УДК 004.2

Костюк Д.А., Четверкина Г.А., Жук А.М., Сидорович Т.П.

МЕТОДИКА ПЕРЕНОСА ИЗУЧЕНИЯ НИЗКОУРОВНЕВОГО ПРОГРАММИРОВАНИЯ И ВЫЧИСЛИТЕЛЬНОЙ АРХИТЕКТУРЫ НА ПЛАТФОРМУ GNU/LINUX

Введение. Изучение низкоуровневого программирования важно для студентов специальностей информатики и радиоэлектроники с двух позиций. В первую очередь, знание машинных команд актуально при разработке драйверов устройств и других аппаратно-зависимых частей операционных систем, при создании компиляторов и интерпретаторов языков программирования, при решении задач, требующих жесткой оптимизации критических участков кода. Кроме того, без знания языка ассемблера и архитектуры ЭВМ невозможно в достаточной степени освоить принципы работы вычислительной техники, а пробелы в данной области служат источником неоптимального и уязвимого кода на языках высокого уровня, приводят к заведомо неверным архитектурным решениям при разработке программных систем [1].

По данным ресурса http://ohloh.net, занимающегося сбором статистики по приблизительно трем миллиардам строк кода двухсот тысяч разработчиков, в 2009 г. код на ассемблере содержался в 1 352 активных проектах (т.е. таких, частота изменения ассемблерного кода в которых в течение года составляет не менее 1 строки в месяц). При этом общее число строк на ассемблере в рассмотренных проектах равнялось 40 223 602 (31 593 665 без учета комментариев). В целом, как показывает рис. 1, доля проектов, использующих ассемблерный код, в последнее десятилетие снижается – из-за более активного использования языков программирования высокого уровня в сфере микроконтроллеров, а также в связи с меньшим вниманием к оптимизации кода в прикладных проектах. Тем не менее за счет экстенсивного развития информационных технологий, общее число проектов, активно использующих ассемблер, также демонстрирует рост. Некоторый спад числа активных проектов, связанных с низкоуровневым программированием, наблюдается на момент экономической рецессии 2009 г., однако в это же время перестает уменьшаться их доля в общем числе проектов, показывая устойчивость на фоне кризиса.

На текущий момент можно выделить две объективные проблемы в изучении низкоуровневого программирования и архитектуры вычислительных систем. Во-первых, рост числа актуальных программных технологий и языков не позволяет выделять данной тематике значительный процент учебной нагрузки. Во-вторых, крайне архаична базовая платформа DOS, массово используемая из-за невозможности уложить в отведенное число часов курс низкоуровневого программирования под Windows.

Практикумы по архитектуре вычислительных систем и их сопряжению с периферийным оборудованием для студентов-программистов также строятся преимущественно на базе 16-битной платформы Intel и DOS. Помимо исторических факторов, переход к актуальным 32 и 64-битным платформам осложнен повышенным уровнем их абстракции от оборудования и изоляцией прикладных процессов [1].

Ниже нами представлен опыт перевода практических курсов по низкоуровневому программированию и архитектуре ЭВМ на платформу ОС GNU/Linux, выполненного в 2009-2011 гг. на кафедре электронных вычислительных машин и систем БрГТУ для студентов специальности «Вычислительные машины, системы и сети».

1. Общие вопросы адаптации к требованиям платформы. Программирование на ассемблере нельзя назвать типичным для Linux, но как платформа для его изучения эта ОС показывает себя удачным решением. Она проста в программировании, предоставляет доступ к исходному коду, демонстрирует рост в ряде сегментов рынка.

С т.з. ассемблера Linux характеризуется строго унифицированным интерфейсом системных вызовов, одинаковым для всех функций ядра (в отличие от многообразия и непредсказуемости интерфейсов системных функций DOS). На начальном этапе упрощает освоение материала и то, что студенты к моменту изучения ассемблера владеют языком С (что закреплено отечественными образовательными стандартами для технических вузов). Унаследованные традиции ОС Unix и стандарта POSIX облегчают освоение многих системных вызовов Linux, интерфейс которых идентичен стандартным функциям С и, таким образом, априори знаком студентам.

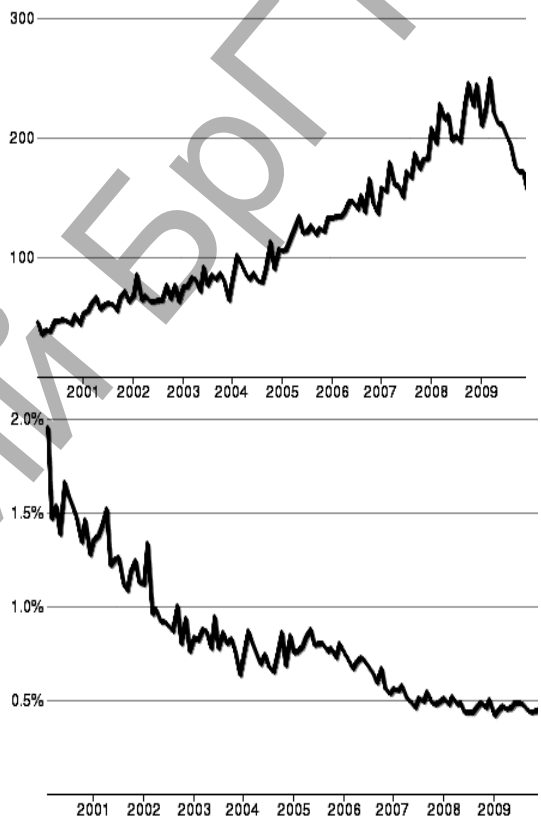

Рис. 1. Динамика использования ассемблера: число активных проектов и их доля от общего количества

Вместе с тем в углубленном курсе низкоуровневого программирования различия аппаратных архитектур вносят дополнительные сложности. Поскольку в современных ОС прикладной процесс не имеет доступа к аппаратным ресурсам, становится невозможным писать простые учебные программы в стиле DOS, свободно взаимодействующие с устройствами.

GNU/Linux позволяет частично решить проблему доступа к системным ресурсам эскалацией привилегий прикладной программы для доступа к специальным механизмам, встроенным в ОС; однако для полноценного практического знакомства с аппаратной архитектурой студенту необходимо научиться создавать модули ядра, и в первую очередь драйвера устройств.

Задача создания драйверов для ядра Linux – относительно проста в сравнении с разработкой драйверов для других современных ОС, как в отношении требуемой квалификации, так и по наличию в свободном доступе необходимых пособий [2]. В этом случае частью учебного процесса также становится исследование исходного кода

Четверкина Галина Андреевна, *старший преподаватель кафедры ЭВМиС Брестского государственного технического университета.*
Сидорович Татьяна Петровна, *старший преподаватель кафедры ЭВМиС Брестского государственного технического университета.*
Беларусь, БрГТУ, 224017, г. Брест, ул. Московская, 267.
Жук Алексей Михайлович*, инженер-программист ИООО «Эрикполь-Брест».*

стандартных драйверов, позволяющее лучше понять архитектуру как программной, так и аппаратной подсистемы, и приобрести выгодную, с точки зрения рынка труда, квалификацию.

2. Уровни абстракции доступа при изучении архитектуры ЭВМ. Курс изучения архитектуры вычислительных систем следует в учебном процессе за курсом низкоуровневого программирования, поскольку без знания принципов программирования на уровне машинных команд невозможно практическое освоение предмета. Кроме того, конкретный перечень системных и периферийных устройств, задействованных в курсе, зависит от выделенных часов и от владения студентами системным программированием под GNU/Linux. Необходимый минимум кроме основ ассемблера включает навыки работы в консоли Unix-подобных ОС и владение языком С.

Традиционно при изучении 16-битной архитектуры системы программирование устройств может выполняться двумя способами: более простым, с использованием прерываний (системных вызовов) BIOS, и более сложным, через порты ввода/вывода соответствующих микроконтроллеров.

Целесообразность обращения к функциям BIOS в GNU/Linux является, по меньшей мере, спорной. Согласно общедоступной информации, ядро не использует их в своей работе; однако предоставляемые BIOS обработчики прерываний технически доступны и могут использоваться при написании модулей ядра – по крайней мере, наименее разрушительные из них. Например, можно относительно безопасно использовать функции BIOS, читающие системные данные из области CMOS, или взаимодействующие с контроллером клавиатуры. Однако при адаптации курса практического изучения архитектуры ЭВМ разумно по возможности ограничить уделяемое данным методам внимание. Анализ большинства изучаемых функций BIOS показывает, что либо их практическая польза ограничена, либо они небезопасны на действующей системе – как, например, функции доступа к дискам, востребованные в загрузчике ОС.

Альтернатива – доступ к устройствам через порты ввода/вывода – в ряде случаев оказывается ненамного сложнее (как упоминавшееся выше взаимодействие с контроллером клавиатуры) и, в свою очередь, может относительно безопасно выполняться на двух уровнях абстракции:

- файл виртуального устройства /dev/port отображает пространство портов и при наличии необходимых прав доступа позволяет пользовательскому процессу выполнять чтение и запись в порты средствами файлового ввода/вывода (функциями open, close, read, write);
- доступ к портам на уровне модуля ядра может быть выполнен классическими ассемблерными инструкциями in и out, непосредственно из ассемблерного кода модуля ядра либо из ассемблерной вставки в тексте программы на С.

Для ряда устройств, обычно неизучаемых в 16-битных практикумах, разумно воспользоваться более высокой абстракцией, предоставляемой виртуальными файловыми системами /dev и /proc. Это дает оправданную экономию, позволяя познакомиться с теми объектами, для которых в противном случае в рамках практикума не нашлось бы учебных часов. В простейшем случае доступ к файловой системе /proc позволяет ознакомиться с более полной конфигурацией оборудования (в сравнении с ограниченными возможностями, предоставляемыми традиционным для классического курса анализом области CMOS). В качестве более сложного примера можно упомянуть файловый доступ к устройствам /dev/dsp и /dev/mixer (выполняемый все теми же системными вызовами read, write и ioctl), который при всей простоте реализации позволяет изучить принципы работы со звуковой картой на примере разработки программы, анализирующей принимаемые аудио-трактом сигналы. Аналогична ситуация с изучением взаимодействия по шине USB.

Точкой схождения двух уровней абстракции может быть написание модуля ядра, связанного с созданием собственного файла устройства в каталоге /dev. Подобная работа проясняет, как осуществляется резервирование аппаратных ресурсов (портов, прерываний) и связь действий над ними с соответствующими операциями на уровне файловой абстракции.

3. Выбор инструментов и способов эскалации привилегий. При выборе программного инструментария для выполнения лабораторных работ был проанализирован ряд ассемблеров, доступных для Linux. Критериями выбора являлись степень документированности, качество поддержки, удобство синтаксиса и использования утилит, портируемость программного кода. Анализ показал преимущества ассемблера NASM (Netwide Assembler) по совокупности приведенных показателей. В качестве средства отладки выбран отладчик GDB (GNU Debugger), активно применяемый при разработке программного обеспечения встраиваемых систем и микроконтроллеров.

При организации практикумов требуется учитывать, что необходимая эскалация привилегий накладывает дополнительные ограничения на рабочую среду, в которой должны выполняться задания.

При апробации лабораторного практикума по программированию на ассемблере нами был использован терминальный сервер под управлением GNU/Linux. Однако при низкоуровневом программировании устройств этот вариант неприемлем: для запуска модулей ядра или доступа к узлам вычислительной системы из пространства пользователя студенты нуждаются в правах администратора, т.е. получают возможность легко нарушить целостность и работоспособность ОС. Проведенные нами эксперименты показали, что решения на базе виртуализации для платформы x86 либо не обеспечивают необходимую точность эмуляции устройств в нестандартных режимах их использования (системы QEMU, VirtualBox), либо обладают недостаточными для комфортной работы производительностью и функционалом (эксперимент демонстрирует точную работу системы виртуализации Bochs ценой 20-кратного снижения производительности по сравнению с выполнением кода непосредственно на аппаратуре) [3].

Поэтому целесообразной альтернативой виртуализованным окружениям является либо использование специализированного LiveCD-подобного дистрибутива, либо фрагментарная эскалация привилегий путем наделения группы студентов правами на запуск ряда системных утилит (например, таких, как утилита загрузки модуля ядра) через систему sudo и групповые права доступа к файлам.

В ряде публикаций можно встретить упоминания о сложностях, связанных с разрозненностью и устареванием технической документации по утилитам и ядру Linux. Очевидно, что преподавательский состав нуждается в более подробных источниках, чем предлагаемые студентам методические материалы; кроме того, не все свободно владеют английским языком. В свою очередь, вопросы устаревания информационных источников характерны для любой динамически развивающейся системы, и в рамках платформы GNU/Linux проявляются в основном при изучении ядра ОС. Поэтому при использовании свободно распространяемых руководств необходимо обращать внимание на дату публикации и избегать источников, ссылающихся на ядра с номерами версий, значительно отличающихся от используемых в учебном процессе. Можно отметить актуальность качественных переводных руководств по ряду утилит GNU (системе сборки make и отладчику gdb), а также транслятору NASM [2]. В целом из наиболее полезных в плане актуальной информации сетевых порталов можно назвать ресурс http://opennet.ru и сайт российско-белорусской программы «СКИФ» http://skif.bas-net.by.

Однако рассмотренные сложности не являются блокирующими, а апробация разработанных курсов показывает высокую усвояемость материала студентами и легкую адаптацию преподавательского состава.

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. Костюк, Д.А. Выполнение лабораторного практикума по основам языка ассемблера на платформе GNU/Linux / Д.А. Костюк, А.М. Жук // Информационные технологии в образовании: материалы международной НПК. – Мн.: БНТУ, 2009. – С. 84–88.
2. Костюк, Д.А. Изучение низкоуровневого программирования и вычислительной архитектуры на базе платформы GNU/Linux // Свободное программное обеспечение в высшей школе: тезисы докладов V конференции. – М.: Институт логики, 2010. – С. 32–35.
3. Костюк, Д.А. Применение платформы GNU/Linux для изучения архитектуры ЭВМ / Д.А. Костюк, А.М. Жук // Современные проблемы радиотехники и телекоммуникаций: матер. 6-й международной молодежной научно-техничической конференции. – Севастополь: СевГУ, 2010. – С. 511.

Материал поступил в редакцию 25.11.11

KOSTIUK D.A., CHETVIORKINA G.A., JUK A.M., SIDOROVICH T.P. Migration of the low-level programming and computer architecture courses to GNU/Linux

The specifics of GNU/Linux operating system usage to teach students low-level programming and computer architecture are investigated. Difficulties related to the platform differences are analyzed, as far as choice of the software and available information sources. Approaches to the arising problems are discussed based on practical experience of implementing the subject in academic activity.

УДК 681.3

Муравьев Г.Л., Хвещук В.И.

О ПОСТРОЕНИИ СИСТЕМ ОБУЧЕНИЯ КОНСТРУИРОВАНИЮ ПРОГРАММ

Введение. Правильно организованное обучение с использованием различных компьютерных сред дает опыт работы с общими принципами, методами разработки и программирования, позволяет адаптировать приобретенные навыки при освоении других средств разработки ПО. К настоящему времени сформулированы проблемы, принципы, требования к обучению в рассматриваемой области, отраженные в стандартах, программах изучения информатики. Это, например, программы ECDL по профессиональной сертификации навыков владения персональным компьютером и повышения компетентности специалистов в сфере ИТ, представленные на сайтах http://www.ecdl.org/programmes, http://www.ecdl.su, учебные планы по информатике ICF-2000, разработанные Международной федерацией по обработке информации IFIP под эгидой ЮНЕСКО и др. [1], где сформулированы такие направления, как обучение проектированию программ на подробно комментированных образцах, на принципах доказательного программирования, методом пошаговой разработки и др. Обучение конструированию программ предполагает выработку специальных навыков работы, включая спецификацию артефактов, проектирование архитектуры, спецификацию алгоритмов решаемых задач, кодирование на языках высокого уровня (ЯВУ), тестирование спецификаций, документирование и т.д. [2].

Таблица. Характеристики систем

№	Характеристика	вид системы			
		1	2	3	4
1	многооконность	+	+	-	+
2	выделение синтаксиса	+	*	-	+
3	средства структурирования текста	+	+	-	*
4	поддержка схем иерархии	+	-	-	-
5	средства спецификации	+	*	-	*
6	русифицированность	+	+	+	-
7	склоняемость имен	+	-	-	-
8	синонимы команд	+	-	-	-
9	пошаговая отладка	+	+	*	+
10	точки останова	+	-	-	+
11	наблюдение диаграмм переменных	*	-	-	-
12	создание exe-модуля	+	-	-	+
13	генерация текста на ЯВУ	+	-	-	-
14	библиотеки эталонных примеров	+	*	*	+
15	средства тестирования	+	-	-	-
16	средства верификации	*	-	-	-

Характеристика систем обучения. Наряду с широко используемыми в целях обучения системами программирования существует много специальных систем, автоматизирующих обучение.

Например, среды на базе языка Лого, системы программирования Комплекта Учебных МИРов (КуМир), тренажеры, web-визуализаторы тренажеров и др. Однако конструирование программ в части работы со спецификациями, архитектурой, модульного проектирования, прототипирования, генерации каркасов обучающими средами поддержаны слабо. Как правило, отсутствуют отладчики спецификаций, нет средств генерации текстов на языках высокого уровня (ЯВУ). Оценка корректности проектных решений, спецификаций, поиск и локализация ошибок, тесно связанные с обучением конструированию программ, разработке алгоритмов, как правило, производится вручную. Используемые лингвистические средства обеспечивают слабую документированность и читаемость кодов.

Результаты сравнительного анализа базовых характеристик систем, автоматизирующих обучение, представлены в таблице. Здесь в графе 1 приведены требования к характеристикам перспективной системы, в графе 2 – характеристики системы КуМИР, в графе 3 – характеристики типовой системы обучения, полученные по “усредненным” данным таких систем, как [3–7] и др., в графе 4 приведены характеристики типовой системы программирования. Это базовые характеристики средств редактирования и структурирования (1–5), возможностей комментирования (6–8), отладки (9–11), средств генерации (12, 13), тестирования (14–16). Символ ”+” означает наличие средства в системе, “*” – желательность средства для систем первого типа и частичность наличия в системах остальных типов. Системы 2–4 отличаются использованием данных средней типизацией, а для системы первого типа достаточно использование стандартных скалярных и структурных типов, в том числе имеющих математические аналоги. В системах 2–3 используются строчные, в 4 – строчные и фрагментарные комментарии, а в системе первого типа необходимы строчные, фрагментарные и ”встраиваемые” в текст комментарии. Исполнимость систем 2–3 основана на интерпретации спецификаций, что достаточно и для системы первого типа.

Структура и функционирование системы. В практике разработки ПО прототипи́рование является этапом получения пробных версий программ для оценки принимаемых решений и согласования их с заказчиком. Оно предусматривает: спецификацию требований, сценариев работы; разработку модульного каркаса проекта, прототипа интерфейса с акцентом на отработку общего управления; – изучение прототипа, тестирование спецификаций и т.д. Перечисленное хорошо согласуется с проблемами обучения, а качество обучения разработке алгоритмов определяется свойствами используемых лингвистических средств и наличием среды, обеспечивающей их исполнимость.

Соответственно качество обучения, его эффективность могут быть в значительной мере повышены использованием специальных компьютерных сред (подобных системам программирования, системам автоматизации проектирования), базирующимся на принципе прототипирования программ, что обеспечивает по аналогии с указанными средствами системность обучения, исполнимость, проверяемость разрабатываемых спецификаций.

Система обучения должна содержать [8]: – в части программного обеспечения средства прототипирования, средства разработки алгоритмов, управления базой данных, сервисные средства; – в части лингвистического обеспечения языки спецификаций, языковые средства проектирования алгоритмов [9]; – в части информационного обеспечения базы данных проектных решений.

Муравьёв Геннадий Леонидович, *к.т.н., профессор кафедры интеллектуальных информационных технологий Брестского государственного технического университета.*

Хвещук Владимир Иванович, *к.т.н., доцент, профессор кафедры интеллектуальных информационных технологий Брестского государственного технического университета.*

Беларусь, БрГТУ, 224017, г. Брест, ул. Московская, 267.