Отчёт по лабораторной работе №9

Дисциплина: архитектура компьютеров и операционные системы

Пестова Ева Констнатиновна

Содержание

# 1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программам с помощью GDB.
3. Добавление точек останова.
4. Работа с данными программы в GDB.
5. Обработка аргументов командной строки в GDB.
6. Задания для самостоятельной работы.

# 3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB.

Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено y (то есть «да»), отладка программы прекращается. Командой run её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются.

Для выхода из отладчика используется команда quit (или сокращённо q).

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g.

Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой info (кратко i).

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable.

Обратно точка останова активируется командой enable.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete.

Для продолжения остановленной программы используется команда continue (c). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число N, которое указывает отладчику проигнорировать N − 1 точку останова (выполнение остановится на N-й точке).

Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

# 4 Выполнение лабораторной работы

1. Реализация подпрограмм в NASM

Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm (рис. [??]).

![Создание файлов для лабораторной работы](data:application/octet-stream;base64,)

Создание файлов для лабораторной работы

Ввожу в файл lab09-1.asm текст программы с использованием подпрограммы из листинга 9.1 (рис. [??]).

![Ввод текста программы из листинга 9.1](data:application/octet-stream;base64,)

Ввод текста программы из листинга 9.1

Создаю исполняемый файл и проверяю его работу (рис. [??]).

![Запуск исполняемого файла](data:application/octet-stream;base64,)

Запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x) = 2x + 7, g(x) = 3x − 1 (рис. [??]).

![Изменение текста программы согласно заданию](data:application/octet-stream;base64,)

Изменение текста программы согласно заданию

Создаю исполняемый файл и проверяю его работу (рис. [??]).

![Запуск исполняемого файла](data:application/octet-stream;base64,)

Запуск исполняемого файла

1. Отладка программам с помощью GDB

Создаю файл lab09-2.asm с текстом программы из Листинга 9.2 (рис. [??]).

![Ввод текста программы из листинга 9.2](data:application/octet-stream;base64,)

Ввод текста программы из листинга 9.2

Получаю исполняемый файл для работы с GDB с ключом ‘-g’ (рис. [??]). ![Получение исполняемого файла](data:application/octet-stream;base64,)

Загружаю исполняемый файл в отладчик gdb (рис. [??]).

![Загрузка исполняемого файла в отладчик](data:application/octet-stream;base64,)

Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив ее в оболочке GDB с помощью команды run (рис. [??]).

![Проверка работы файла с помощью команды run](data:application/octet-stream;base64,)

Проверка работы файла с помощью команды run

Для более подробного анализа программы устанавливаю брейкпоинт на метку \_start и запускаю её (рис. [??]).

![Установка брейкпоинта и запуск программы](data:application/octet-stream;base64,)

Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start, и переключаюсь на отображение команд с синтаксисом Intel, введя команду set disassembly-flavor intel (рис. [??]).

![Использование команд disassemble и disassembly-flavor intel](data:application/octet-stream;base64,)

Использование команд disassemble и disassembly-flavor intel

В режиме ATT имена регистров начинаются с символа %, а имена операндов с $, в то время как в Intel используется привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы с помощью команд layout asm и layout regs (рис. [??]).

![Включение режима псевдографики](data:application/octet-stream;base64,)

Включение режима псевдографики

1. Добавление точек останова

Проверяю, что точка останова по имени метки \_start установлена с помощью команды info breakpoints и устанавливаю еще одну точку останова по адресу инструкции mov ebx,0x0. Просматриваю информацию о всех установленных точках останова (рис. [??]).

![Установление точек останова](data:application/octet-stream;base64,)

Установление точек останова

1. Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi.

Просматриваю значение переменной msg1 по имени с помощью команды x/1sb &msg1 и значение переменной msg2 по ее адресу (рис. [??]).

![Просмотр значений переменных](data:application/octet-stream;base64,)

Просмотр значений переменных

С помощью команды set изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2 (рис. [??]).

![Использование команды set](data:application/octet-stream;base64,)

Использование команды set

С помощью команды set изменяю значение регистра ebx в соответствии с заданием (рис. [??]).

![Использование команды set для изменения значения регистра](data:application/octet-stream;base64,)

Использование команды set для изменения значения регистра

Разница вывода команд p/s $ebx отличается тем, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется

1. Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm с программой из листинга 8.2 в файл с именем lab09-3.asm и создаю исполняемый файл (рис. [??]).

![Создание файла](data:application/octet-stream;base64,)

Создание файла

Загружаю исполняемый файл в отладчик gdb, указывая необходимые аргумен- ты с использованием ключа –args (рис. [??]).

![Загрузка файла с аргументами в отладчик](data:application/octet-stream;base64,)

Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запус- каю ее (рис. [??]).

![Установление точки останова и запуск программы](data:application/octet-stream;base64,)

Установление точки останова и запуск программы

Посматриваю вершину стека и позиции стека по их адресам (рис. [??]).

![Просмотр значений, введенных в стек](data:application/octet-stream;base64,)

Просмотр значений, введенных в стек

1. Задания для самостоятельной работы
2. Преобразовываю программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму (рис. [??]).

![Написание кода подпрограммы](data:application/octet-stream;base64,)

Написание кода подпрограммы

Запускаю код и проверяю, что она работает корректно (рис. [??]).

![Запуск программы и проверка его вывода](data:application/octet-stream;base64,)

Запуск программы и проверка его вывода

1. Ввожу в файл lab09-5.asm текст программы из листинга 9.3 (рис. [??]).

![Ввод текста программы из листинга 9.3](data:application/octet-stream;base64,)

Ввод текста программы из листинга 9.3

При корректной работе программы должно выводится “25”. Создаю исполняемый файл и запускаю его (рис. [??]).

![Создание и запуск исполняемого файла](data:application/octet-stream;base64,)

Создание и запуск исполняемого файла

Видим, что в выводе мы получаем неправильный ответ. Получаю исполняемый файл для работы с GDB, запускаю его и ставлю брейкпоинты для каждой инструкции, связанной с вычислениями. С помощью команды continue прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров. При выполнении инструкции mul ecx происходит умножение ecx на eаx, то есть 4 на 2, вместо умножения 4 на 5 (регистр ebx). Происходит это из-за того, что стоящая перед mov ecx,4 инструкция add ebx,eax не связана с mul ecx, но связана инструкция mov eax,2.Из-за этого мы получаем неправильный ответ (рис. [??]).

![Неверное изменение регистра](data:application/octet-stream;base64,)

Неверное изменение регистра

Исправляем ошибку, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx (рис. [??]).

![Исправление ошибки](data:application/octet-stream;base64,)

Исправление ошибки

Создаем исполняемый файл и запускаем его. Убеждаемся, что ошибка исправлена (рис. [??]).

![Ошибка исправлена](data:application/octet-stream;base64,)

Ошибка исправлена

# 5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005 — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.