**Front matter**

title: "Лабораторная работа №13" subtitle: "Средства, применяемые при разработке программного обеспечения в ОС типа UNIX/Linux" author: "Камкина Арина Леонидовна"

**Generic otions**

lang: ru-RU toc-title: "Содержание"

**Bibliography**

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

**Pdf output format**

toc: true #Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt

**I18n polyglossia**

polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english

**I18n babel**

babel-lang: russian babel-otherlangs: english

**Fonts**

mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT Mono mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions: Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase,Scale=0.9

**Biblatex**

biblatex: true biblio-style: "gost-numeric" biblatexoptions:

• parentracker=true

• backend=biber

• hyperref=auto

• language=auto

• autolang=other\*

• citestyle=gost-numeric

**Pandoc-crossref LaTeX customization**

figureTitle: "Рис." tableTitle: "Таблица" listingTitle: "Листинг" lofTitle: "Список иллюстраций" lotTitle: "Список таблиц" lolTitle: "Листинги"

**Misc options**

indent: true header-includes:

• \usepackage{indentfirst}

• \usepackage{float} # keep figures where there are in the text

• \floatplacement{figure}{H} # keep figures where there are in the text

**Цель работы**

Приобрести простейшие навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

**Задание**

1. В домашнем каталоге создайте подкаталог ~/work/os/lab\_prog.

2. Создайте в нём файлы: calculate.h, calculate.c, main.c. Это будет примитивнейший калькулятор, способный складывать, вычитать, умножать и делить, возводить число в степень, брать квадратный корень, вычислять sin, cos, tan. При запуске он будет запрашивать первое число, операцию, второе число. После этого программа выведет результат и остановится.

|  |  |
| --- | --- |
| 3. | С помощью gdb выполните отладку программы calcul (перед использованием gdb исправьте Makefile): |
| • | Запустите отладчик GDB, загрузив в него программу для отладки: gdb ./calcul |

• Для запуска программы внутри отладчика введите команду run:

run

• Для постраничного (по 9 строк) просмотра исходного код используйте команду list:

list

• Для просмотра строк с 12 по 15 основного файла используйте list с параметрами:

list 12,15

• Для просмотра определённых строк не основного файла используйте list с параметрами:

list calculate.c:20,29

• Установите точку останова в файле calculate.c на строке номер 21:

list calculate.c:20,27

break 21

• Выведите информацию об имеющихся в проекте точка останова:

info breakpoints

|  |  |
| --- | --- |
| • | Запустите программу внутри отладчика и убедитесь, что программа остановится в момент прохождения |

точки останова:

run

5

"-"

backtrace

• Отладчик выдаст следующую информацию:

#0 Calculate (Numeral=5, Operation=0x7fffffffd280 "-") at calculate.c:21

#1 0x0000000000400b2b in main () at main.c:17

а команда backtrace покажет весь стек вызываемых функций от начала программы до текущего места.

• Посмотрите, чему равно на этом этапе значение переменной Numeral, введя:

print Numeral

На экран должно быть выведено число 5.

• Сравните с результатом вывода на экран после использования команды:

display Numeral

• Уберите точки останова:

info breakpoints

delete 1

7. С помощью утилиты splint попробуйте проанализировать коды файлов calculate.c и main.c.

**Выполнение лабораторной работы**

1. Создала подкаталог lab\_prog и перешла в него (рис. [-@fig:001])

ПСоздание и переход в каталог { #fig:001 width=70% }

2. Создала в нём файлы: calculate.h, calculate.c, main.c (рис. [-@fig:002])

Создание файлов { #fig:002 width=70% }

3. Выполнила компиляцию программы посредством gcc (рис. [-@fig:003])

Компиляция программы { #fig:003 width=70% }

4. Создала Makefile (рис. [-@fig:004])

Создание Makefile { #fig:004 width=70% }

5. Исправила все ошибки и добавила отступы, где нужно (рис. [-@fig:005])

Итоговый Makefile { #fig:005 width=70% }

6. Выполнила отладку через gdb (рис. [-@fig:006])

[Отладка gdb](file:///app/image/6.png) { #fig:006 width=70% }

7. Запустила программу (рис. [-@fig:007])

Запуск программы { #fig:007 width=70% }

8. Постранично просмотрела исходный код (рис. [-@fig:008])

Постранично код { #fig:008 width=70% }

9. Просмотрела определённый диапазон строчек кода (рис. [-@fig:009])

Диапазон кода { #fig:009 width=70% }

10. Просмотрела определённый диапазон строчек кода неосновного файла (рис. [-@fig:0010])

Диапазон кода неосновного файла { #fig:0010 width=70% }

11. Установила точку останова в файле calculate.c на строке номер 21 (рис. [-@fig:0011])

Установка точки останова { #fig:0011 width=70% }

12. Вывела информацию об имеющихся в проекте точка останова (рис. [-@fig:0012])

Информация о точках останова { #fig:0012 width=70% }

13. Запустила программу внутри отладчика и убедилась, что программа остановится в момент прохождения

точки останова: (рис. [-@fig:0013])

Программа с точкой останова { #fig:0013 width=70% }

14. Посмотрела, чему равно Numeral в разных моментах - они равны (рис. [-@fig:0014])

Numeral { #fig:0014 width=70% }

15. Посмотрела информацию о точке останова (рис. [-@fig:0015])

Информация о точке останова { #fig:0015 width=70% }

16. Удалила точку останова (рис. [-@fig:0016])

Удаление точки останова { #fig:0016 width=70% }

**Выводы**

Я приобрела практические навыки разработки, анализа, тестирования и отладки приложений в ОС типа UNIX/Linux на примере создания на языке программирования С калькулятора с простейшими функциями.

**Контрольные вопросы**

1. Можно использовать стандартные команды для получения справки - man gcc, man make, man gdb ну или вместо утилиты man использовать команду --help

2.

Создание исходного кода (написание в IDE)

Сохранение промежуточных файлов или альтернативных веток разработки исходного кода

Компиляция исходных файлов или их интерпритация в зависимости от выбранного языка программирования и/или системы сборки проектов

Тестирование проекта который был собран

Запись в соответствующую ветку разработки Git (main или dev, по-умолчанию)

3. Суффикс - нужен для определения расширения в контексте файловой системы или компилятора с помощью которого будет производиться компиляция или интерпретация исходного кода в работающую программу.

4. Компилятор Си предназначен для компиляции внутренних файлов системы без полного скачивания программ, а просто скачав исходный код системных утилит и произвести с помощью встроенного компилятора компиляцию системных утилит

5. Утилита make - предназначена для упрощения разработки приложений, путем написания файла конфигурации который описывает пути компиляции для компилятора языка программирования

6. Можно использовать пример из лабараторной работы

7. Пошаговая отладка программ (трассировка) - её суть заключается в пошаговом выполнении каждой

строчки кода

8. Основные команды отладчика gdb:

• backtrace - вывод на экран путь к текущей точке останова.

• break - установить точку останова (строка или функция)

• clear - удалить все точки останова в функции

• ontinue - продолжить выполнение программы

• delete (n) - удалить точку останова

• display - добавить выражение в список выражений, значения которых отображаются при достижении точки останова программы

• finish - выполнить программу до момента выхода из функции

• info breakpoints - вывести на экран список используемых точек останова

• info watchpoints - вывести на экран список используемых контрольных выражений

• list - вывести на экран исходный код (в качестве параметра может быть указано название файла и через двоеточие номера начальной и конечной строк)

• next - выполнить программу пошагово, но без выполнения вызываемых в программе функций • print - вывести значение указываемого в качестве параметра выражения

• run - запуск программы на выполнение

• set[variable] - установить новое значение переменной

• step - пошаговое выполнение программы

• watch - установить контрольное выражение, при изменении значения которого программа будет

остановлена

9. Мои действия при отладке программ: Запустил Makefile -> Начал отладку (run) -> Вывел содержимое main файла -> Установил точку останова в main файле -> Продолжил выполнение (run) -> Использовал команды print & display для вывод промежуточных данных -> Удалил точку останова -> Закончил отладку

10. Нейтральная реация компилятора, т.е. программных ошибок обнаружено не было

11. cppcheck, splint, cscope и другие

12. Проверка корректности аргументов и поиск ошибок и значений в программе которые могут быть улучшены, а также оценка всей программы