УДК 004.6

СИСТЕМА ТЕСТИРОВАНИЯ ЗАДАЧ

В.Э. Бойко, Д.Ю. Мелеш, *2 курс*
Научный руководитель – ***Н.В. Силаев,*** *доцент*
Брестский государственный университет им. А.С. Пушкина

Система тестирования состоит из двух взаимно зависимых программ. Это – **серверная часть** и **клиентская часть**. Связь в сети между ними построена на основе сокетов (Socket).

Серверная часть отвечает за регистрацию групп пользователей, добавление/удаление отдельных пользователей, добавление/удаление тем задач, добавление/удаление отдельных задач и наборов тестов к ним, за инициализацию процесса тестирования и многое другое. Естественно, что эта часть может быть доступна только администратору тестирования и устанавливается на особый компьютер.

Клиентская часть доступна студентам. Здесь происходит их авторизация зарегистрированных администратором пользователей, выбор задач, с возможностью ознакомления с условием. Используя возможности клиентской части комплекса, пользователь может отправлять на тестирование реализованный им код.

Интуитивно понятный **интерфейс** программы строился таким образом, чтобы администратору было удобно управлять различными параметрами и свойствами. Были использованы такие технологии как Drag&Drop (перетаскивание записей из одной базы данных в другую), всплывающие меню над многими компонентами для ускорения и упрощения работы. В дальнейшем планируется введение мастера для добавления групп и задач из файлов, с целью еще более гибкого и оперативного обслуживания баз данных.

В находящейся в настоящее время в опытной эксплуатации версии комплекса сделаны некоторые улучшения для **многопользовательского** использования программы, заключающиеся в основном в том, что реализована возможность раздельного хранения базы данных, журналов тестирования, и других настроек. Имеется возможность гибко менять файлы со всеми настройками. Эта возможность помогает, в частности, при отборе задач для занятий типа «Лабораторное задание» и «Экзамен», т.е. можно один раз организовывать нужные наборы параметров, а в последующем их просто загружать, что избавляет администратора от рутинной работы (поиска из огромного числа нужных задач) и делает его работу более технологичной.

В нам комплексе тестирования реализованы два режима работы приложений: «Экзамен» и «Лабораторная», что дает возможность использовать приложение в разных условиях.

Несколько слов хотелось бы сказать о построении **кода**. Длительное время мы решали вопрос о том, чему отдать предпочтение: использованию библиотек dll или средств обычных модулей. Дело в том, что преимущества библиотеки в том, что она загружается только при вызове методов из кода программы. Это освобождает от использования лишней памяти, но при этом, при первой загрузке, вызов процедур и функций dll происходит не с очень высокой скоростью, что при передаче данных по сети может вызвать определенные проблемы, которые могут привести к потере информации, следствием чего является неработоспособность программы. Помимо этого «минусом» библиотеки является довольно непростая передача параметров, связанная с ОС (т.е. очень плохо, допустим, передавать такой тип данных как String: вместо него нужно использовать PChar). Сложным также является, по понятным причинам, отладка библиотек. Всех этих минусов нет в модулях, но и тут не все гладко. При добавлении модуля к проекту размер его сильно увеличивается, что ведет также к дополнительным расходам оперативной памяти, но нужно учитывать и то, что мы использовали только собственные модули, следовательно, все методы в них были затребованы, в связи с чем «лишней памяти» не требуется. Не будем забывать и то, что мы живем в XXI веке и программисты уже не следят за лишними килобайтами и даже мегабайтами памяти. Учитывая приведенные выше рассуждения, мы отдали предпочтение модулям.

Программный комплекс строился по технологии «сверху вниз», т.е. вначале писались элементарные структуры, а потом на основе их создавались более сложные. Это позволило уменьшить код модуля, оптимизировать его, и, самое главное, учитывая то, что элементарные структуры ис-

пользуются не раз и не два, а довольно часто, то при исправлении ошибки на низком уровне, она автоматически исправляется во всех случаях ее употребления на высоком уровне и повышало показатель повторного использования отлаженного кода. Все это делает любую программу, пользующуюся указанной технологией, более простой в отладке и возможной модификации.

Очень полезным моментом оказалось использование в проекте обработка исключительных ситуаций, это не только помогло поддерживать неплохую работоспособность программы, а также более точно локализовать существующие ошибки. Обработка исключительных ситуаций строилась также с учетом необходимости выдачи более информативных сообщений об ошибках пользователю, с тем, чтобы он самостоятельно корректировать свои последующие действия, с целью выбора наиболее подходящих и корректных. Мы убеждены, что пользователь не получит никакой полезной информации, если по ходу работы программы на экран будет выведен адрес памяти, где произошел сбой. Информативное сообщение, вызванное некорректными действиями пользователя, должно содержать сведения о типе ошибки и возможных причинах ее возникновения. Нами планируется реализовать журнал ошибок, который можно время от времени просматривать и модифицировать код. Осуществить такую процедуру не так уж и сложно. Мы предлагаем создать бета-версию программы, которая будет вести журнал всех происходящих событий и при возникновении ошибки сохранять в файл последние из них. Это даст возможность отследить, где именно произошла ошибка.

По нашему мнению, самой интересной и сложной проблемой в тестирующей программе является собственно фаза **тестирования задачи** и уяснения базы ее построения. Сервер получает от клиента код задачи (хотя в нашей системе это не единственный вариант), первым делом его нужно скомпилировать , используя нужный компилятор. (Обычно это консольное приложение, которое нужно запускать с некоторыми ключами). Для этого мы использовали WinApi функцию CreateProccess , но ее не всегда хватает для этих целей, кроме того, она является лишь маленькой частью всей процедуры. Дополнительно использовались такое средство, как поименованные каналы Pipes. Через них осуществляется связь с консольным приложением (т.к. оно имеет стандартный тип ввода/вывода). В случае удачной компиляции, сервер получает exe-файл, затем, используя те же методы, тестирует задачу, но уже не компилируя ее каждый раз, экономя на этом время (иногда удавалось протестировать несколько задач от разных клиентов за 1-2 секунды(!), учитывая время на передачу задачи по сети и это еще на первой версии программы).

Важным моментом также является и построение **связи по сети**. Как уже упоминалось строится она на основе сокетов. Мы решили использовать не один, а несколько портов одновременно (их же 65000), это должно дать выигрыш в производительности, т.к., допустим, регистрация клиентов происходит по одному порту, а тестирование задачи по другому, что дает более распределенную нагрузку на каждый, особо не перегружая их. По большей части, как и все клиент-серверные приложения, наши программы обмениваются обычными строками, для передачи файла используются файловые потоки.

Как отмечалось выше, комплекс наших программ проходит экспериментальную проверку в ходе тестирования решения лабораторных задач на математическом факультете БрГУ.