# РОCСИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

## Факультет физико-математических и естественных наук

## Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ

## ПО ЛАБОРАТОРНОЙ РАБОТЕ № 14

### *дисциплина: Операционные системы*

Студент: Соболевский Денис Андреевич

Группа: НФИбд-02-20

Преподаватель: Велиева Татьяна Рефатовна

МОСКВА

2021 г.

### 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.

### Задачи:

1. Научиться выполнять компиляцию по средствам командной строки;
2. Освоить отладчик GDB;
3. Научиться анализировать исходные коды.

### Теоретическое введение:

***Этапы разработки приложений***

* планирование, включающее сбор и анализ требований к функционалу и другим характеристикам разрабатываемого приложения;
* проектирование, включающее в себя разработку базовых алгоритмов и спецификаций, определение языка программирования;
* непосредственная разработка приложения:
  + кодирование — по сути создание исходного текста программы (возможно в нескольких вариантах);
  + анализ разработанного кода;
  + сборка, компиляция и разработка исполняемого модуля;
  + тестирование и отладка, сохранение произведённых изменений;
* документирование.

***Компиляция исходного текста и построение исполняемого файла***

Для компиляции, например, файла main.c используют команду:

gcc -c main.c

Если требуется получить исполняемый файл с определённым именем (например, hello), то требуется воспользоваться опцией -o и в качестве параметра задать имя создаваемого файла:

gcc -o hello main.c

С прочими опциями компилятора gcc можно ознакомиться в статье *"Опции компиляторов"*[1].

Для сборки разрабатываемого приложения и собственно компиляции полезно воспользоваться утилитой make. Она позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.

***Тестирование и отладка***

Для использования отладчика GDB необходимо скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле. Для этого следует воспользоваться опцией -g компилятора gcc:

gcc -c file.c -g

После этого для начала работы с gdb необходимо в командной строке ввести одноимённую команду, указав в качестве аргумента анализируемый бинарный файл:

gdb file.o

Затем можно использовать по мере необходимости различные команды gdb.

***Анализ исходного текста программы***

Для анализа кода программы example.c следует выполнить следующую команду:

splint example.c

***В ходе работы*** над понадобится установить утилиту splint на Centos 7, поэтому воспользуемся статьей *"УСТАНОВКА ПАКЕТОВ В CENTOS 7"*[2].

Все коды, которые использовадись во время выполнения работы были взяты из *Лабораторной работы №14*[3].

### Выполнение работы:

1, 2. В домашнем каталоге нам нужно создать подкаталог ~/work/os/lab\_prog.

Создаем каталог os в уже созданном ранее каталоге work и создаем в нем подкаталог lab\_prog - mkdir lab\_prog (*рисунок 1*). Перейдем в него командой cd lab\_prog. Создадим в нём файлы: calculate.h, calculate.c, main.c с помощью текстового редактора emacs (*рисунок 1*). Это будет примитивнейший калькулятор.

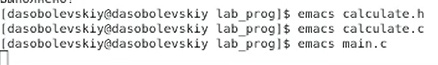


рисунок 1: создание подкаталога ~/work/os/lab\_prog и файлов calculate.h, calculate.c, main.c

В файл calculate.h вводим код на языке программирования С, предоставленный в материалах к ЛР №14 (*рсиунок 2*)

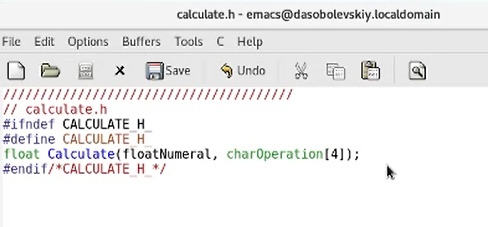


рисунок 2: файл calculate.h

То же делаем с файлами calculate.c (*рисунок 3*) и main.c (*рисунок 4*).

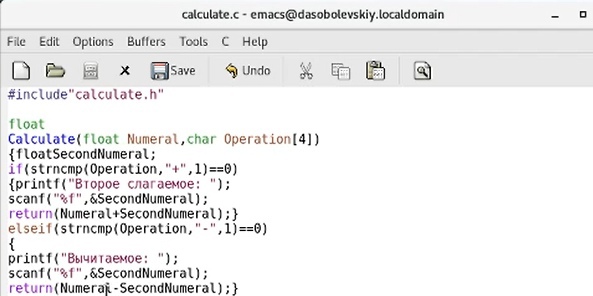


рисунок 3: файл calculate.c

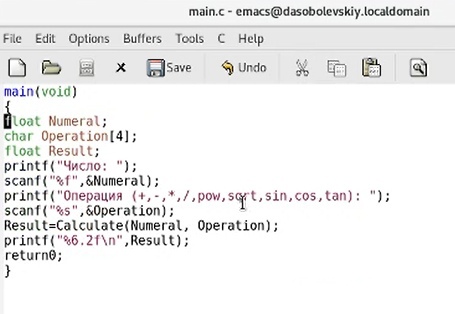


рисунок 4: файл main.c

3, 4. Теперь выполним компиляцию программы посредством gcc, ввода следующие команды (*рисунок 5*):

gcc -c calculate.c  
gcc -c main.c  
gcc calculate.o main.o -o calcul -lm



рисунок 5: компиляция прогарммы посредством gcc

Видим, что система не выдает нам сообщений об ошибках, следовательно код написан правильно, и нам нечего исправлять.

1. Создадим Makefile, который будет расположен в каталоге lab\_prog, поскольку makefile должен находиться в том же месте, где и проект, связанный с ним. Создаем файл с помощью редактора emacs и вводим в него предложенный код файла из лабораторной работы (*рисунок 6*).

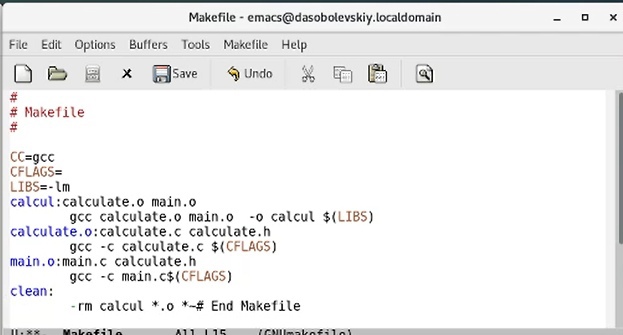


рисунок 6: создание Makefile

1. С помощью gdb выполним отладку программы calcul. Для использования GDB нам необходимо сначала скомпилировать анализируемый код программы таким образом, чтобы отладочная информация содержалась в результирующем бинарном файле (*рисунок 7*). Для этого следует воспользоваться опцией -g компилятора gcc, тогда синтаксис компиляции будет следующим:

gcc -c [имя файла] -g

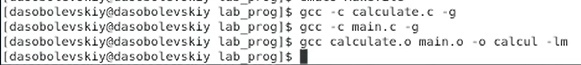


рисунок 7: компиляция перед запуском GDB

* Запустим отладчик GDB, загрузив в него программу для отладки: gdb ./calcul (*рисунко 8*).

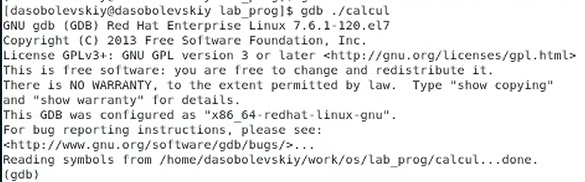


рисунок 8: запуск отладчика GDB

* Теперь нам нужно запустить программу внутри отладчика. Для этого внутри отладчика введем команду ```run```` (*рисунок 9*).

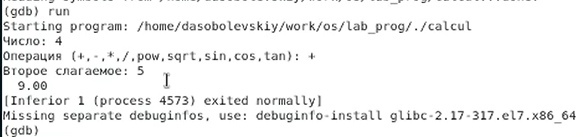


рисунок 9: запуск программы в отладчике

Видим, что программа была успешно запущенна. На ввод нам предлагается ввести какое-либо число (вводим 4), операцию, которая будет производится с ним (в нашем случае это сложение), далее нам предлагается ввести второе слагаемое (5). Результат выделен черным - 9. Программа работает исправно.

* Для постраничного (по 9 строк) просмотра исходного код используем команду list (*рисунок 10*). Видим, что действительно вывелось 9 строк (4-13).

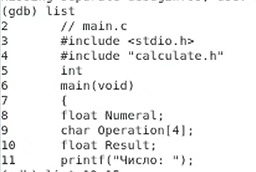


рисунок 10: постраничный вывод list

* Для просмотра строк с 12 по 15 основного файла используем list с параметрами - list 12,15 (*риснуок 11*).

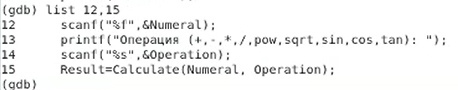


рисунок 11: просмотр определенных строк

* Для просмотра определённых строк не основного файла используем list с параметрами: list calculate.c:20,29 (*рисунок 12*).

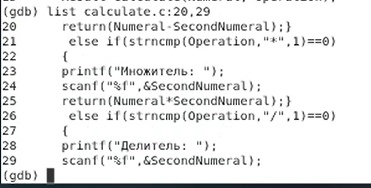


рисунок 12: просмотр определённых строк не основного файла

* Установим точку останова в файле calculate.c на строке номер 21: break 21 (*рисунок 13*).



рисунок 13: установка точки останова

Видим, что точка была успешно установлена.

* Выведем информацию об имеющихся в проекте точка останова. Для этого введем команду info breakpoints (*рисунок 14*).

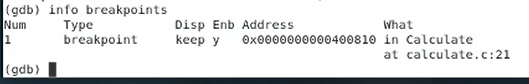


рисунок 14: информация об имеющихся в проекте точка останова

Можем наблюдать информацию о точке, которую мы только что установили: ее номер, тип, адрес, место установки.

* Убираем точки останова. Сначала посмотрим информацию о текущих точках info breakpoints, чтобы узнать номер точки, которую мы собираемся удалить. Далее удаляем ее delete 1, где 1 - номер точки. Снова просматриваем info breakpoints, чтобы убедиться в удлении точки (*рисунок 15*).

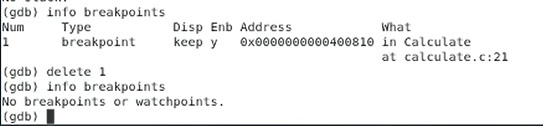


рисунок 15: удаление точки останова

Видим по последнему выводу, что точка была успешно удалена.

1. Теперь с помощью утилиты splint нам нужно проанализировать коды файлов calculate.c и main.c. Для этого сначала установим данную утилиту. (*рисунок 16*).

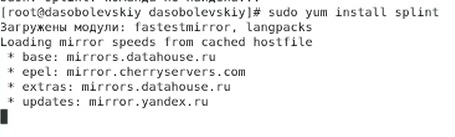


рисунок 16: установка утилиты splint

Анализируем коды файлов calculate.c (*рисунок 17*) и main.c (*рисунок 18*) через splint [имя файла].

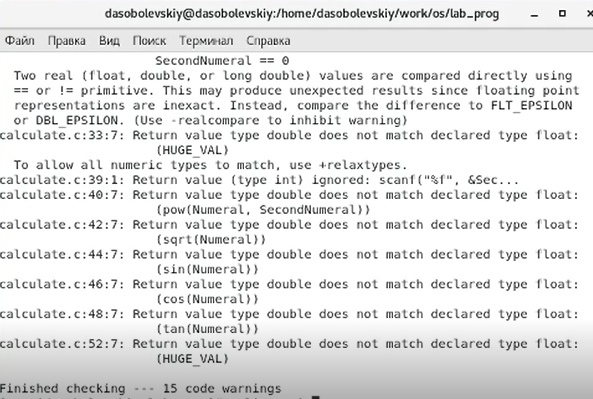


рисунок 17: анализ кода calculate.c

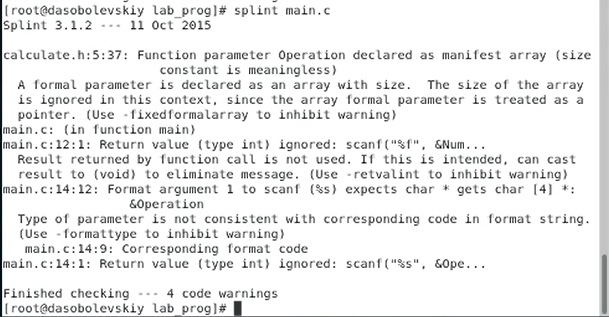


рисунок 18: анализ кода main.c

Видим, что утилита splint анализирует программный код, проверяет корректность задания аргументов использованных в программе функций и типов возвращаемых значений, а также обнаруживает синтаксические и семантические ошибки.

## Вывод:

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

### Библиография:

[1]: [Опции компиляторов](https://parallel.uran.ru/book/export/html/15)

[2]: [УСТАНОВКА ПАКЕТОВ В CENTOS 7](https://losst.ru/ustanovka-paketov-v-centos-7)

[3]: [Лабораторная работа №14](https://esystem.rudn.ru/pluginfile.php/1142386/mod_resource/content/2/011-lab_prog.pdf)