детельствуют о том, что повышение курса доллара на 1 % приводит к возрастанию ВВП, экспорта, импорта, потребления, сбережений и к уменьшению величины утечки капитала за рубеж. При этом наиболее сильное влияние оказывается на увеличение экспорта. Повышение величины процентной ставки на 1 % практически не оказывает влияния на величину всех приведенных макроэкономических параметров, кроме утечки капитала за рубеж, которая возрастает на 0,38 %. Эффект увеличения налоговой ставки на 1 % ведет к снижению всех приведенных макроэкономических показателей, кроме увеличения утечки капитала за рубеж, которая составляет 1,66 %. Таким образом, данные таблицы 6 дают количественную оценку положительного влияния на экономическое развитие России при повышении курса доллара и снижения налоговой ставки, что обусловливает увеличение ВВП и уменьшение величины утечки капитала за рубеж.

На основании полученных результатов исследования можно сделать следующие выводы.

1. В российской экономике в отличие от экономики США не принимаются эффективные меры для достижения потенциальной (максимально возможной) величины ВВП.

2. Нереализованные возможности повышения темпов экономического роста России, вызванные отличием фактических величин денежной массы от их оптимальных значений, оцениваются: в 2000 г. – 22,6 млрд. долл., в 2001 г. – 11,6 млрд. долл., в 2002 г. – 7,5 млрд. долл., то есть общая оценка за 3 года превышает 41,7 млрд. долл. В связи с тем, что меры по оптимизации величины денежной массы практически не требуют дополнительных затрат, то они являются экономически безусловно оправданными даже в случае, если в реальных условиях потенциальные возможности повышения темпов экономического роста будут достигнуты не полностью.

3. На основе использования МВМ-О может быть сделан важный шаг в направлении информационной поддержки управленческих решений на макроэкономическом уровне.

4. В дальнейшем целесообразно продолжить развитие моделей типа МВМ-О для расширения их возможностей по прогнозированию российской экономики.

Список литературы

1. Узяков М.Н., Ефимов В.М., Серебряков Г.Р. и др. Макроэкономическая политика и ее последствия (возможности анализа и обоснования с помощью экономико-математического инструмента)// Проблемы прогнозирования. – 2003. - № 4.

2. Масюков В.А., Масюков В.В. Балансовая модель прогнозирования параметров макроэкономического развития. // Сб. науч. тр.: Теоретические проблемы управления производством и капиталом. – Тверь: МЭСИ. – 2001.

3. Масюков В.А., Матылина Л.В. Макроэкономическая балансовая модель для стран с элементами переходной экономики. //Программные продукты и системы. – 2003.- № 2.

4. Масюков В.А. Математическое описание макроэкономических оптимизационных балансовых моделей (МВМ-О) и их программные демонстрационные версии применительно к экономике России и США. // http://homepages.tversu.ru/~p000133, 2003.

5. Волков В.Н. Итоги социально-экономического развития России в 2000 – 2002 годах и прогнозируемые параметры в 2003 году. //Деньги и кредит. – 2003 - № 3.

6. Садыков Ф.К. Межбюджетные отношения в РФ: оценка состояния и перспективы развития. //Банковское дело. – 2003 - № 1.

7. Российский статистический ежегодник: [Сборник]. – М.: Госкомстат России, 2001.

8. Бюджетная система Российской Федерации // www.budgetrf.ru

9. Экономическая экспертная группа // www.eeg.ru

КОММУНИКАЦИОННАЯ БИБЛИОТЕКА LβCOMM

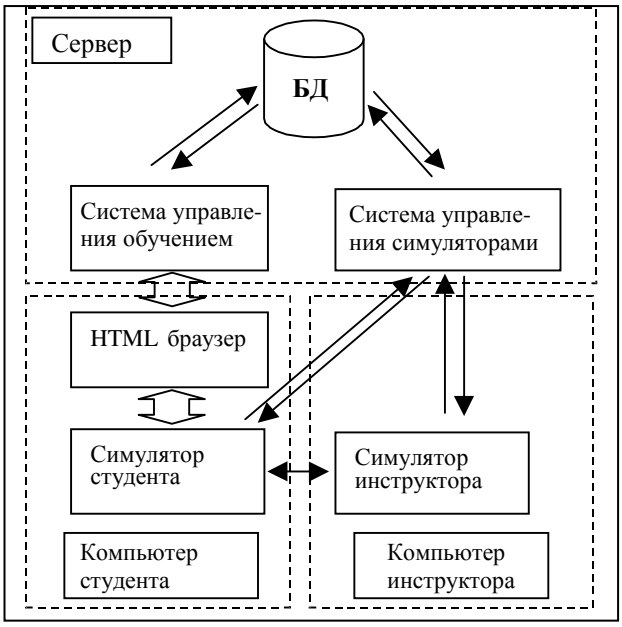
А.М. Альбертьян

Библиотека **LβComm** позволяет осуществлять передачу сообщений по IP сети с возможностью подтверждения приема и гарантированной доставкой.

Библиотека разрабатывалась в процессе работы над проектом по созданию распределенной обучающей системы в соответствии с рекомендациями стандарта SCORM (*Sharable courseware object reference model*) [1]. Рассмотрим общее устройство и функционирование обучающей системы. Упрощенная схема обучающей системы представлена на рисунке.

В качестве каналов связи между компонентами обучающей системы предусмотрено использование любых физических линий связи, на которых возможна работа протоколов из семейства TCP/IP.

Из рисунка видно, что обучающая система состоит из трех основных частей, которые функционируют на стороне сервера, на стороне обучаемого (далее – студента) и на стороне инструктора. Взаимодействия системы управления обучением (*Learning management system* – LMS) и приложения студента (обычного HTML-браузера) регламентированы стандартом распределенного обуче-



ния SCORM и осуществляются в рамках протокола HTTP. LMS реализована с использованием языка *Java* и *Java script*. В БД хранятся учетные записи студентов с информацией о них и об уровне их текущей подготовки.

Главным отличием от стандартной обучающей системы, реализованной в соответствии со SCORM, стало введение в состав системы симулятора, работающего в режиме реального времени. Важной особенностью симулятора является возможность работы с инструктором, который может вмешиваться в процесс симуляции на компьютере студента. В системе предусмотрена возможность работы с несколькими студентами и несколькими инструкторами.

Серверная часть симулятора – система управления симуляторами (*Learning simulation system* – LSS) предназначена для авторизации и регистрации инструкторов и студентов в системе и предоставлении свободного инструктора по запросу со стороны студента. То есть при запуске симулятора инструктора происходит его авторизация и в случае успеха – добавление в список доступных на текущий момент инструкторов (*Active List* – AL).

При желании студента войти во взаимодействие с инструктором производится его авторизация LSS и поиск свободного инструктора в AL. Инструктор извещается о попытке соединения со стороны студента, и студенту возвращаются его координаты (IP адрес и порт). Кроме того, он помечается в AL как ожидающий подтверждения со стороны симулятора студента. При подтверждении подобного запроса в течение некоторого временного интервала происходит установка соединения между симуляторами, и инструктор помечается в AL как занятый. LSS периодически проверяет наличие связи с симуляторами студента и инструктора и в случае невозможности связи в течении некоторого времени посылает предупреждение другой стороне и отражает изменения в AL.

При нормальном завершении работы любого из приложений симулятора (на стороне студента, инструктора или LSS) посылаются предупреждения всем остальным компонентам. Аварийное завершение любой части симулятора определяется по отсутствию запросов или ответов на запросы в течение заданного временного интервала.

Так как симулятор был реализован на языке C^{++}, то с целью обеспечения необходимой скорости работы возникла проблема взаимодействия компонентов симулятора: приложений инструктора, студента и LSS. Так родилась идея создания некоторой универсальной коммуникационной компоненты, которая, находясь на сторонах всех клиентов, обеспечивала бы их надежное взаимодействие. Поскольку в качестве основного протокола LMS используется HTTP, было решено использовать протоколы из семейства TCP/IP.

Большая часть информационного обмена LSS происходит в режиме *запрос-ответ* с нерегулярным обменом короткими информационными блоками, и было бы удобно в качестве основного протокола обмена информацией использовать протокол UDP. Но протокол UDP не обеспечивает подтверждения передачи данных и не гарантирует доставку сообщений. Поэтому возникла необходимость в реализации этих функций непосредственно в коммуникационной компоненте.

Особенности и основные алгоритмы работы

Коммуникационная компонента была названа **Lβ(LSS)Comm** и была реализована на *Watcom* C^{++} (версия *11.0b*) с использованием встроенного ассемблера в критичных ко времени выполнения местах. Полученная библиотека динамической компоновки (*LbComm.dll*) может функционировать под управлением ОС из семейства *MS Windows* и любых других, где реализована полноценная поддержка *Win32 API* и установлена библиотека *Windows Sockets DLL* версии 1.1 и выше.

Библиотека поддерживает работу с несколькими программными потоками и реализует асинхронную передачу и прием сообщений переменной длины с возможностью подтверждения доставки и многократными попытками передачи при нарушениях связи. Функции библиотеки позволяют гибко настраивать все временные интервалы и количество повторов попыток передачи. Результат выполнения всех длительных операций можно проверить с помощью специального номера операции (*handle*) или с помощью обратного вызова (*callback*-функции). При использовании *callback*-функций структуры данных, связанные с операцией, уничтожаются сразу после завершения функции. При использовании только *handle* невостребованные в течение заданного времени результаты

операций или принятые сообщения удаляются автоматически.

В библиотеке есть функции гибкого управления выдачей диагностических сообщений. Также поддерживается отладка связи с возможностью выдачи информации на консоль или в файл.

Главной особенностью библиотеки является способ передачи сообщений и подтверждений о приеме. После вызова функции отправки сообщения при необходимости происходит асинхронное преобразование адреса назначения. Адреса кэшируются библиотекой для ускорения последующих преобразований.

Перед отправкой все сообщения разбиваются на блоки фиксированной длины (в данной версии – 512 байтов). Для каждого из этих блоков заводится свой счетчик времени и количества повторов. После инициализации отправки по мере освобождения буфера *Winsock* блоки разных сообщений порциями отправляются адресатам. При отправке к блоку добавляется короткий заголовок (см. таблицу), содержащий сигнатуру, информацию о сообщении, номер блока и 32-разрядную CRC блока данных и основной части заголовка.

Таблица

Формат блока данных LβComm

Смещение (байтов)	Значение
+0	сигнатура
+4	CRC32 (начиная с +8)
+8	***Id-сообщения***
+12	***длина сообщения***
+16	***номер блока***
+20	***512 байтов данных***

Самым последним передается пустой блок-признак конца сообщения (*End of transmission* - EOT). После отправки очередного блока в соответствующий ему счетчик заносится определенное значение тайм-аута (он активизируется). По событиям системного таймера происходит уменьшение значений во всех активных счетчиках. В цикле передачи происходит последовательная проверка счетчиков для каждого блока и, если обнаружено нулевое значение счетчика, идет (повторная) отправка блока. Также предпринимаются агрессивные попытки передачи блоков с истекающим значением тайм-аута. Количество повторных отправлений для каждого блока ограничено и подсчитывается независимо. Также есть возможность отправки сообщения без подтверждения – в этом случае блок сразу маркируется как корректно отправленный и не формируется блок EOT.

При получении блока на приемной стороне проверяется его корректность с помощью CRC32 и содержимого полей заголовка. Если обнаруживается принадлежность блока к одному из незаконченных сообщений, он добавляется на свое место в последовательность блоков данного сообщения. Если блок принадлежит новому сообщению, то инициируется прием нового сообщения – создается буфер и все необходимые для приема сообщения структуры данных. Для обеспечения надежности функционирования системы в целом и защиты от переполнения памяти устанавливаются ограничения на размер и количество одновременно принимаемых сообщений. После приема корректного блока данных, если для данного сообщения запрошено подтверждение, формируется запрос на отправку передающей стороне короткого блока данных с подтверждением приема. Блоки подтверждения приема пересылаются в общем цикле отправки сообщений.

После получения корректного подтверждения приема блока он помечается как отправленный. Передача сообщения завершается при подтверждении приема всего сообщения или по исчерпании счетчика повторных передач для одного или нескольких блоков. Передача прекращается по истечении тайм-аута на отправку блока или приема подтверждения передачи (если все блоки уже отправлены).

Прием сообщения считается завершенным при корректном приеме всех блоков сообщения. При получении каждого следующего блока сбрасывается счетчик времени для следующих за ним еще не принятых блоков. При достижении счетчиком времени нуля формируется запрос на повторную передачу данного блока. Количество запросов для каждого блока ограничено и подсчитывается независимо. Прием прекращается по исчерпании счетчика повторов для одного или нескольких блоков, и сообщение считается непринятым по истечении тайм-аута на прием блока.

Данные алгоритмы работы позволяют обеспечить надежность и контроль доставки сообщений без излишнего увеличения сложности и низких накладных расходов на передачу данных, свойственных протоколу UDP. Небольшой размер отдельных блоков данных и "агрессивность" попыток передачи потерянных блоков позволяют использовать данную библиотеку на очень плохих линиях связи, например, модемных (даже полудуплексных) соединениях.

Описание основных функций

Приведем краткое описание основных функций библиотеки. Для каждого потока основной программы функции работают независимо. В состав библиотеки входят различные дополнительные и вспомогательные функции, функции управления тайм-аутами, управления отладкой с выдачей отладочной информации в файл и/или на консоль и т.д.

int CommGetStatus(COMMHANDLE handle);

Проверка статуса операции (прием, передача, преобразование адреса).

Возможные коды возврата этой функции:

COMM_SUCCESS – операция выполнена успешно,
COMM_FAILURE – ошибка выполнения,
COMM_WAITING – операция все еще выполняется.

*int CommResolveAddr(char const *addr, char *host, struct sockaddr_in *sin);*

Преобразование строки адреса (адрес:порт) из *addr* в структуру *sin*, а выделенный из строки адрес (без порта) возвращается в буфер *host* (должен иметь размер *COMM_MAX_HOSTNAME + 1* или быть равен *NULL*).

Возможные коды возврата этой функции:
COMM_SUCCESS – операция выполнена успешно,
COMM_FAILURE – ошибка выполнения,
COMM_WAITING – операция все еще выполняется; можно проверить статус, используя *handle* COMM_ADDR_HANDLE.

*int CommResolveAddrE(char const *addr, char *host, struct sockaddr_in *sin, COMMEVENT handler);*

То же, что и *CommResolveAddr( )*, только по завершении операции после возврата кода *COMM_WAITING* происходит вызов функции *handler* типа *COMMEVENT*, определенного как:
*typedef void (*STDCALL COMMEVENT)(int result, void *info, unsigned infolen);*

Ей передается адрес заполненной структуры *struct in_addr* в *info* и *sizeof(struct in_addr)* в *infolen*.

*bool CommBind(char const *port);*

Привязка библиотеки к определенному порту.

bool CommUnbind( );

Освобождение порта. Возвращает *false* при ошибке.

*COMMHANDLE CommSend(char const *addr, void const *data, unsigned sz);*

Передать сообщение *data*, длиной *sz* по адресу *addr* (адрес:порт). Возвращает *handle* операции или значение *COMM_INVALID_HANDLE*.

*COMMHANDLE CommSendE(char const *addr, void const *data, unsigned sz, COMMEVENT handler, dword id);*

То же, что и *CommSend( )*, только по завершении операции происходит вызов функции *handler* (см. описание функции *CommResolveAddrE( )*). Ей передается адрес заполненной структуры *struct sockaddr_in* с преобразованным адресом в *info* и *sizeof(struct sockaddr_in)* в *infolen*, если *id* равняется нулю, или значение *id*, если оно не равняется нулю.

*COMMHANDLE CommSendSock(struct sockaddr_in const *sin, void const *data, unsigned sz);*

*COMMHANDLE CommSendSockE(struct sockaddr_in const *sin, void const *data, unsigned sz, COMMEVENT handler, dword id);*

То же самое, что *CommSend( )* и *CommSendE( )*, только вместо строки адреса используется уже преобразованный адрес из структуры *sin*.

COMMEVENT CommSetRecvCallBack(COMMEVENT handler);

Устанавливает функцию *handler* (см. описание функции *CommResolveAddrE( )*), которая будет вызываться при получении сообщения. Адрес сообщения передается функции *handler* в *info*, а его длина в *infolen*. По адресу *(unsigned char)info + infolen* создается структура *struct sockaddr_in* в которой передается адрес и порт отправителя сообщения. Для отмены обратного вызова нужно задать значение *handler,* равное *NULL*. Функцией возвращается предыдущее значение *handler*.

COMMHANDLE CommRecvNextHandle( );

Возвращает *handle* принятого сообщения или значение *COMM_INVALID_HANDLE* при отсутствии сообщений.

*int CommRecvPeek(COMMHANDLE h, void *buf, unsigned sz, struct sockaddr_in *sin);*

Копирует заданное значением *handle h* сообщение из очереди сообщений в буфер *buf* и сохраняет адрес и порт отправителя сообщения в структуре *sin*. Возвращает *COMM_FAILURE* при ошибке или длину принятого сообщения (в виде беззнакового целого). Сообщение остается в очереди.

*int CommRecv(COMMHANDLE h, void *buf, unsigned sz, struct sockaddr_in *sin);*

Копирует заданное значением *handle h* сообщение из очереди сообщений в буфер *buf* и сохраняет адрес и порт отправителя сообщения в структуре *sin*. Возвращает *COMM_FAILURE* при ошибке или длину принятого сообщения (в виде беззнакового целого). Сообщение удаляется из очереди.

*int CommRecvNextPeek(COMMHANDLE *h, void *buf, unsigned sz, struct sockaddr_in *sin);*

*int CommRecvNext(void *buf, unsigned sz, struct sockaddr_in *sin);*

То же самое, что *CommRecvPeek( )* и *CommRecv( )*, только *handle* сообщения определяется вызовом функции *CommRecvNextHandle( )*. В случае использования *CommRecvNextPeek( )* значение *handle* копируется в *h*.

В заключение отметим, что библиотека **LβComm** представляет собой гибкое и простое в использовании средство для ОС фирмы *Microsoft*, которое позволяет осуществлять передачу сообщений по IP сети с возможностью подтверждения приема и гарантированной доставкой.

Список литературы

1. Sharable Courseware Object Reference Model SCORM version 1.1. http://www.adlnet.org

2. Семенов Ю.А. Протоколы и ресурсы Internet. - М.: Радио и связь, 1996.

3. Тод Леммл и др. Учебное руководство для специалистов MCSE: TCP/IP. - М.: Изд-во "Лори", 1997.

4. Internet Protocol, RFC-791.

5. User Datagram Protocol, RFC-768.

6. Platform SDK / Win32 SDK help. http://msdn.microsoft.com