Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
ДАЛЬНЕВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Школа естественных наук
Кафедра информатики, математического и компьютерного моделирования

Храпченков Пётр Фёдорович

Модуль контролируемого исполнения программ Spawner

ДИПЛОМНАЯ РАБОТА

по основной образовательной программе подготовки специалистов по специальности 010501.65 – прикладная математика и информатика

Студент ОЗО _____
(подпись)

Руководитель ВКР – ст. преп.
(должность)

_____ А.С. Кленин
(подпись) (и.о.ф)

«____»_____ 2014г.

Защищена в ГАК с оценкой _____

Секретарь ГАК

_____ _____
(подпись) (и.о.фамилия)

«____»_____ 2014г.

«Допустить к защите»

Заведующий кафедрой – д.ф.-м.н., проф.
(ученое звание, должность)

_____ А. Ю. Чеботарев
(подпись) (и.о.ф.)

«____»_____ 2014г.

г. Владивосток

2014

Содержание

Аннотация

Модуль Spawner является частью автоматической проверяющей системы CATS и используется для запуска программ с ограничениями на использование системных ресурсов и получения информации об использованных системных ресурсов программой, а так же информации о ходе исполнения.

Рассмотрены подходы к решению проблемы и осуществлена реализация.

1. Введение

1.1. Глоссарий

Системные ресурсы – Набор системных ресурсов включает в себя время исполнения программы в пользовательском режиме, физическое время исполнения, объем выделенной памяти, объем записанной на диск информации.

Физическое время исполнения – абсолютное время, определяемое глобальным системным таймером.

Уровень загрузки – величина, отображающая отношение времени в пользовательском режиме к физическому времени.

Время простоя – время, в течении которого уровень загрузки программы находился не выше установленного минимального.

Механизм песочницы(англ. Sandboxing) – в компьютерной безопасности защитный механизм для изолированного исполнения программ. Часто используется для запуска непроверенного кода или сомнительных программ от неизвестных разработчиков, поставщиков, подозрительных пользователей или веб сайтов.

JOB_OBJECT – это объект, способный иметь имя, настройки безопасности, который контролирует атрибуты ассоциированных с ними процессов. Операции, применяемые к ним, влияют на все процессы, включенные в него. Так, например можно ограничить память, приоритет процессов, а так же завершить их все.

1.2. Описание предметной области

1.2.1. Соревнования по программированию

На сегодняшний день в России и мире проводится множество соревнований по программированию, включая известный командный чемпионат мира по программированию среди студентов высших учебных заведений ACM[1], а так же олимпиады школьников по информатике. Подобные соревнования получили широкое распространение и помогают относитель-

но объективно оценить уровень знаний и умений участников в области программирования[2]. В рамках соревнований участникам предлагается набор задач, который они должны решить за определенное время, не выходя за какие-то заданные условиями соревнований ограничения. Каждому участнику или команде участников предоставляется компьютер с предустановленной средой программного обеспечения, позволяющей сразу же приступить к решению задач.

При проведении соревнования по программированию, от организатора требуется разработать пакет заданий, состоящий как из условий самих задач, так и набора тестов и эталонного решения. Когда участник разработал программу и считает, что она удовлетворяет условиям задачи, он предоставляет данное решение для проверки организаторам, где, в свою очередь, проходит тестирование и сравнение результата работы программы участника с результатами работы эталонного решения. После этого делается вывод о правильности предложенного на проверку решения.

Для того, чтобы упростить организаторам работу, а участникам их участие, стали применяться так называемые автоматизированные системы организации соревнований. Такие системы обычно состоят из нескольких частей, содержащих в себе интерфейс для наблюдения за ходом соревнования, интерфейс для отправки задач и интерфейс для их проверки.

1.2.2. Система CATS

Одной из таких систем является автоматическая система организации соревнований по программированию CATS[3][4] (Рис. 1). Система разрабатывалась на базе ДВФУ и успешно применяется при проведении соревнований начиная от школьных олимпиад и заканчивая четвертьфиналом чемпионата мира по программированию ACM ICPC. История системы начинается с 2002 года и продолжается до сих пор. За это время на базе CATS было проведено огромное количество соревнований, система неоднократно улучшалась и дополнялась[4][5][6][7][8].

На текущий момент система включает в себя такие модули как веб-сервер, предоставляющий пользователям веб-интерфейс к CATS, базу данных, а так же модуль judge[4](подробнее см. раздел Архитектура).

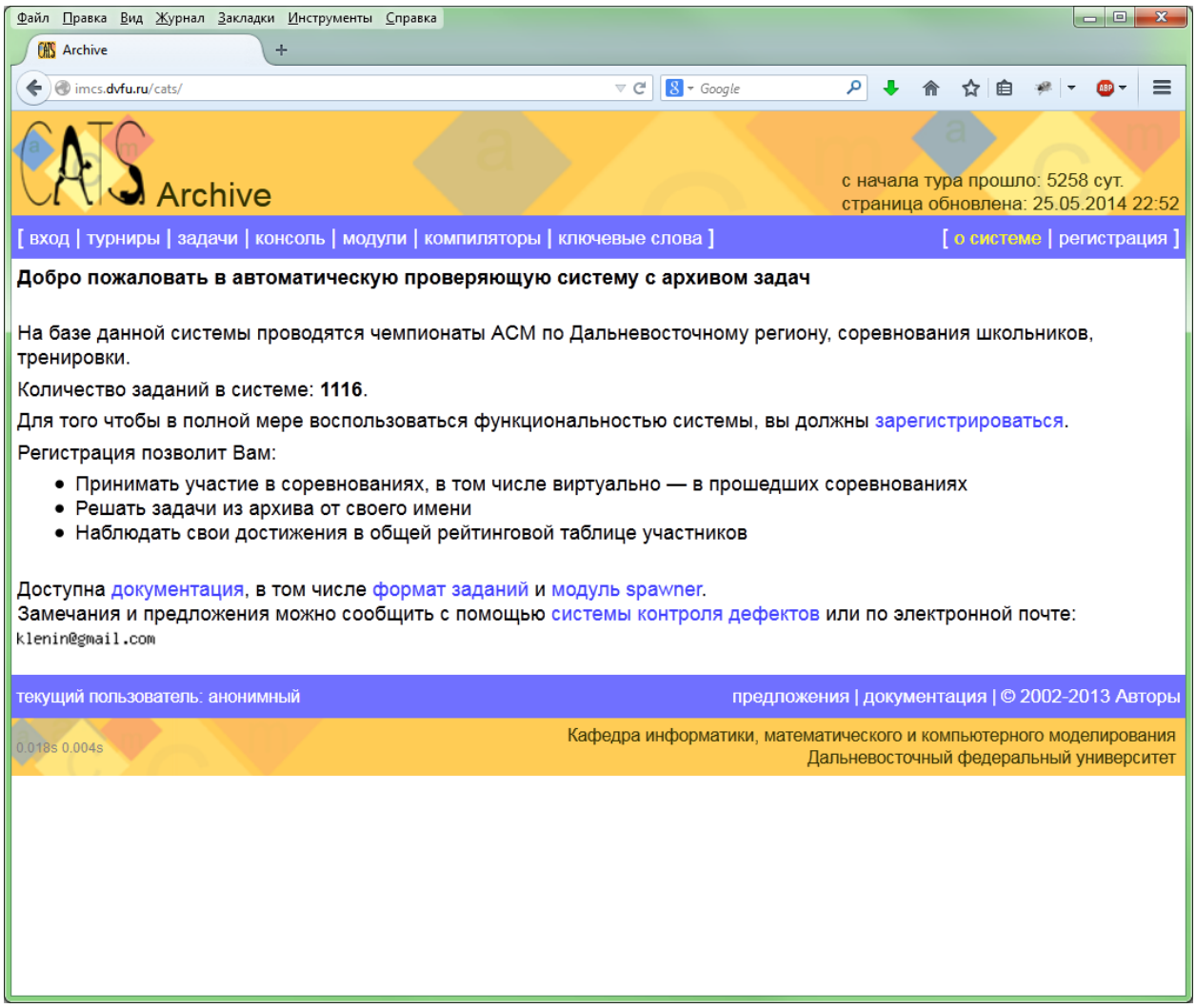

Рис. 1: Веб-интерфейс системы

1.2.3. Веб-интерфейс

Веб-интерфейс предоставляет возможность участвовать в доступных турнирах, просматривать задачи, отправлять соответствующие решения, выполненные в разрешенных системой (турниром) средах разработки, следить за ходом их тестирования, следить за ходом турнира и др. Для организаторов, в свою очередь, веб-интерфейс позволяет создавать турниры, добавлять в них задачи, следить за ходом турнира и др.

1.2.4. База данных

Связь между веб интерфейсом и модулем judge осуществляется через базу данных. База данных хранит информацию о существующих в системе

турнирах, участниках, задачах, решениях и пр.

Веб интерфейс и база данных являются кросс-платформенными составляющими системы и могут быть установлены на компьютеры под управлением различных операционных систем.

1.2.5. Модуль judge

Данный модуль отвечает за самую ответственную часть, иначе говоря за контролируемый запуск и тестирование программ решений участников. Во время исполнения программ от judge требуется чтобы они не нарушали набора ограничений на системные ресурсы, наборы правил безопасности, а, в случае некорректного хода работы, информация о произошедших ошибках должна быть корректно обработана и учтена. После чего результаты о ходе тестирования и запуска решения участника отправляются в базу данных.

1.3. Общая задача контроля исполнения

В общем случае данная задача является гораздо более обширной, чем будет рассмотрена в рамках данной работы, что, возможно, даст дальнейшее развитие модуля Spawner. На данный момент уже существует множество менее специфичных инструментов для ее решения. Для контролируемого исполнения программ используется механизм компьютерной безопасности песочницы[10]. Песочницы предоставляют жестко контролируемый набор системных ресурсов для заданной исполняемой программы – например, место на диске, в памяти, время исполнения и пр. Данная технология находит применение когда требуется исполнить неизвестную программу, которая может содержать потенциально опасный код. Помимо применения в системах организации соревнований технология получила широкое распространение в современных операционных системах, антивирусах, браузерах и пр[10]. В общем случае, для создания изолированной среды исполнения требуется большое количество ресурсов, что приводит к большой нагрузке на систему.

В рамках систем проведения соревнований, применение механизма пе-

сочницы необходимо для того, чтобы сбои в потенциально содержащих ошибки исполняемых решениях участников не могли привести к сбою системы в целом, а так же чтобы запретить доступ к таким системным ресурсам, как сеть, диск и др.

1.4. Модуль Spawner

Для задачи контролируемого исполнения в judge используется модуль Spawner, представленный консольной утилитой[11]. В модуле используются стандартные системные механизмы обеспечения наложения ограничений и создания изолированной среды исполнения.

Через консольный интерфейс задаются накладываемые ограничения а так же опции запуска исполняемой программы. После исполнения утилитой выдается отчёт, по которому можно получить представление об использовании ресурсов, нарушении ограничений и пр.

Консольный интерфейс был выбран для того, чтобы участники, да и просто все желающие, могли проверять свои решения на предмет соответствия условиям задачи.

До текущего момента модуль Spawner разрабатывался как независимая часть системы CATS и в полной мере отвечал необходимым требованиям.

Однако, консольная утилита Spawner оказалась несовместимой с 64 разрядными операционными системами Windows, в силу специфики программного кода. К тому же, модуль Spawner являясь исполняемым файлом Windows не позволяет проводить более тесную интеграцию веб-сервера с модулем judge из-за отсутствия возможности запуска системно-специфичного кода на других платформах.

К тому же аналогичные модули реализуют более расширенный функционал, например, по вводу дополнительных ограничений, таких как ограничение на время простоя. А вследствие отсутствия финальной версии исходного кода Spawner'а возможность дальнейших изменений и улучшений отсутствует.

Дополнительно, с внедрением новых видов интерактивных задач в соревнования Всероссийской олимпиады школьников по информатике[12] по-

явилась необходимость в реализации поддержки таких задач для CATS. Так же, на базе ДВФУ неоднократно проводились соревнования искусственного интеллекта, где также требуется одновременное контролируемое исполнение программ участников, а так же обмен данными между программами. Эти проблемы привели к постановке задачи множественного исполнения.

Таким образом, перечисленные выше пункты послужили мотивацией к написанию данной работы.

1.5. Неформальная постановка задачи

Требуется доработать модуль Spawner для выполнения следующих требований:

- Полная совместимость с предыдущей версией Spawner'а для первоначального внедрения в систему CATS.
- Проверка на наличие и корректная обработка исключений, возникших во время работы программы.
  - Обработка исключений программы.
  - Обработка исключений среды разработки.
- Ограничения на использование подконтрольной программой системных ресурсов:
  - память
  - время работы
  - время исполнения
  - запись в файлы
  - время простоя
- Вывод отчета в других форматах, включая как json, так и форматы отчетов других подобных модулей
- Кроссплатформенность

  - Возможность работы системы как в 32 так и в 64 битном окружении системы Windows.
  - Дальнейшая разработка совместимых версий для Linux и др.

- Работа со стандартными потоками - stdin, stdout и stderr, их перенаправление
- Корректный запуск программ из-под заданного пользователя.
- Множественное исполнение программ
- Перенаправление потоков ввода вывода
- Поддержка ролей запускаемых процессов
- Поддержка ролей запускаемых процессов
- Анализ и дополнение существующего формата входных аргументов, соответствующего указанным выше требованиям
- Интеграция доработок в систему CATS
- Возможность отдельного использования функционала утилиты в виде библиотеки.

1.6. Обзор существующих методов решения

1.6.1. Существующие решения песочниц

В настоящее время существует огромное количество доступных механизмов для создания изолированных окружений исполнения программ. Однако не все из них в своем чистом виде удовлетворяют задаче контроля исполнения программ участников соревнований по программированию. Рассмотрим существующие механизмы песочниц.

1. Виртуализация на уровне операционной системы — использование нескольких изолированных экземпляров пространства пользователя.

С точки зрения пользователя эти экземпляры полностью идентичны реальному серверу[13].

2. Исполнение, основанное на правилах — позволяет накладывать ограничения на программы и пользователей, задавая наборы правил для конкретной группы пользователей или программ. Ограничения касаются доступа к файлам, к реестру и т.п. Дополнительно могут использоваться инъекции в код программ, для контроля их поведения.

3. Виртуальная машина — полная эмуляция аппаратного обеспечения некоторой платформы. Программы исполняются внутри этого изолированного эмулированного окружения.

4. seccomp[14] — довольно простой, но мощный механизм обеспечения безопасности. Полностью ограничивает работу с системными дескрипторами давая возможность использовать ограниченный набор операций над ними. В случае попытки вызова запрещенных системных функций работа программы будет завершена. Таким образом для программы не создается отдельная виртуальная среда. Вместо этого программа полностью изолируется от системных ресурсов.

Основными критериями в выборе технологии являются: кроссплатформенность, нагрузка на систему, уровень изоляции.

Не смотря на строгие требования к производительности, задача контролируемого исполнения в рамках проведения соревнований по программированию может быть решена одним из указанных способов с высокой изоляцией и нагрузкой на систему. Так, в финале чемпионата мира по программированию ACM активно используется технология виртуальных машин. Дальнейшее изучение технологии песочниц, возможно, позволит развить данную работу и найти новые способы использования модуля Spawner.

1.7. Аналогичные модули или системы

В рамках данной работы было проведено изучение подобных систем организации соревнований по программированию, а так же модулей контролируемого исполнения.

Технология	Кроссплат-форменность	Нагрузка	Уровень изоляции	Примеры
Виртуализация на уровне операционной системы	Да	Средняя	Высокий	chroot, Parallels Virtuozzo Containers, FreeBSD Jail
Исполнение, основанное на правилах	Нет	Низкая	Средний	SELinux, Apparmor
Виртуальная машина	Да	Высокая	Очень высокий	VirtualBox, VMware
seccomp	Нет	Низкая	Высокий	—
Системы проведения соревнований	Да/Нет	Средняя	Средний	spawner, PCMS/Run

Таблица 1: Сравнение технологий песочницы

1.7.1. ejudge

ejudge[15] — это система для проведения различных мероприятий, в которых необходима автоматическая проверка программ. Система может применяться и применяется для проведения олимпиад, поддержки учебных курсов и т.д. Система предоставляет веб-интерфейс администратора и участника турнира, а так же доступ к серверам турниров из командной строки. Для контроля исполнения используется патч к ядру Linux. Модуль контроля исполнения программ представляет собой набор скриптов. Невозможно портирование скриптов на ОС Windows в силу их зависимости от системных функций.

1.7.2. PCMS2/Run

PCMS2(Programming Contest Management System)[16] Система контроля исполнения представляет собой консольную утилиту. В программе используется динамическая библиотека с вынесенными функциями. Используется во время проведения Всероссийского этапа ACM ICPC. Для контролируемого исполнения постоянно использует режим отладки. Так-

же следует отметить, что присутствует возможность создания и контроля исполнения процесса от другого пользователя, но для этого требуются особые привилегии.

1.7.3. Contester

Contester[17] — это система для проведения турниров и индивидуального решения задач по олимпиадному программированию (спортивному программированию). Система содержит условия задач - от легких до олимпиадных - и возможность проверки решений на большинстве современных языков: C++, Object Pascal, Java и языках .NET: C#, J# и Visual Basic.

Contester работает на Windows и на Linux. Язык реализации Delphi/freepascal. Представляет собой полноценную систему организации соревнований готовую к запуску с момента установки и распространяется в виде установочного файла или архива, в зависимости от операционной системы.

1.7.4. Executor

Executor[18] — автоматизированная сетевая тестирующая система для проведения турниров по программированию по правилам ACM. Executor - freeware, распространяется бесплатно и без каких-либо ограничений на использование. Тестирование контроля исполнения не производилось, так как сайт не работает (домен зарегестрирован на localhost).

1.7.5. PC^2

PC^2(Programming Contest Control System)[19] — система разработанная в Калифорнийском Государственном Университете Sacramento (CSUS) в поддержку соревнований по программированию, проводимых ACM и, в частности, ACM International Collegiate Programming Contest (ICPC) и его региональных этапов. Контроль исполнения, как неотъемлемая часть общей системы.

1.7.6. olympiads.ru/RUN

Система самотестирования olympiads.ru[20]. Используется для тестирования решений задач, представленных на соответствующем сайте. Система работает на платформе Windows, имеет инсталляционную программу и документацию. Не умеет компилировать, проверяет только exe, непригодна для проведения соревнований. Вследствие того что система не поддерживается с 2003 года и имеет узкий функционал в рамках данной работы детально рассматриваться не будет.

1.7.7. DOMjudge

DOMjudge[21] — автоматизированная тестирующая система, для проведения соревнований по программированию, подобных ACM ICPC. Основной упор сделан на удобство использования и безопасность. Система применялась во многих соревнованиях, распространяется свободно на условиях открытого ПО. Для контроля исполнения используется набор небольших консольных утилит.

1.7.8. dudge

Dudge[22] — это универсальная система для проведения олимпиад по программированию и другим предметам, написанная на Java и J2EE с использованием СУБД PostgresSQL и распространяющаяся по лицензии GPL. В качестве контроля исполнения используется динамическая библиотека, реализована для Windows и Unix. Библиотека используется в Java коде. Для создания и контроля исполнения процесса от другого пользователя требуются особые привилегии.

1.7.9. openjudge/sandbox

openjudge/sandbox[?] — набор библиотек, предоставляющий API, которое можно использовать для контроля простых программ с единственным процессом в ограниченном окружении, или, говоря иначе, песочнице. Позволяет создавать инструменты которые могут отслеживать и блокировать

поведение исполняемых программ прямо во время исполнения.

Простая в обращении библиотека. Возможно использование для портирования под Linux.

1.7.10. CATS/Spawner 1

CATS[11] — автоматизированная тестирующая система, для проведения как одиночных, так и командных соревнований по программированию. Для контроля исполнения используется консольная утилита. Создание и контроль исполнения процесса от другого пользователя требуют особые привилегии. Не поддерживается.

1.7.11. Spawner 2

Дальнейшее развитие модуля контроля запуска и исполнения программ cats-judge/Spawner[23]. На данный момент представляет собой обратно совместимую консольную утилиту с дополненными и улучшенными, по сравнению с предыдущей версией, возможностями. Функционал реализован только для систем семейства Windows.

Название	Операционная система	Лицензия	Интерфейс	Язык реализации	Запуск под другим пользователем	Комментарий
ejudge	Linux	GPL	Набор утилит	C	?	—
PCMS2/Run	Windows	Открытый код	Консольная утилита + динамич библиотека	C++	+ права	Аналогична спавнеру
Contester	?	?	?	?	?	—
Executor	Windows 2000/XP/2003	Закрытый код	?	?	?	—
PC^2	Windows, Linux	Закрытый код		+	+	—
olympiads.ru/ RUN	Windows	Закрытый код	Исполняемый файл	?	-	—
DOMjudge	Linux	GPL[24]	Набор консольных утилит	C	-	Запись статусов в файлы
dudge	Windows, Unix	GPL	Динамич. библиотека + биндинг	C/Java	?	—
openjudge/ sandbox	Linux	Закрытый код	Библиотека	C	?	
CATS/ Spawner1	Windows x86	???	Консольная утилита	C	+ права	—
Spawner2	Windows	GPL	Консольная утилита	C++	+	—

Таблица 2: Сравнение доступных систем контроля исполнения программ

По полученным данным можно судить о том что большинство популярных систем либо имеют закрытый код, либо являются платформозависимыми. Так же нельзя не отметить простоту библиотеки sandbox. В рамках данной работы для реализации функционала контролируемого исполнения под системами Linux предлагается использовать именно её.

1.7.12. Описание предшествующих работ

Студенты нашей кафедры успешно развивали систему CATS и модуль Spawner:

1. Дипломная работа «Система автоматического тестирования программ и организации соревнований по программированию», Автор Рожков М. Руководитель Кленин А. С. 2004[4]
2. Курсовая работа «Рендеринг математических выражений в MathML и HTML», Автор Матвиенко В. Руководитель Кленин А. С. 2005[5]
3. Курсовая работа «Поиск сходных алгоритмических конструкций в программном коде», Автор Коновалова Д. Руководитель Кленин А. С. 2005[6]
4. Курсовая работа «AJAX-интерфейс для системы CATS», Автор Перепечин В.В. Руководитель Кленин А. С. 2007[7]
5. Курсовая работа «Универсальный генератор тестов для системы CATS», Автор Туфанов И.Е. Руководитель Кленин А. С. 2008[8]
6. Курсовая работа «Модуль контролируемого исполнения программ Spawner», Автор Храпченков П.Ф. Руководитель Кленин А. С. 2013[9]

1.7.13. Вывод

Ценность данной работы подтверждается неугасающей популярностью соревнований по программированию и, как следствие, необходимостью развивать уже существующую систему CATS в целом и модуль Spawner в частности.

1.8. План работ

1. Рассмотреть возможность улучшения взаимодействия модуля judge со Spawner'ом.
2. Рассмотреть возможность переработки процедуры парсинга командных аргументов.
    - Изучить существующий набор опций и аргументов и оптимизировать его.
    - Добавить возможность изменять набор консольных аргументов.
3. Изучить готовые решения sandboxing'а.
4. Изучить openjudge/sandbox.
5. Спроектировать реализацию.
6. Поэтапно осуществить реализацию:
    - Реализовать улучшенное взаимодействие с judge.
    - Пересмотреть набор консольных аргументов и переписать их парсер.
    - Добавить поддержку openjudge/sandbox.

2. Требования к окружению

2.1. Требования к аппаратному обеспечению

Для работы требуется компьютер, пригодный для компиляции исходного кода программы или исполнения существующих пакетов. Т.е. кроме работающих процессора, оперативной и постоянной памяти требуются клавиатура и монитор.

2.2. Требования к программному обеспечению

2.2.1. Для разработчика

- ОС Windows не ниже XP
- CMake не ниже 2.6
- Компилятор C++(MSVC++ 2005+) или g++ 4.7.7 и выше
- Perl 5 для тестов
- Необходимые компиляторы и интерпретаторы для языков тестовых программ

2.2.2. Для пользователя

- ОС Windows не ниже XP
- При необходимости запуска процессов от имени другого пользователя - учетная запись с ограничениями

2.3. Требования к пользователям

От пользователей требуется умение пользоваться консолью.

3. Архитектура системы

Несмотря на то, что для конечного пользователя автоматизированная система представлена в виде Web интерфейса, она является аппаратно-программным комплексом, состоящим из нескольких модулей. В рамках данной работы необходимо рассмотреть внутреннее устройство системы, для обоснования некоторых принятых решений.

3.1. Архитектура системы CATS

Автоматизированная система CATS состоит из нескольких частей: веб-интерфейс, база данных, а так же модуль judge.

3.1.1. Веб-интерфейс

Веб-интерфейс[4] является кросс-платформенной частью системы. Веб сервер может быть запущен практически на любой системе. Фактически, с помощью веб-интерфейса осуществляется управление содержимым базы данных.

3.1.2. База данных

В базе данных[4] системы хранятся данные о проводимых турнирах, пакеты задач, исходный код, информация об участниках, их попытках, а так же информация по средам разработки. Однако привязка модуля judge к какому-либо виду базы данных усложняет его развертывание на конечной машине.

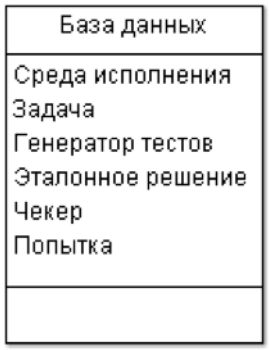

Рис. 2: Схематичное изображение базы данных

3.1.3. Модуль judge

Модуль judge[4] в свою очередь содержит загруженный кэш задач, кэш попыток, различные конфигурационные файлы и модуль Spawner.

Обычно, для сокращения нагрузки одновременно запущено несколько компьютеров с judge. Они подключаются к базе данных и распределяют задачи на исполнение между собой.

Рис. 3: Схематичное устройство модуля judge

В случае отсутствия задачи в кэше происходит загрузка из базы данных, генерация тестов и т.д.

В общем случае коммуникация между модулями проходит в следующем порядке:

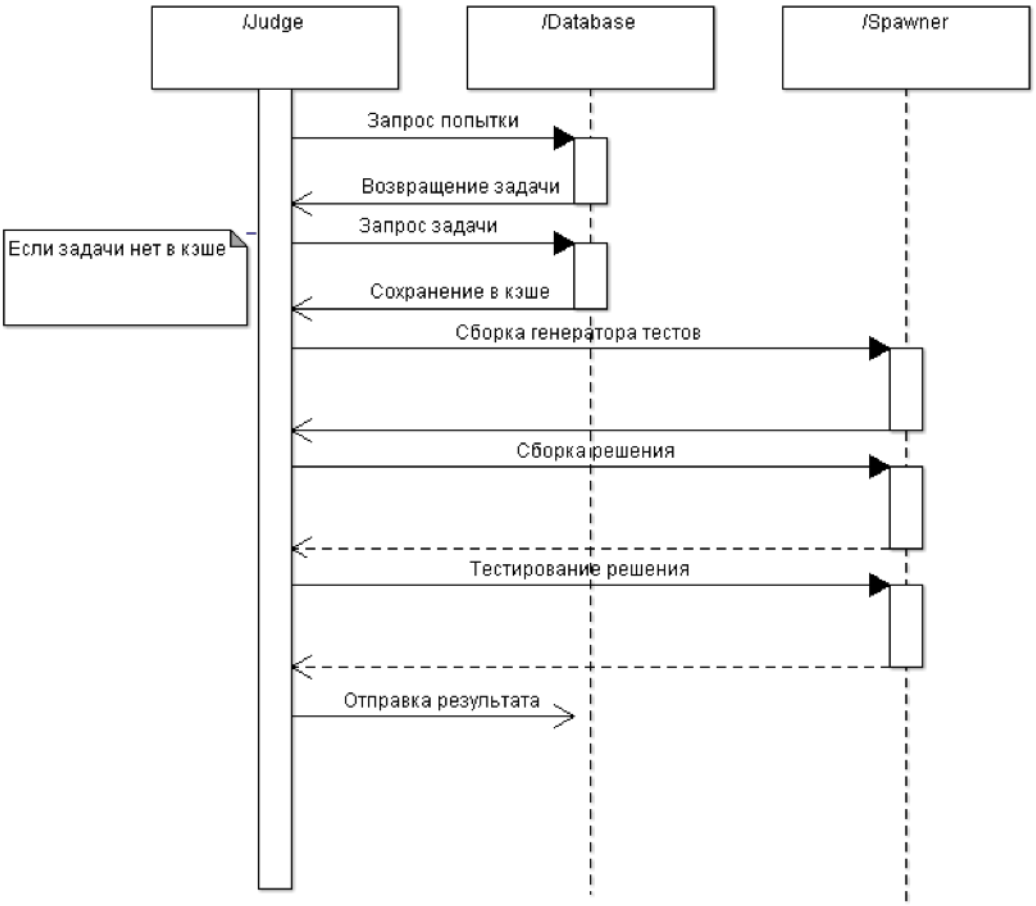

Рис. 4: Схема работы модуля judge

3.1.4. Модуль Spawner

Во время проведения тестирования требуется создать для программы достаточно изолированную среду и задать необходимые ограничения на ресурсы. Так же в случае некорректного хода исполнения тестируемой программы требуется определить характер и возможную причину ошибок и должным образом отреагировать на это. Этой работой занимается модуль Spawner.

Являясь подмодулем judge, Spawner в свою очередь разбит на три основных слоя: консольный интерфейс, прослойка совместимости и библиотека с набором внутренних классов.

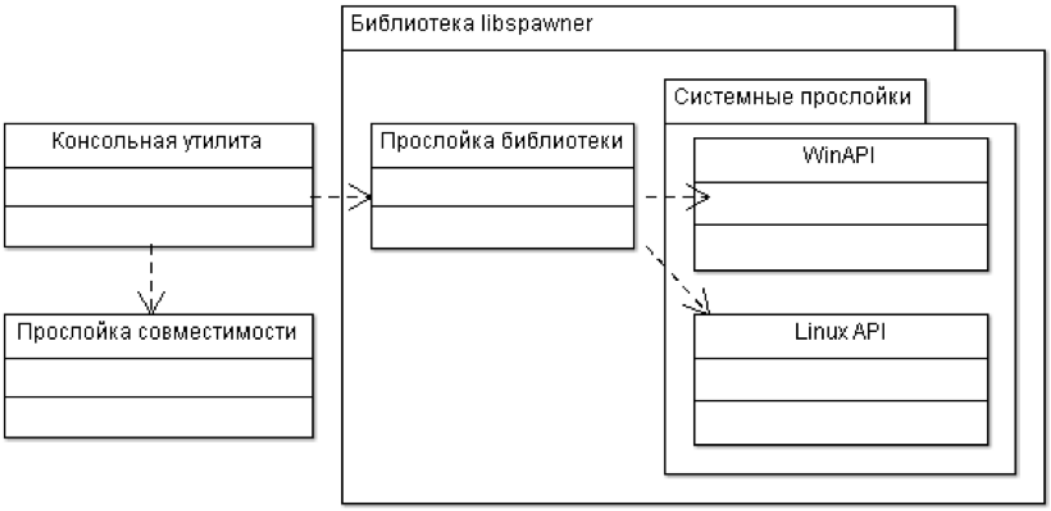

Рис. 5: Схема устройства модуля Spawner

Консольная утилита предоставляет интерфейс командной строки пользователю и, в зависимости от режима совместимости, набор аргументов и формат отчета. Режим совместимости задается специальным параметром. Прослойка совместимости выполняет связующую роль выходных данных библиотеки libspawner и требуемого формата отчета/аргументов. Сама же библиотека представляет набор интерфейсов, обеспечивающий доступ к конечным функциям операционной системы.

На текущий момент

4. Спецификация данных

4.1. Ограничения

Любые неправильные значения перечисленных ниже ограничений(например строковые константы) приводят к игнорированию соответствующего аргумента. Ограничения можно задавать и в других смежных единицах, так например память можно указывать в килобайтах(kB), а время в минутах (m). Подробнее смотри соответствующий раздел. По превышению любого из ограничений программа завершается с соответсвующим статусом. Иначе программа завершается со статусом ExitProcess.

Ограничение	Опция	Переменная окружения	Статус по заершению	Единицы	Описание
Время в пользовательском режиме	-tl	SP_TIME_LIMIT	TimeLimitExceeded	секунды	Устанавливает ограничение на время исполнения процесса в пользовательском режиме
Фактическое время	-d	SP_DEADLINE	TimeLimitExceeded	мегабайты	Устанавливает ограничение на физическое время исполнения программы
Память	-tl	SP_MEMORY	MemoryLimitExceeded	мегабайты	Устанавливает ограничение на используемую память
Жесткий диск	-tl	SP_WRITE_LIMIT	WriteLimitExceeded	мегабайты	Устанавливает ограничение на запись в файл
Минимальный уровень загрузки	-tl	SP_LOAD	—	проценты	Устанавливает минимальный порог уровня загрузки.
Время простоя	-lr	SP_IDLE_LIMIT	IdleLimitExceeded	секунды	Устанавливает ограничение на время в состоянии простоя.
Ограничение на системные ресурсы	-s	SP_SECURITY			Устанавливает ограничение на работу с системными дескрипторами

Таблица 3: Ограничения Spawner'a

Перевод единиц измерения Поддерживаются единицы трех видов – память, время и безтиповые. У памяти основными единицами являются байт и бит, у времени – секунда, минута, час, день.

Единица	Тип	Краткая запись
Байт	Память	B
Бит	Память	b
Секунда	Время	s
Минута	Время	m
Час	Время	h
День	Время	d

Таблица 4: Единицы измерения

Так же возможно добавление префикса из системы «Си». Т.е. например – мегабайт, килобит, миллисекунда и т.д.

Префикс	Значение	Краткая запись
Дека	10	da
Гекто	10^2	h
Кило - Киби	10^3 - 1024^1	k - Ki
Мега - Меби	10^6 - 1024^2	M - Mi
Гига - Гиби	10^9 - 1024^3	G - Gi
Тера - Теби	10^{12} - 1024^4	T - Ti
Деци	10^{-1}	d
Санти	10^{-2}	c
Милли	10^{-3}	m
Микро	10^{-6}	u

Таблица 5: Степени единиц измерения

Таким образом, значение ограничения можно записывать в виде:

<value>[degree][unit]

Где value – значение, degree – степень, unit – единица измерения. Если указано только значение, то его единица измерения будет установлена по-умолчанию.

4.2. Опции общие для всех режимов совместимости

4.2.1. Выбор режима совместимости

Ключ – - -legacy=<значение>

Переменная окружения – SP_LEGACY = <значение>

Возможные значения – sp99, sp00, pcms2

sp99 – режим совместимости со старой версией Spawner'a.

sp00 – основной режим Spawner'a.

pcms2 – режим совместимости с PCMS2/Run

4.2.2. Вывод помощи

Ключ – - -help, -h

Выводит справку для конкретно выбранного режима совместимости

4.3. Опции Spawner

Отключение системных диалогов ошибок Задается ключом – - -hide_errors

Отключает стандартный диалог о составлении отчета об ошибке, а так же диалог исключения. Подробнее смотри соответствующий раздел.

Отключение отображения пользовательского интерфейса Задается ключом – - -hide_gui

Отключает видимый пользовательский интерфейс и окна. Подробнее смотри соответствующий раздел.

Отключение ошибок crtdebug Задается ключом – - -hide_crt

Отключает у программ, скомпилированных в Visual Studio в конфигурации Debug отображение окна исключения crtdebug. Подробнее смотри соответствующий раздел.

Запуск программы с поиском ее в системных путях Задается ключом – - -systempath

Позволяет системе искать исполняемый файл запускаемой программы в системных путях.

Рабочая директория Задается ключом -wd=<working_directory>

Устанавливает рабочую директорию для запускаемой программы.

Управление стандартным потоком ввода процесса Задается ключом – - -in=<what>

Перенаправляет поток ввода исполняемого процесса на указанный ресурс. Если <what> начинается с *, то будет произведен поиск по глобальному пространству имен, в которое входят идентификаторы stdout, stdin, stderr, report, и произведено соответствующее соединение или будет выдан отчет с ошибкой.

Переменная окружения SP_INPUT_FILE

Управление стандартным потоком вывода процесса Задается ключом – - -out=<what>

Перенаправляет поток вывода исполняемого процесса на указанный ресурс. Если <what> начинается с *, то будет произведен поиск по глобальному пространству имен, в которое входят идентификаторы stdout, stdin, stderr, report, и произведено соответствующее соединение или будет выдан отчет с ошибкой.

Переменная окружения SP_OUTPUT_FILE

Управление стандартным потоком ошибок процесса Задается ключом – - -err=<what>

Перенаправляет поток ошибок исполняемого процесса на указанный ресурс. Если <what> начинается с *, то будет произведен поиск по глобальному пространству имен, в которое входят идентификаторы stdout, stdin, stderr, report, и произведено соответствующее соединение или будет выдан отчет с ошибкой.

Переменная окружения SP_ERROR_FILE

Установка имени пользователя Задается ключом – -u=user@domain

Устанавливает имя пользователя для делегированного запуска программы. Работает только в связке с опцией пароля - -p=<password>.

Переменная окружения SP_USER

Установка пароля пользователя Задается ключом – -p=<password>

Устанавливает пароль для делегированного запуска программы. Работает только в связке с опцией пароля - -u=user@domain.

Переменная окружения SP_PASSWORD

4.4. Опции для совместимости с предыдущей версией Spawner'a

Совпадающие опции Совпадающие опции имеют абсолютно тот же функционал с новым стандартом Spawner'a

Опция старой версии	Опция новой версии
–i:<what>	- -in=<what>
–so:<what>	- -out=<what>
–se:<what>	- -err=<what>

4.4.1. Сохранение отчета в файл

Задается ключом – –sr:<what>

Сохраняет сгенерированный отчет в заданный файл.

Переменная окружения SP_REPORT_FILE

4.4.2. Скрыть отчет

Задается ключом – –hr:<what>

Если значение опции 1, то возвращаемый отчет не выводится.

Переменная окружения SP_HIDE_REPORT

4.4.3. Скрыть стандартный вывод приложения

Задается ключом – –ho:<what>

Если значение опции 1, то возвращаемый программой стандартный вывод не выводится в консоль.

Переменная окружения SP_HIDE_OUTPUT

4.5. Опции совместимости с PCMS2/Run

Ограничения

4.5.1. Ограничение времени в пользовательском режиме

Ключ – -t <значение>

Задает ограничение на время исполнения в пользовательском режиме. Единицы по-умолчанию секунды.

4.5.2. Ограничение оперативной памяти

Ключ – -m <значение>

Задает ограничение на оперативную память. Единицы по-умолчанию мегабайты.

4.5.3. Минимальный порог загрузки

Ключ – -r <значение>

Задает порог загрузки для определения состояния простоя. По-умолчанию принимает дробные значения от 0 до 1. Может принимать значения выраженные в процентах, например 25%.

4.5.4. Ограничение на время простоя

Ключ – -y <значение>

Задает ограничение на время исполнения в состоянии простоя. Единицы по-умолчанию секунды.

4.5.5. Другие опции

Название	Ключ
Имя пользователя	-l <значение>
Пароль	-p <значение>
Рабочая директория	-d <значение>
Сохранить отчет в файл	-s <значение>
Сохранить stdout в файл	-o <значение>
Сохранить stderr в файл	-e <значение>
Считать stdin из файла	-i <значение>
Не выводить в консоль	-q
Показать интерфейс	-w

4.6. Формат отчета

4.6.1. Отчет модуля Spawner1

По завершению работы контролируемого процесса в консоль и/или в выбранный файл выводится отформатированный отчет. Все значения выводятся в виде

<Свойство>: <Значение>

В первой половине отчета идет описание заданных ограничений и опций, а во второй - рабочие значения по соответствующим характеристикам, а так же статусы.

Пример совместимой версии отчета:

```
spawner>sp --cmd gcc.exe

--------------- Spawner report ---------------
Application:                gcc.exe
Parameters:                 <none>
SecurityLevel:              0
```

```
CreateProcessMethod:        CreateProcess
UserName:                   shigidono
UserTimeLimit:              Infinity
Deadline:                   Infinity
MemoryLimit:                Infinity
WriteLimit:                 Infinity
-----------------------------------------------
UserTime:                   0.000000 (sec)
PeakMemoryUsed:             0.953125 (Mb)
Written:                    0.000060 (Mb)
TerminateReason:            ExitProcess
ExitStatus:                 1
-----------------------------------------------
SpawnerError:               <none>
```

И для предыдущей версии spawner'а

```
sp gcc.exe

--------------- Spawner report ---------------
Application:            gcc.exe
Parameters:             <none>
SecurityLevel:          0
CreateProcessMethod:    CreateProcess
UserName:               shigidono
UserTimeLimit:          Infinity
DeadLine:               Infinity
MemoryLimit:            Infinity
WriteLimit:             Infinity
-----------------------------------------------
UserTime:               0.000000 (sec)
PeakMemoryUsed:         0.000000 (Mb)
Written:                0.000000 (Mb)
TerminateReason:        <none>
ExitStatus:             0
-----------------------------------------------
SpawnerError:           CreateProcess failed with error 2 ()
```

В данном формате отчета:

- Application - Имя приложения;
- Parameters - Параметры приложения;
- SecurityLevel - Уровень защиты;
- CreateProcessMethod – Метод создания процесса;

- UserName - Имя пользователя под которым был запущен дочерний процесс в формате: User[@Domain];
- UserTimeLimit - максимальное время в сек. выполнения процесса в пользовательском режиме по истечении которого процесс прерывается. По умолчанию: "Infinity";
- Deadline - Время в сек., которое выделено процессу. По умолчанию: "Infinity";
- Отличается от TimeLimit тем, что это физическое время. Если процесс непрерывно осуществляет ввод/вывод, находиться в состоянии ожидания или система перегружена, то процесс может выполнятся неограниченно долго несмотря на TimeLimit. Для предотвращения данной ситуации нужно установить DeadLine;
- MemoryLimit - Максимальный объем выдаляемой памяти процессу в Mb. По умолчанию: "Infinity";
- WriteLimit - Максимальный объем информации, который может быть записан процессом в Mb. По умолчанию: "Infinity";
- UserTime - Фактическое время выполнения процесса в сек;
- PeakMemoryUsed - Максимальное использование виртуальной памяти процессом в Mb;
- Written - Объем информации, который был записан процессом в Mb;
- TerminateReason - Причина завершения процесса. Может быть:
  - "ExitProcess" – процесс завершился нормально
  - "MemoryLimitExceeded" – превышен лимит памяти
  - "TimeLimitExceeded" – превышен лимит времени выполнения (либо TimeLimit, либо Deadline);
  - "WriteLimitExceeded" – превышен лимит записи;
  - "AbormalExitProcess" – процесс завершился с исключением (спи-

сок исключений см. ниже);

Если процесс не был завершен, то данному полю соответствует значение "<none>".

- ExitStatus - Статус завершения процесса. Принимает значение кода возврата процесса. Если ExitStatus = 0, то процесс завершился нормально.

- SpawnerError - Текст ошибки при работе Spawner'a. Если при работе ошибка не произошла, то полю соотвествует значение "<none>".

4.6.2. Отчет в режиме совместимости с PCMS2/Run

В результате исполнения модуля в режиме совместимости pcms2 и в случае, когда не отключен вывод на консоль, выводится следующий отчет:

```
Running "c:\Windows\System32\PING.EXE ya.ru", press ESC to terminate...
Memory limit exceeded
Program tried to allocate more than 10 bytes
  time consumed: 0.00 of 1.00 sec
  time passed:   0.03 sec
  peak memory:   155648 of 10 bytes
```

Когда задан файл для вывода отчета, отчет выводится в следующем формате:

```
average.memoryConsumed=1125376.000
average.timeConsumed=15.250
average.timePassed=446.000
invocations=4
last.memoryConsumed=155648
last.timeConsumed=0
last.timePassed=30
max.memoryConsumed=4030464
max.timeConsumed=46
max.timePassed=1518
min.memoryConsumed=155648
min.timeConsumed=0
min.timePassed=30
total.memoryConsumed=4501504
total.timeConsumed=61
total.timePassed=1784
```

4.6.3. Отчет Json

Для простоты более плотной интеграции с другими модулями, такими как judge было принято решение о реализации вывода отчета в формате json. Вывод в этом формате задается специальным аргументом или переменной окружения. Пример отчета:


```
{
   "Application" : "3.exe",
   "CreateProcessMethod" : "CreateProcess",
   "Deadline" : "Infinity",
   "ExitCode" : 0,
   "ExitStatus" : "0",
   "MemoryLimit" : {
      "real_value" : 268435456,
      "units" : "MB",
      "value" : 256.0
   },
   "Parameters" : [],
   "PeakMemoryUsed" : {
      "real_value" : 0,
      "units" : "MB",
      "value" : 0.0
   },
   "SecurityLevel" : false,
   "SpawnerError" : "<none>",
   "TerminateReason" : "MemoryLimitExceeded",
   "UserName" : "shigidono",
   "UserTime" : {
      "real_value" : 0,
      "units" : "Second",
      "value" : 0.0
   },
   "UserTimeLimit" : {
      "real_value" : 1000,
      "units" : "Second",
      "value" : 1.0
   },
   "WriteLimit" : {
      "real_value" : 31457280,
      "units" : "MB",
      "value" : 30.0
   },
   "Written" : {
      "real_value" : 0,
      "units" : "MB",
```

```
      "value" : 0.0
   }
}
```

4.7. Множественный ввод/вывод

Благодаря своей структуре, библиотека libspawner позволяет сделать множественное определение потоков вывода. Так поток вывода программы может быть направлен как на стандартный вывод в консоль, так на запись в файл. Эти процессы происходят одновременно. Возможно это благодаря тому, что каждому потоку вывода можно сопоставить несколько конечных и начальных точек. В консоли это решается путем задания соответствующих конкретным потокам вывода аргументов. Например:

```
sp.exe --out=1.txt --out=2.txt --out=*std a.exe
```

Данная команда произведет запись в 2 файла и выведет информацию со стандартного потока вывода программы a.exe в консоль.

4.8. Запуск многих процессов

В данной работе был спроектирован и реализован консольный интерфейс запуска многих процессов. Для разделения аргументов, опций и ограничений, принадлежащих разным программам, используется специальный ключ-разделитель. Он задается при помощи флага

```
--separator=<separator_string>
```

После этого когда в аргументах Spawner'а встречается флаг вида --<separator_stri
система считает следующий за ним набор аргументов принадлежащих следующей программе. В случае если внутри двух флагов разделителей не встречаются аргументы необходимые для запуска программы, Spawner выдает отчет с ошибкой. Для перенаправления портов ввода/вывода используется синтаксис ссылок и пространств имен. Каждая программа представляется в виде пространства имен с идентификатором, по умолчанию являющимся порядковым номером программы от 0. Каждое пространство имен предоставляет такие идентификаторы, как stdout, stderr, stdin, report.

Т.е. для того чтобы подключить к потоку ввода поток вывода 2го процесса необходимо задать следующий флаг

```
--in:*1.stdout
```

Пример

```
sp.exe --separator=// --in:*1.stdout program.exe --// --in:*0.stdout --err:
    error.txt checker.exe --// --in:*0.stderr logger.exe
```

В данном примере поток вывода 2го процесса будет подключен к потоку ввода первого, а поток вывода первого процесса – ко вводу второго. Можно изменить идентификатор программы внутри пространства аргументов этой программы, задав флаг –program:<id>, причем <id> не может содержать служебные символы. Тогда обращение будет выглядеть как - -in:*<id>.stdout

5. Функциональные требования

Spawner должен соответствовать следующим требованиям

1. Возможность накладывать ограничения:
   - Ограничение на объем памяти используемой исполняемой программой
   - Ограничение на объем дискового пространства доступного исполняемой программе
   - Ограничение на время исполнения в пользовательском режиме ОС
   - Ограничение на фактическое время исполнения
   - Ограничение на время простоя вместе с ограничением на пороговое значение загрузки
   - Ограничение на доступ к системным дескрипторам
2. Корректная обработка ошибок:
   - Системный диалог об ошибке
   - Диалог об ошибке среды разработки
   - Исключения во время исполнения
3. Доступное перенаправление каналов ввода/вывода:
   - Вывод и ввод в/из файлов в стандартные каналы ввода/вывода/ошибки исполняемого процесса
   - Вывод и ввод в/из стандартных потоков Spawner'а в стандартные каналы ввода/вывода/ошибки исполняемого процесса
4. Кросс-платформенность:
   - Поддержка операционных систем Windows с XP
   - Поддержка операционных систем Windows x64

5. Возможность множественного исполнения:
    - Одновременный запуск нескольких программ
    - Вывод и ввод в/из стандартных потоков одного исполняемого процесса в стандартные каналы ввода/вывода/ошибки другого исполняемого процесса
6. Тесная интеграция с модулем judge:
    - Возможность запуска как первой, так и второй версии Spawner'a
7. Возможность запуска от другого пользователя

6. Требования к интерфейсу

Весь функционал к модулю Spawner должен быть доступен через интерфейс командной строки. В зависимости от режима, интерфейс должен взаимодействовать с интерфейсом Spawner1. Параметры по умолчанию должны быть загружены из переменных окружающей среды. По окончанию исполнения программа должна выдавать отчет. Должна присутствовать возможность сохранения отчета в отдельный файл. Так же для удобства должна присутствовать возможность вывода стандартных потоков в консоль или файл(по выбору). К тому же должна присутствовать возможность запуска программы из-под пользователя с заданным именем и паролем. И наконец, должна быть реализована простейшая система перевода единиц, благодаря которой было бы возможно передавать программе ограничения в разных форматах. Дополнительно требуется возможность для соединения потоков ввода/вывода разных процессов с помощью консольных аргументов.

Требуется реализовать удобный интерфейс программной библиотеки для использования возможностей контролируемого исполнения в других проектах.

Так же необходимо исследовать другие аналогичные системы автоматизированные системы соревнований и предложить интерфейс, упрощающий написание соответствующих прослоек совместимости для возможного

в них внедрения модуля Spawner. Например, системы, у которых аналогичные модули представлены зависимыми от операционной системы программами, могут быть портированы с помощью замены системнозависимого модуля на Spawner.

Запуск контроля исполнения программ должен осуществляться следующим способом

sp.exe [ограничения и опции] программа[аргументы программы] Пример использования:

```
sp.exe -tl=400ms -wl=1 -ml=1 tests/write-limit/write-limit.exe
```

Пример отчета можно посмотреть в соответствующем разделе.

7. Проект

7.1. Средства реализации

При анализе задачи ставился акцент на следующие моменты

- переносимость проекта на другие операционные системы
- гибкая настройка проекта
- интеграция с системой тестирования
- удобная система объектов и модулей

В следствие чего был выбран язык реализации C++ и система сборки CMake. А так как требуется достичь совместимости с CATS, для системы тестов был выбран язык Perl.

7.2. Структуры данных

Основным классом можно назвать класс runner, а так же его потомок – secure_runner. Он отвечает за создание процесса с определенными настройками и дальнейший контроль исполнения. Этому классу передаются опции и ограничения, а так же настройки перенаправления потоков ввода/вывода.

Опции представляют собой класс options_class, полями которого являются настройки исполнения. Ограничения представляют собой класс restrictions_class. Для изменения какого-либо ограничения необходимо вызвать одну из функций

```
set_restriction(const restriction_kind_t &kind, const restriction_t &value);
set_restriction(const std::string &kind, const restriction_t &value);
```

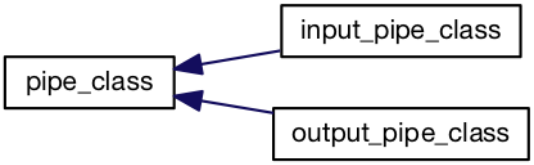

Рис. 6: Диаграмма классов каналов ввода/вывода

Перенаправление потоков ввода/вывода определяется двумя классами - input_pipe_class и output_pipe_class. Они принимают векторы начальных и конечных точек соединения соответственно. Начальные точки соединения определяются потомками класса input_buffer_class. Конечные – output_buffer_class. Точки могут соединять с файлами, а так же с потоками ввода/вывода. Так, например, класс handle_buffer_class отвечает за запись и чтение в заданные дескрипторы, будь то дескрипторы консоли или дескрипторы файла.

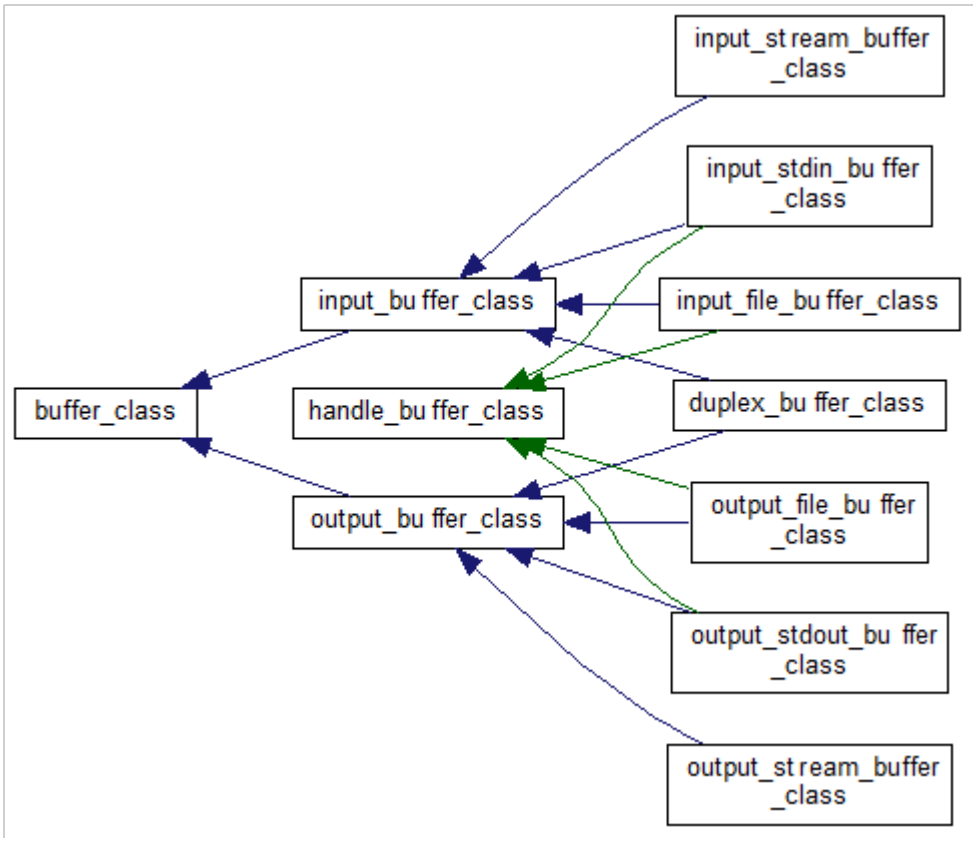

Рис. 7: Диаграмма классов буфферов

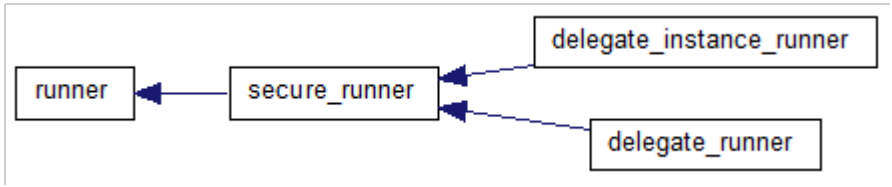

Рис. 8: Диаграмма классов исполнителей

Создание процесса происходит вызовом соответствующей функции у объекта-потомка класса runner – процесс может быть создан синхронно(run_proces и асинхронно(run_process_async). В зависимости от класса(secure_runner, delegate_runner, delegate_instance_runner) выбирается способ запуска программы. Изначально процесс создается в “замороженном” состоянии. В случае ошибки создания процесса - устанавливается соответствующий статус. После этого процесс запускается. В случае обычного запуска функция возвращает только когда выполнение процесса завершено или случилась какая-либо ошибка. В случае асинхронного запуска функция возвращает сразу – выполнение процедуры запуска происходит в отдельном потоке. Информацию о статусе выполнения можно получить вызовом метода

get_process_status(). В том случае, если программа не завершилась – статус будет иметь значение process_still_active.

7.3. Модули и алгоритмы

Spawner, являясь модулем автоматизированной системы CATS, в свою очередь разделен на несколько подмодулей — статическая библиотека libspawner, сам spawner или sp, представляющий из себя консольную утилиту, а так же модуль совместимости. Помимо этого в проекте содержится модуль тестирования.

7.3.1. Библиотека libspawner

Исходные файлы библиотеки располагаются в соответствующей директории. В папке inc - содержатся заголовочные файлы. В корне директории находится файл spawner.h, который является основным, для включения в какой-либо проект.

Содержимое папки inc

- buffer.h – модуль отвечающий за конечные точки перенаправленных потоков ввода/вывода
- compatibility.h – модуль отвечающий за совместимость со старой версией Spawner'a
- delegate.h – модуль отвечающий за делегированный запуск процессов
- error.h – модуль отвечающий за обработку ошибок системы Spawner а так же за дальнейший их вывод
- options.h - модуль отвечающий за класс настроек
- pipes.h - модуль отвечающий за перенаправление потоков ввода/вывода
- platform.h - модуль отвечающий за системно зависимые переменные и типы

- report.h - модуль для получения строковых значений из перечислимых типов проекта
- restrictions.h - модуль класса ограничений
- runner.h – модуль содержащий базовый и минимальный класс, способный запускать программы
- securerunner.h – модуль отвечающий за ограниченный запуск программ
- session.h – модуль отвечающий за уникальность сессии делегированного запуска
- status.h - модуль статусов
- uconvert.h - модуль преобразования единиц, созданный для повышения комфорта работы с программой

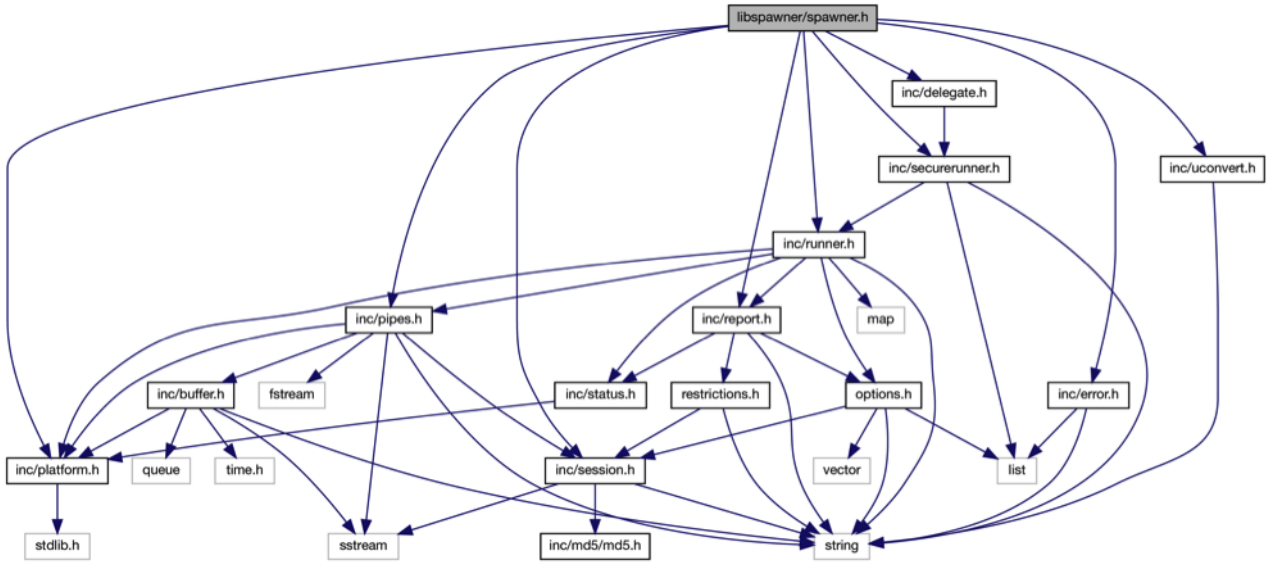

Рис. 9: Схема связи модулей библиотеки libspawner

В папке src, в корне или в соответствующей операционной системе директории содержатся реализации соответствующих классов.

Используя набор предоставляемых классов и интерфейсов, можно создавать множество процессов и, контролируя их исполнение, обеспечивать их взаимосвязь друг с другом.

В качестве примера был создан демонстрационный проект(см. материалы в конце), представляющий из себя игру с угадыванием последовательности чисел. После того как программно задаются участники – некоторые исполняемые файлы, программа-исполнитель загадывает последовательность чисел и на догадки программ-участников отвечает сколько из чисел угаданы верно. Игра продолжается, до тех пор, пока все участники не угадают загаданную последовательность чисел. Коммуникация между игроками и исполнителем осуществляется через потоки ввода/вывода, соединенные друг с другом программно, с помощью средств libspawner, а так же подобная программа была реализована с использованием консольного интерфейса Spawner'a.

7.3.2. Модуль консольной утилиты

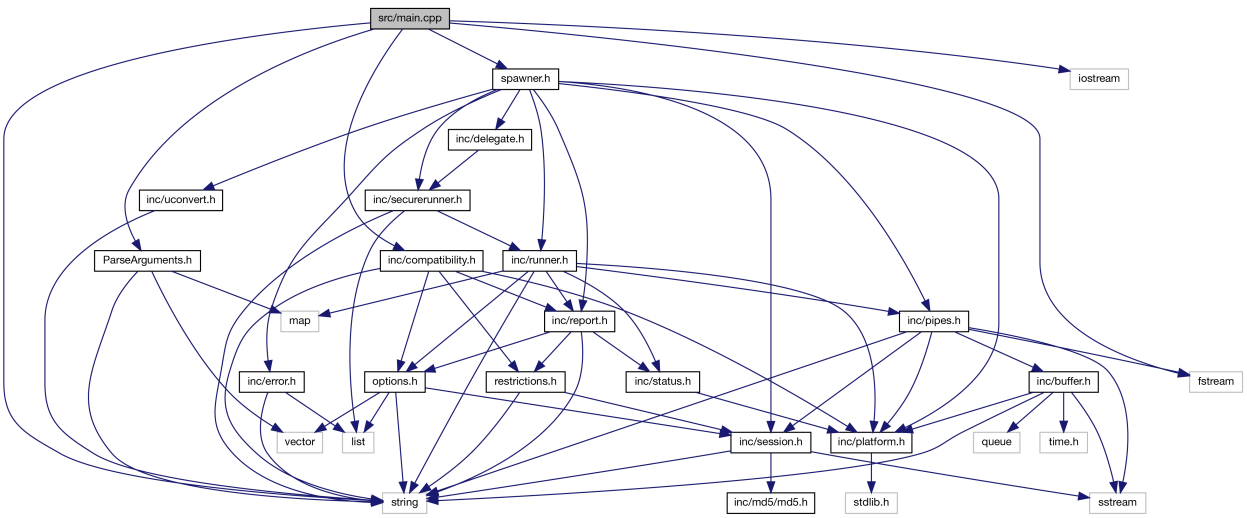

Рис. 10: Схема связи модулей консольной утилиты

Проект включает в себя статическую библиотеку libspawner. Состоит из двух внутренних модулей - модуль парсинга консольных аргументов SimpleOpt и главный модуль, использующий интерфейсы libspawner для реализации поставленной темой задачи.

7.3.3. Модуль совместимости

Представляет собой прослойку, позволяющую структуры данных получаемых от библиотеки переводить в различные форматы соответствую-

щие

7.3.4. Модуль тестирования

Содержит набор тестов. Подробнее смотри раздел «Реализация и тестирование».

7.3.5. Механизм ограничения

В новой реализации Spawner унаследовал способ наложения ограничений на дочерние процессы. Для этого используется используется механизм JOB_OBJECT'ов[25], позволяющий на уровне системы контролировать использование системных ресурсов программой, которой был назначен этот объект. В случае превышения конкретного ресурса, можно определить что это был за ресурс и каково было его использование. На некоторые ресурсы ограничения можно накладывать непосредственно. Ресурсы для которых такой возможности нет предлагается наблюдать в отдельном потоке Spawner'а, и, в случае нарушения, завершать JOB_OBJECT с указанным кодом.

7.3.6. Механизм перенаправления потоков ввода/-вывода

В данной работе использовался подход, сходный с реализованным в оригинальной версии Spawner'а. Был реализован гибкий интерфейс классов, построенный поверх системной прослойки для осуществления таких схем как «один источник – много выходов». Интерфейс классов представлен двумя базовыми объектами – соединительный канал (pipe) и буфер данных. Соединительный канал осуществляет связь между соответствующим потоком ввода/вывода дочернего процесса и буфером. Буфер данных представляет собой абстрактный объект, который может осуществлять как запись/чтение файла, консоли, так и работу с участками памяти. Т.е. данные из соединительного канала с помощью объекта буфера могут быть выведены в совершенно различные источники. Каждый объект соедини-

тельного канала имеет свой отдельный программный поток, в котором в зависимости от направления канала происходит чтение из буферов и запись или наоборот, чтение из канала и запись в объекты буферов.

7.3.7. Отключение диалогов об ошибках и пользовательского интерфейса

Одним из требований к модули была возможность отключение сообщений об ошибках, выдаваемых как средой разработки так и операционной системой. Для решения проблемы были использованные специальные функции WinAPI, такие как SetErrorMode, а так же были установлены специальные параметры при создании дочерних процессов. Возможность скрытия графического интерфейса также реализуется при помощи создания процесса со специальными параметрами STARTF_USESHOWWINDOW и SW_HIDE.

Среда разработки Microsoft Visual Studio при компиляции программ в режиме отладки в случае исключения генерирует диалог об ошибке, которые невозможно скрыть ни одним из указанных выше методом. Для решения этой проблемы использовался запуск контролируемого процесса в режиме отладки и постоянная проверка на исключение. Когда оно возникает Spawner просто завершает дочерний процесс.

7.3.8. Запуск программ от другого пользователя

В изначально версии Spawner'а, а так же в некоторых других подобных модулях для запуска процесса от учетной записи другого пользователя использовалась стандартная функция WinAPI CreateProcessWithLogon. Однако, при выполнении этой функции для дочернего процесса автоматически создается анонимный JOB_OBJECT. Таким образом, в следствие невозможности назначения нескольких таких объектов одной программе получается, что установление ограничений не представляется возможным. В рамках данной работы было проведено исследование и выделено несколько возможных путей решения.

Первый путь решения предполагает собой написания драйвера ядра с

использованием недокументированного API (5). В низкоуровневом системном Windows Driver API существует функция ObReferenceObjectByHandle, позволяющая получить доступ ко всем дескрипторам выбранного процесса. Однако, сложность данного решения заключается как в написании подобного драйвера ядра, так и портировании его на другие системы.

Другим решением является запуск процесса-агента. Однако данное решение было признано неподходящим, вследствие необходимости запуска сервиса-агента, постоянно запущенной пользовательской сессии, а так же сложностями с контролируемым исполнением.

Еще одним решением является делегированный запуск. Делегированный запуск представляет собой запуск дочерним процессом еще одного Spawner'а от учетной записи другого пользователя. Этому процессу «делегируется» роль исполнения конечной программы. Ограничения по системным ресурсам создаются в именном объекте JOB_OBJECT в главном процессе Spawner'а. В роли имени JOB_OBJECT'а выступает некоторый идентификатор, получаемый исходя из следующих пунктов:

- Время запуска родительского процесса Spawner'а
- Опции запуска дочернего процесса
- Порядковый номер в случае множественного исполнения

Полученный идентификатор передаются делегату с помощью внутреннего аргумента – - -session=<id>.

Однако существует сложность наследования дочерними процессами родительского JOB_OBJECT'а. Эта сложность решается недокументированным поведением API. Для этого родительскому Spawner'у назначается JOB_OBJECT с параметрами, запрещающими выход дочерних процессов из него. После этого при создании дочерних процессов под другим пользователям они не получают свой анонимный JOB_OBJECT, наследуя вместо этого соответствующий объект JOB_OBJECT родителя. После Затем в родительском процессе отменяется запрет на выход дочерних процессов из JOB_OBJECT'а и делегированный процесс уже запускает заданную программу с необходимыми опциями. В рамках данной работы была реализована система перенаправления потоков ввода/вывода, основанная на имен-

ных соединительных каналах. Однако, из-за этого некоторые программы не исполнялись корректно. Поэтому было решено реализовать перенаправления на основе стандартных безымянных каналов. Созданные каналы передаются процессу «делегату» в качестве стандартных потоков ввода/вывода при создании. После чего, процесс «делегат» уже передает эти потоки запускаемому дочернему процессу. В случае если отключен режим отладки (см. подробнее раздыл о скрывании ошибок), процесс «делегат» завершается. Иначе он работает в качестве отладчика. Здесь возникает проблема нотификации главного процесса о том, что в дочернем процессе произошло исключение. Одним из возможных способов реализации является использование системы сообщений JOB_OBJECT'а.

7.3.9. Множественное исполнение

Благодаря гибкой системе соединительных каналов и буферов (см. соответствующий пункт о перенаправлении потоков ввода/вывода), соединение двух каналов возможно с помощью объекта буфера, работающего с памятью как на чтение так и на запись.

7.3.10. Ограничение по времени простоя

В рамках данной работы для совместимости, а так же улучшения было принято решение реализовать ограничение по времени простоя. Подобное ограничение успешно используется на Полуфинальном этапе чемпионата ACM[16].

Благодаря этому ограничению можно сэкономить значительное время на уничтожении процессов по факту не выполняющих полезной работы определенное время. Причинами такого поведения программ могут быть: ожидание выполнения операции ввода/вывода, системная операция сна, различные ожидания объектов и пр. Если в старой версии Spawner'а требовалось ждать 10 секунд глобального таймаута, что для 100 тестов является довольно продолжительным, то сейчас подобные ничего не делающие программы могут быть уничтожены за гораздо более короткое время. В качестве входных параметров принимается уровень пороговый загруз-

ки, который будет считаться за простой и ограничение на время простоя. Когда уровень загрузки опускается ниже порога, запускается глобальный таймер, по истечении которого дочерний процесс уничтожается с помощью механизма JOB_OBJECT'ов с соответствующим кодом.

7.4. Стандарт кодирования

За основу стиля наименования классов, методов, свойств, функций, типов и т.д. был взят стиль стандартной библиотеки C/C++. Некоторая часть имен использует стиль WinAPI, вследствие специфики.

8. Реализация и тестирование

Работоспособность системы была проверена на следующих системах

- Windows 7 x64
- Windows Server 2008 x86
- Windows XP SP3 x86

И на следующих компиляторах

- Microsoft Visual Studio 2008 x86
- Microsoft Visual Studio 2012 x86
- Microsoft Visual Studio 2012 x64
- MinGW GCC 4.7.7

Для наибольшей совместимости рекомендуется использовать компилятор GCC.

В рамках данной работы, для облегчения проверки основных механизмов работы на корректность на разных системах была разработана система тестирования.

Для разработки данной системы был использован язык Perl и его стандартная библиотека тестирования. Для сборки тестов была написана серия

шаблонов CMake файлов, позволяющих создавать иерархию в тестовых директориях не усложняя при этом процесс сборки. Так же данные шаблоны автоматически копируют получившиеся исполняемые файлы тестов для дальнейшей работы с ними. Конкретный тест представляет собой исполнение тестовой программы при помощи модуля Spawner, получение отчета исполнения и, наконец, проверки соответствующих полей. Так как в модуле judge системы CATS существует реализация процедуры запуска Spawner'а и разбора отчета, было принято решение использовать для написания системы тестирования язык Perl и вынести общие процедуры в отдельный модуль. Тесты разбиваются по категориям и разносятся по соответствующим папкам, представляющим собой отдельную директорию с модулем тестирования и набором проектов. В каждый проект входит отдельные исходники на каком-либо языке программирования, а так же файл тестов. Библиотека тестирования имеет составляющую, написанную на CMake, а так же составляющую, написанную на Perl. Часть на CMake используется исключительно для сборки и ,поэтому, проводится в первую очередь. После того как все тестовые программы были успешно собраны и перенесены в общую директорию можно запускать основной скрипт Perl, который запустит все добавленные тесты. Один из наборов тестов представляет собой проверку контроля исполнения старой версии и новой Spawner'а на идентичность в некотором приближении на наборе простых программ.

Так же присутствуют тесты на проверку каждых ограничений и опций в отдельности. Общее количество тестов - около 40. Было проведено внедрение модуля в систему CATS и он прошел тестирование в реальных условиях нескольких соревнований, таких как Четвертьфинал командного чемпионата ACM, а так же школьные олимпиады по программированию. Работа модуля сопровождалась переменным успехом, однако выявленные в ходе проверки ошибки и несовместимости были или будут исправлены. Помимо этого тестирование модуля осуществлялось вручную по методу белого ящика на нескольких операционных системах и средах разработки.

Заключение

В ходе дипломной работы был расширен модуль контроля исполнения программ. Были реализованы в полном объеме поставленные функциональные требования. Так же разработанный модуль был введен в состав системы CATS на место старой версии, а в саму систему были внесены изменения для более тесной интеграции.

Основная программа состоит из двух частей — своих исходников и библиотеки libspawner. Объём кода составляет 250KB (5088 строк). В ходе предквалификационной практической работы были изучены: модуль judge системы CATS, система сборки CMake, язык программирования Perl.

Так же были углублены знания в WinApi и Microsoft Windows. По имеющимся на данный момент результатам можно говорить об успешной реализации подмножества функциональных требований.

Список литературы

[1] Международная студенческая олимпиада по программированию ACM/ICPC — Wikipedia

[2] Steven S Skiena, Miguel A. Revilla, Programming Challenges: The Programming Contest Training Manual, 2002

[3] Система организации соревнований по программированию http://imcs.dvfu.ru/cats

[4] Рожков М., Кленин А.С. Дипломная работа «Система автоматического тестирования программ и организации соревнований по программированию». — ДВГУ, 2004г.

[5] Матвиенко В., Кленин А.С.Курсовая работа «Рендеринг математических выражений в MathML и HTML». — ДВГУ, 2005г.

[6] Коновалова Д., Кленин А.С. Курсовая работа «Поиск сходных алгоритмических конструкций в программном коде». — ДВГУ, 2005г.

[7] Перепечин В.В., Кленин А.С. Курсовая работа «AJAX-интерфейс для системы CATS». — ДВГУ, 2007г.

[8] Туфанов И.Е., Кленин А.С. Курсовая работа «Универсальный генератор тестов для системы CATS». — ДВГУ, 2008г.

[9] Храпченков П.Ф., Кленин А.С. Курсовая работа «Модуль контролируемого исполнения программ Spawner». — ДВФУ, 2013г.

[10] Vassilis Prevelakis, Diomidis Spinellis «Sandboxing Applications», 2001

[11] Модуль контролируемого исполнения программ Spawner, предыдущая версия http://imcs.dvgu.ru/cats/docs/spawner.html

[12] Всероссийская олимпиада школьников — Методические рекомендации. http://rosolymp.ru/index.php?option=com_content&view=article&id=6451&Itemid=909

[13] Виртуализация: технологические подходы http://www.pcmag.ru/solutions/detail.php?ID=34643

[14] [PATCH] seccomp: secure computing support http://git.kernel.org/cgit/linux/kernel/git/tglx/history.git/commit/?id=d949d0ec9c601f2b148bed3cdb5f87c052968554

[15] Система проведения соревнований EJudge http://ejudge.ru/

[16] Модуль контролируемого исполнения программ PCMS2 http://neerc.ifmo.ru/trains/information/software.html

[17] Система для проведения турниров и индивидуального решения задач по олимпиадному программированию http://www.contester.ru/

[18] Автоматизированная сетевая тестирующая система для проведения турниров по программированию по правилам ACM http://acmtest.ru/

[19] PC^2(Programming Contest Control System) http://www.ecs.csus.edu/pc2/

[20] Олимпиады по программированию http://olympiads.ru/school/system/index.shtml

[21] Автоматизированная тестирующая система, для проведения соревнований по программированию http://domjudge.sourceforge.net/

[22] Универсальная система для проведения олимпиад по программированию http://code.google.com/p/dudge/

[23] Модуль контролируемого исполнения программ Spawner https://github.com/ShigiDono/Spawner

[24] Free Software Foundation, Inc. GNU General Public License. http://www.gnu.org/licenses/gpl.html. - 2007г.

[25] Официальный сайт MSDN. Статья о JOB_OBJECT. http://msdn.microsoft.com/en-us/library/windows/desktop/ms684161%28v=vs.85%29.aspx

Автор работы _____ (Ф.И.О.)
(подпись)

Квалификационная работа допущена к защите

Назначен рецензент

_____
(Фамилия, И.О. рецензента, ученая степень, ученое звание)

Зав. кафедрой информатики,
математического и компьютерного моделирования А. Ю. Чеботарев

Дата «____»_____ 2014 г.