрямую с логикой ее выполнения.

Примеры ресурсов:

- тексты сообщений программы,
- Цветовая гамма интерфейса;
- Надписи на элементах управления.

Редакторы ресурсов

- Язык описания ресурсов
- RAD – средства быстрой разработки приложений.

Мобильность и переносимость программного обеспечения

Мобильность ПО – способность ПО выполнять свои функции на различных вычислительных системах вне зависимости от их архитектуры.

Факторы, влияющие на мобильность ПО:

- Состав специализированных аппаратных средств и периферийных устройств, используемых ПО;
- Тип ОС, на которую ориентировано ПО;
- Состав и функции динамически загружаемых библиотек;
- Структура и формат хранения ресурсов пользовательского интерфейса, используемых ПО;
- Перечень внешних программ и модулей, с которыми взаимодействует данное ПО.

Обеспечение переносимости ИСХОДНОГО КОДА программ

ПО сохраняет способность выполнять свои функции на различных вычислительных системах вне зависимости от их архитектуры в том случае, если оно создано на основе одного и того же ИСХОДНОГО КОДА с помощью некоторой системы программирования и ориентировано на определенную целевую вычислительную систему (переносимость ПО).

Правила обеспечения переносимости исходного кода

- Не использовать в исходном коде прямые обращения к периферийным устройствам компьютера, к драйверам аппаратных средств;
- Не включать в исходный код прямые обращения к драйверам ОС;
- Не использовать статические и динамические библиотеки, ориентированные на определенный тип ОС;
- Использовать только широко распространенные динамические библиотеки, либо библиотеки, для которых имеется исходный код;
- Подключать динамически подключаемые библиотеки только средствами системы программирования, не использовать для этого специфические функции ОС;
- Использовать только средства системы программирования для создания ресурсов пользовательского интерфейса, не обращаться к системным ресурсам;
- Исключить взаимодействие с внешними программами и модулями, либо ограничить его только программами, доступными во всех типах ОС;
- Не использовать для взаимодействия с внешними программами и модулями прямые обращения к средствам ОС.

Мобильность и переносимость

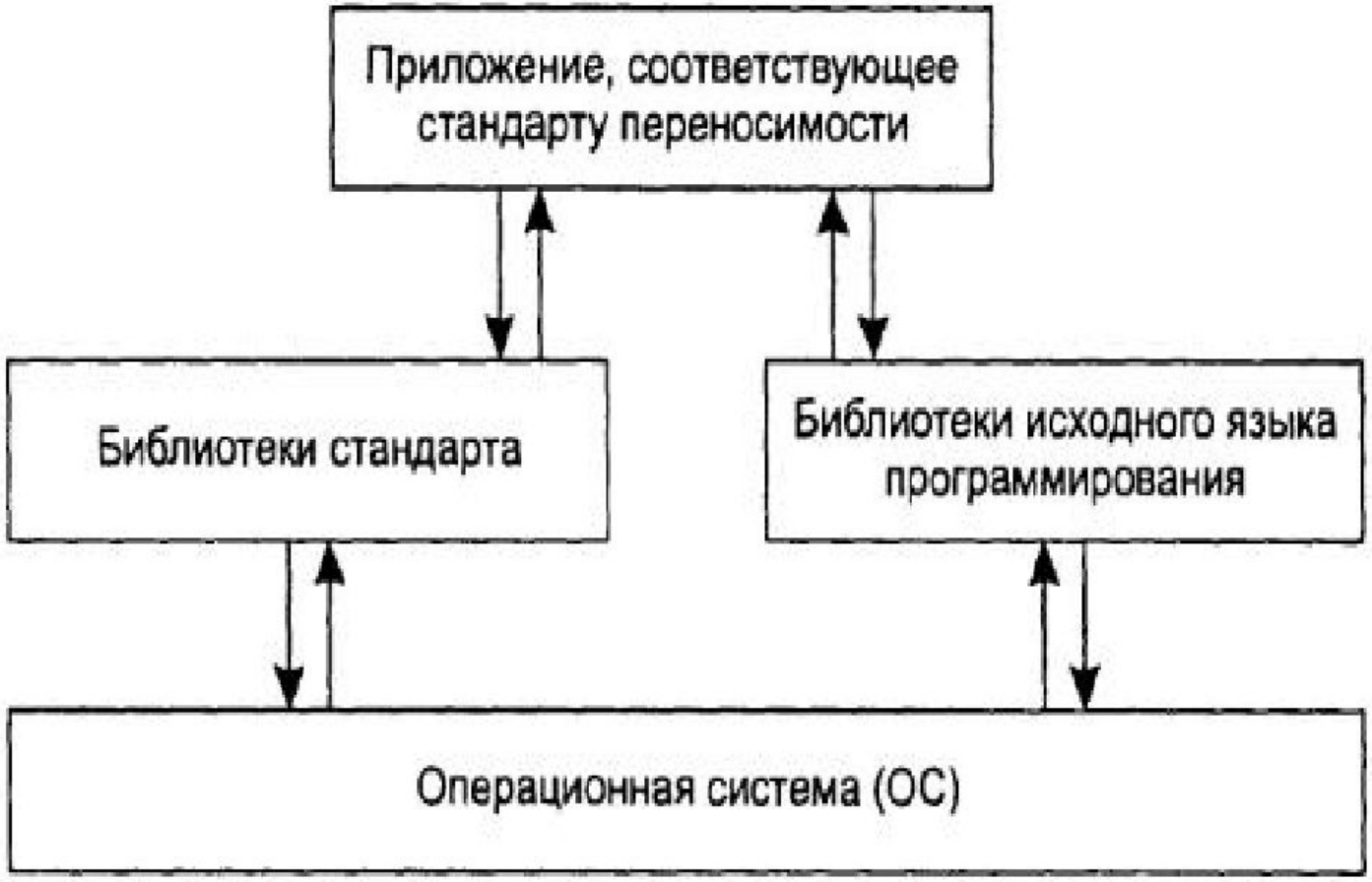

Рис. 6.2. Структура приложения, соответствующего стандарту переносимости

Стандарт переносимости

- POSIX (IEEE Std 1003.1)
- C, C++ (ОС UNIX)
- FORTRAN
- Basic, Pascal

Мобильность ПО на основе интерпретаторов

Полная переносимость исходного кода при условии ограничения на прямое использование функций ОС

Недостатки:

- Исходный текст программы может быть выполнен интерпретатором далеко не для всех ЯП
- Скорость выполнения кода интерпретатором ниже, чем у откомпилированного кода

Мобильность ПО на основе промежуточного двоичного кода

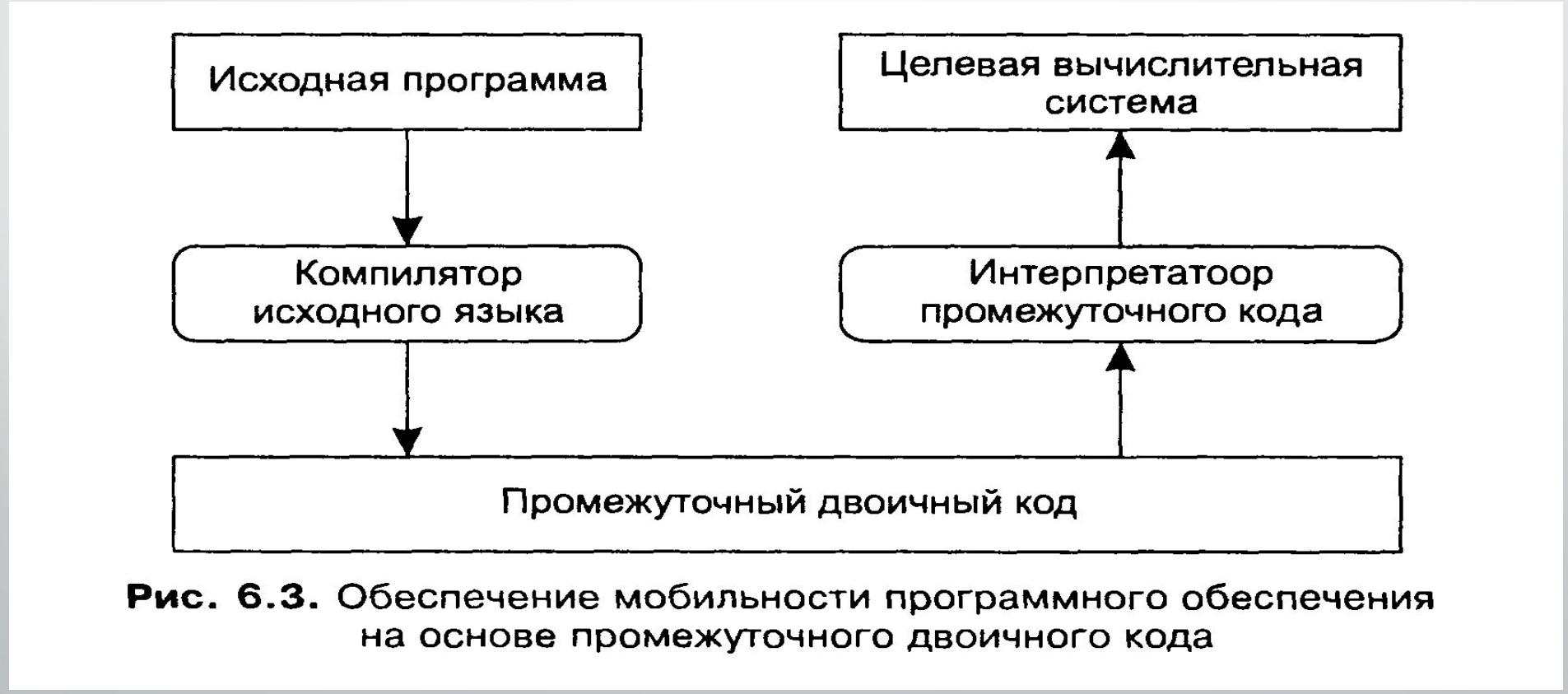

Рис. 6.3. Обеспечение мобильности программного обеспечения на основе промежуточного двоичного кода

Преимущества и недостатки переносимости программ

Преимущества:

- Расширение рынка сбыта ПО
- Снижение зависимости от изменения архитектуры вычислительных систем
- Снижение зависимости коммерческого успеха создаваемого ПО от позиции на рынке производителей ОС и систем программирования

Недостатки:

- Потеря эффективности (работа с драйверами аппаратуры, прямое обращение к функциям ОС – самые эффективные средства работы)
- Увеличение объема кода программ
- Снижение скорости выполнения при интерпретации по сравнению с откомпилированным кодом

Разработка приложений в архитектуре «клиент-сервер»

Две логически цельные составляющие прикладной программы:

- Обеспечение «нижнего уровня» работы приложения, отвечающее за методы хранения, резервирования, доступа и разделения данных
- обеспечение «верхнего уровня» работы приложения, отвечающее за логику обработки данных и интерфейс пользователя

Структура приложения в архитектуре «файл - сервер»

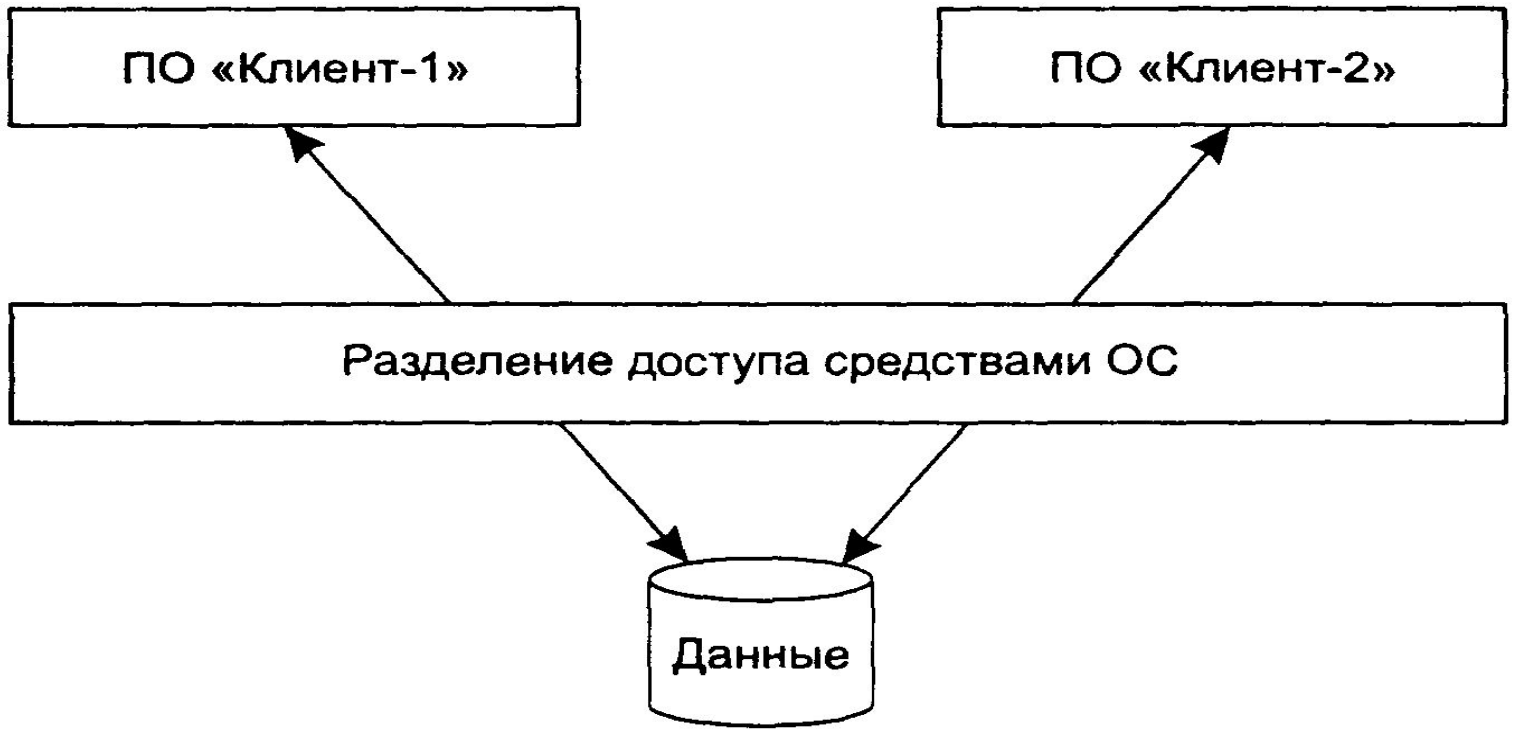

Рис. 6.4. Схема доступа к данным для приложений типа «файл-сервер»

Структура приложения в архитектуре «клиент - сервер»

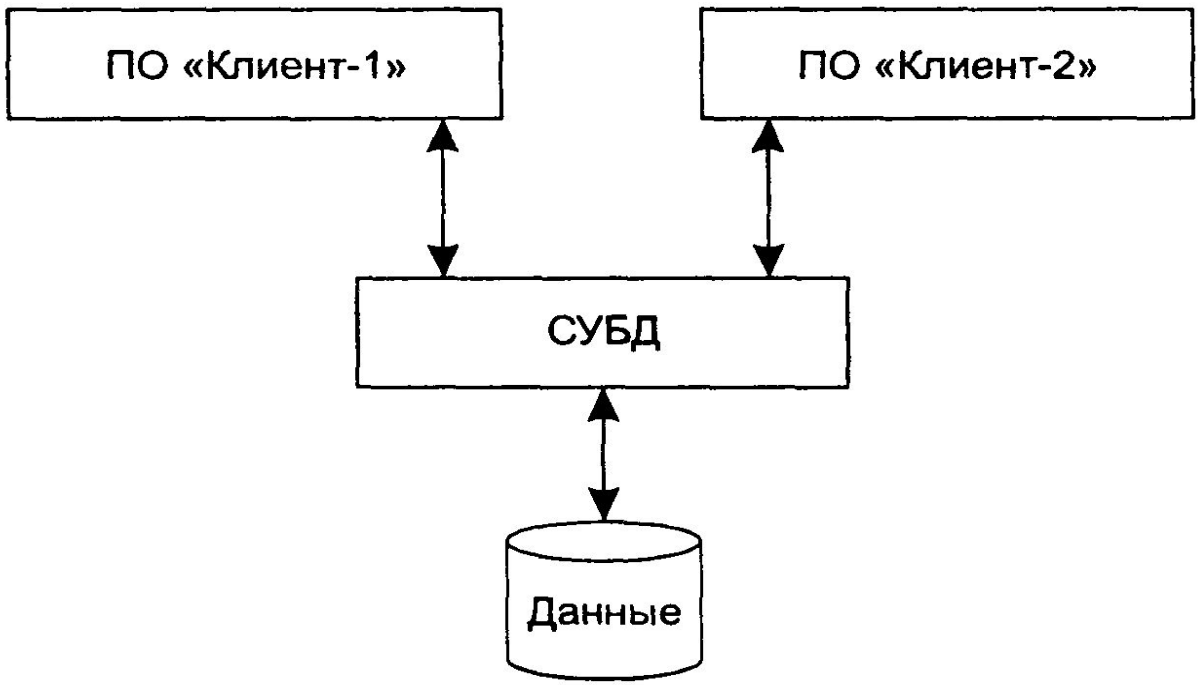

Рис. 6.5. Схема доступа к данным для приложений в архитектуре «клиент-сервер»

Типы СУБД

- Иерархические
- Сетевые
- Реляционные
- Объектно-ориентированные

Современные серверы данных

- Реляционные СУБД (Sybase, Microsoft SQL Server, Oracle, DB2)
- Механизм запросов (язык запросов SQL – Structured Query Language)

Язык запросов к данным

Основные операции над данными на сервере:

- Выборка (SELECT)
- Добавление (INSERT)
- Обновление (UPDATE)
- Удаление (DELETE)

Механизм транзакций

Основной недостаток – интерпретируемость (каждый раз производится распознавание запроса и проверка его правильности)

Технологии доступа к серверам данных

- Динамически загружаемые библиотеки
- Универсальный интерфейс взаимодействия клиентской и серверной части - интерфейс ODBC (Open DataBase Connectivity)

Создание приложений в архитектуре «клиент-сервер»

Преимущества использования:

- Разработчики приложений в архитектуре «клиент-сервер» избавлены от необходимости самостоятельно создавать средства для хранения данных, разделения доступа к ним, защиты и резервного копирования данных – все эти функции берет на себя СУБД
- СУБД обеспечивает высоконадежные механизмы разделения доступа к данным и защиты их от несанкционированного доступа, удовлетворяющие общепризнанным стандартам
- Все функции по управлению данными выполняются на сервере данных, что снижает требования к вычислительным ресурсам рабочих станций, на которых выполняются клиентские приложения
- Для обмена данными между клиентскими приложениями и сервером данных через сеть передаются не все данные, а только запросы клиента и ответы сервера, что снижает нагрузку на сеть
- При необходимости увеличить число клиентов в сети достаточно включить в сеть новые рабочие станции, увеличить мощность сервера и пропускную способность сети, но нет необходимости обновлять ПО и аппаратуру существующих рабочих станций

Недостатки архитектуры «клиент - сервер»

- Функции управления возложены на сервер данных, но обработка данных осуществляется по-прежнему клиентскими приложениями, что не позволяет существенно снизить требования к ним, если логика системы предусматривает достаточно сложные манипуляции с данными
- При необходимости изменить или дополнить логику обработки данных надо выполнить обновление клиентских приложений на всех рабочих местах, что может быть достаточно трудоемко
- Если необходимо изменить только внешний вид интерфейса пользователя (отображение данных), но оставить неизменной логику обработки данных, то чаще всего требуется заново создать и установить на рабочем месте новый вариант клиентской части системы
- При использовании мощной промышленной СУБД требуется наличие лицензии на подключение к СУБД каждого рабочего места, где установлена клиентская часть ПО

Разработка программ в многоуровневой архитектуре