Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Ким Денис Вячеславович

Содержание

# 1 Цель работы

Приобрести навыки написания программ с использованием подпрограмм и познакомиться с методами отладки при помощи GDB и его основными возможностями.

# 2 Задание

В ходе данной работы мне предстоит познакомиться с понятием отладки, методами отладки, основными возможностями отладчика GDB. Также я научусь пользоваться отладкой программ с помощью данного отладчика.

# 3 Теоретическое введение

Здесь описываются теоретические аспекты, связанные с выполнением работы.

Например, в табл. 1 приведено краткое описание стандартных каталогов Unix.

Таблица 1: Описание некоторых каталогов файловой системы GNU Linux

| Имя каталога | Описание каталога |
| --- | --- |
| / | Корневая директория, содержащая всю файловую |
| /bin | Основные системные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям |
| /etc | Общесистемные конфигурационные файлы и файлы конфигурации установленных программ |
| /home | Содержит домашние директории пользователей, которые, в свою очередь, содержат персональные настройки и данные пользователя |
| /media | Точки монтирования для сменных носителей |
| /root | Домашняя директория пользователя root |
| /tmp | Временные файлы |
| /usr | Вторичная иерархия для данных пользователя |

Более подробно про Unix см. в [1–4].

# 4 Выполнение лабораторной работы

Создаём каталог для выполнения лабораторной работы № 9, переходим в него и создаём файл lab9-1.asm: (рис. 1).



Рис. 1: Создание нового каталога и файла в нём

В качестве примера рассмотрим программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы \_calcul. Изучаем его(рис. 2).

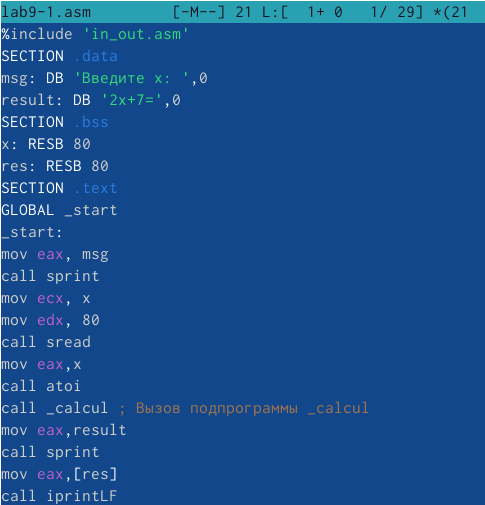


Рис. 2: Текст программы из листинга

Проверяем работу файла: (рис. 3).

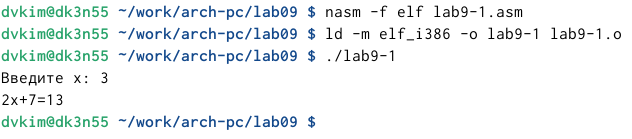


Рис. 3: Запуск программы

Изменяем текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul:(рис. 4).

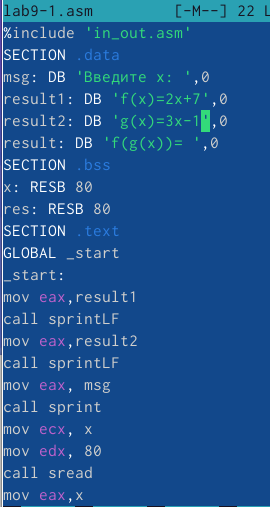


Рис. 4: Изменение текста программы

Проверяем работу файла программы с внесёнными изменениями: (рис. 5).

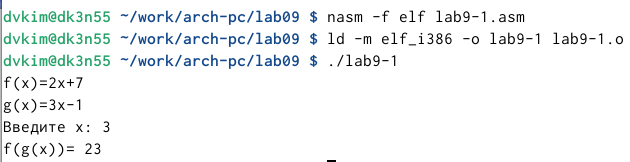


Рис. 5: Запуск программы

Создаём файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!). Загружаем исполняемый файл в отладчик gdb: (рис. 6).

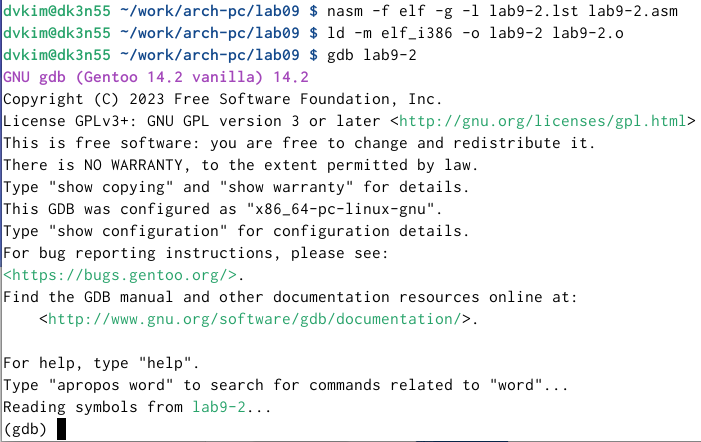


Рис. 6: Загрузка исполняемого файл в отладчик gdb

Проверяем работу программы, запустив ее в оболочке GDB с помощью команды run: (рис. 7).

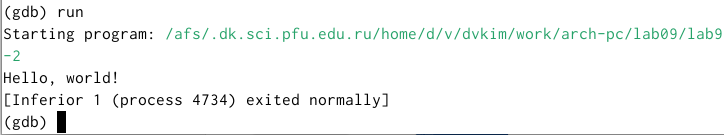


Рис. 7: Проверка работы программы

Для более подробного анализа программы устанавливаем брейкпоинт на метку \_start, с которой начинается выполнение любой ассемблерной программы, и запускаем её. (рис. 8).

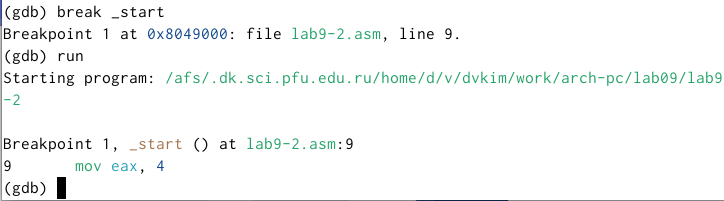


Рис. 8: Установка брейкпоинта и запуск программы

Посмотрим дисассимилированный код программы с помощью команды disassemble, начиная с метки \_start (рис. 9).

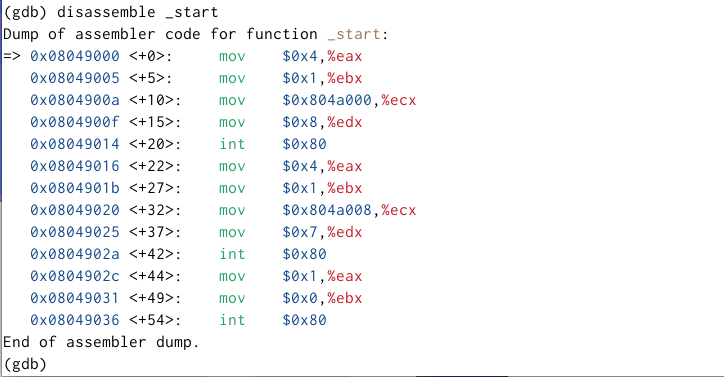


Рис. 9: Дисассимилированный код программы

Переключаемся на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel. Отличия состоят в том, что в дисассимилированном отображении используются % и $, Intel их не использует: (рис. 10).



Рис. 10: Ввод команды set disassembly-flavor intel

Включаем режим псевдографики для более удобного анализа программы (рис. 9.2): (рис. 11).

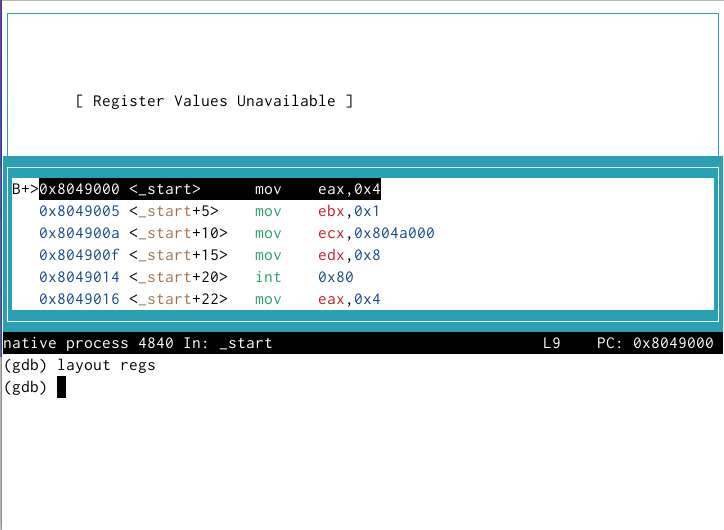


Рис. 11: Включение режима псевдографики

На предыдущих шагах была установлена точка останова по имени метки (\_start). Проверяем это с помощью команды info breakpoints: (рис. 12).

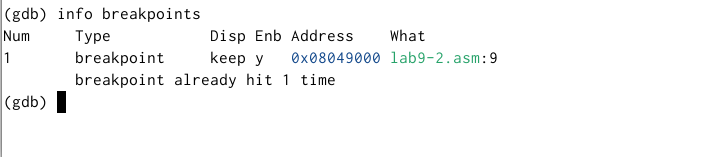


Рис. 12: Ввод команды info breakpoints

Установим еще одну точку останова по адресу инструкции. Определяем адрес предпоследней инструкции (mov ebx,0x0) и устанавливаем точку останова: (рис. 13).

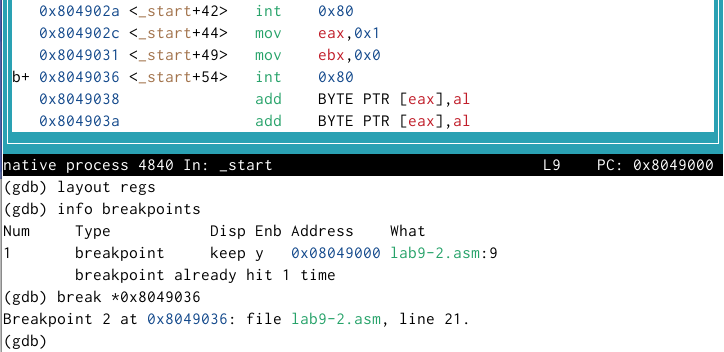


Рис. 13: Установка точки останова

Посмотрим информацию о всех установленных точках останова: (рис. 14).

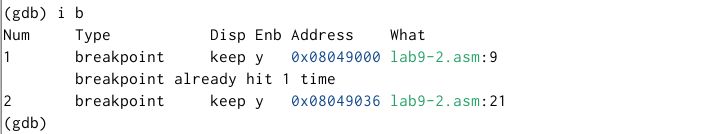


Рис. 14: Ввод команды info breakpoints

Выполняем 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров. (рис. 15).

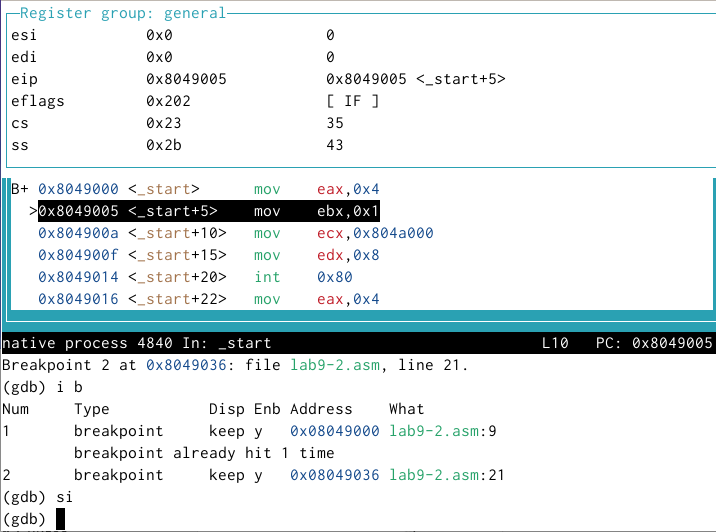


Рис. 15: Использование команды stepi

Посмотреть содержимое регистров также можно с помощью команды info registers: (рис. 16).

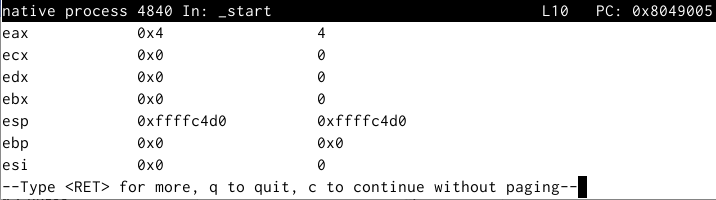


Рис. 16: Ввод команды info registers

Посмотрите значение переменной msg1 по имени: (рис. 17).

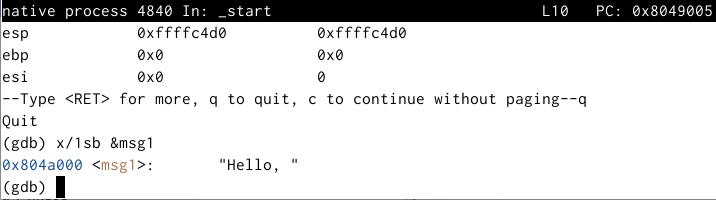


Рис. 17: Просмотр значения переменной msg1

Посмотрим значение переменной msg2 по адресу. Адрес переменной можно определить по дизассемблированной инструкции. В данном случае я просмотрел значение переменной по имени. (рис. 18).

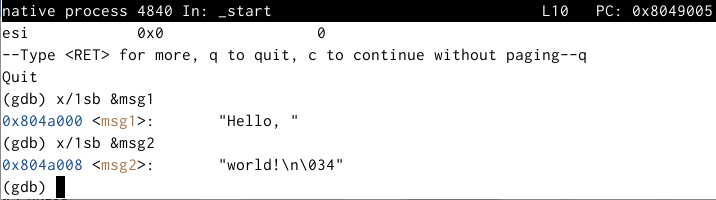


Рис. 18: Просмотр значения переменной msg2

Изменяем первый символ переменной msg1 (рис. 9.5): (рис. 19).



Рис. 19: Изменение первого символа переменной

Заменяем любой символ во второй переменной msg2: (рис. 20).

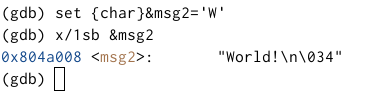


Рис. 20: Изменение первого символа другой переменной

Выводим в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx: (рис. 21).



Рис. 21: Вывод значения регистра edx

С помощью команды set изменяем значение регистра ebx. Разница состоит в том, что команда выводит два разных значания, значения разнятся, так как в первый раз вносится значение 2, а второй регистр равен двум: (рис. 22).

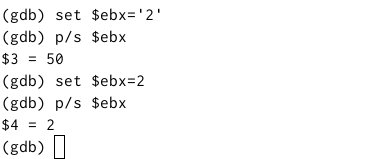


Рис. 22: Изменение значения регистра ebx

Завершаем выполнение программы с помощью команды continue и выходим из GDB с помощью команды quit (рис. 23).



Рис. 23: Завершение выполнения программы и выход

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8: (рис. 24).

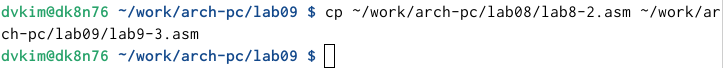


Рис. 24: Копирование файла

Создаём исполняемый файл. (рис. 25).

Рис. 25: Создание исполняемого файла

Рис. 25: Создание исполняемого файла

Загружаем исполняемый файл в отладчик, указав заданные аргументы. Для начала установим точку останова перед первой инструкцией в программе и запустим её: (рис. 26).

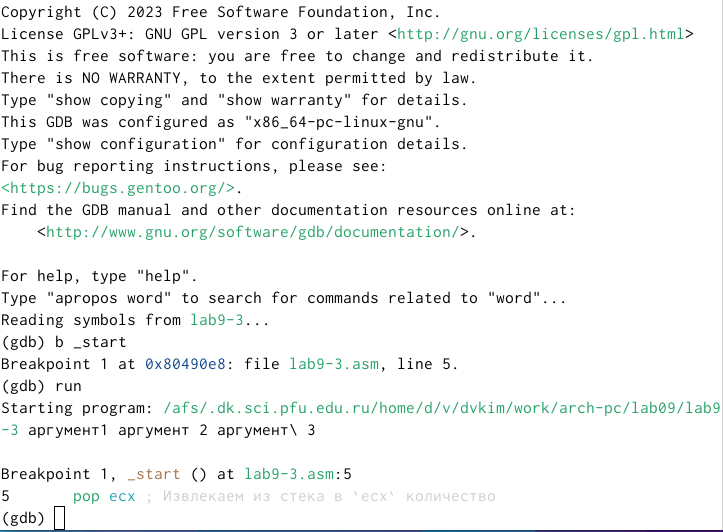


Рис. 26: Загрузка исполняемого файла в отладчик

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число, равное количеству аргументов командной строки. Посмотрим остальные позиции стека – по адресу [esp+4] располагается адрес в памяти, где находится имя программы, по адесу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] – второго и т.д. Элементы расположены с интервалом в 4, так как стек может хранить до 4 байт: (рис. 27).

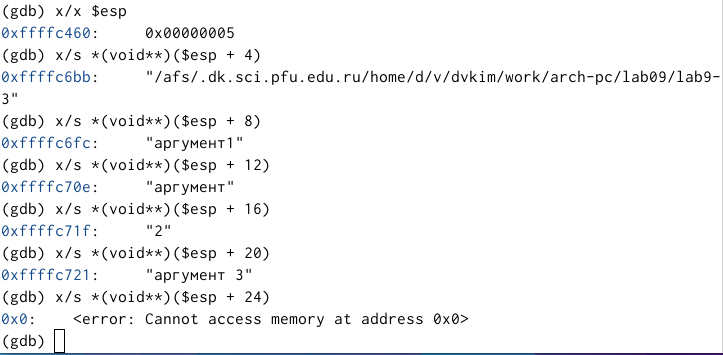


Рис. 27: Просмотр остальных позиций стека

Выполняем задания для самомстоятельной работы. Преобразуем программу из лабораторной работы №8, реализовав вычисление значения функции f(x) как подпрограмму: (рис. 28).

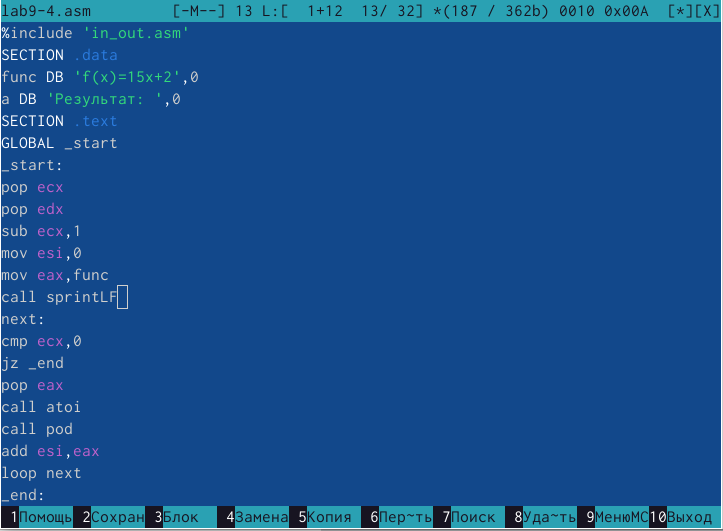


Рис. 28: Изменение текста программы

Проверяем правильность: (рис. 29).

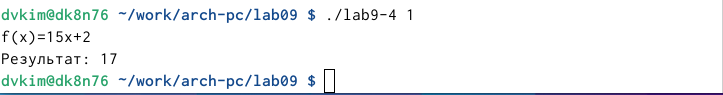


Рис. 29: Запуск программы

В листинге 9.3 приведена программа вычисления выражения (3 + 2) ∗ 4 + 5. При запуске данная программа дает неверный результат - выражение равно 25, но программа выдаёт 10: (рис. 30).

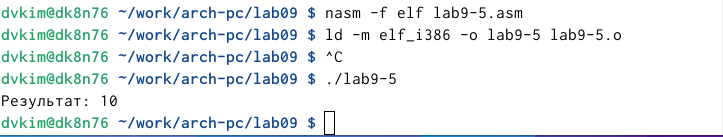


Рис. 30: Получение неправильного ответа

Запускаем отладчик и анализируем: (рис. 31).

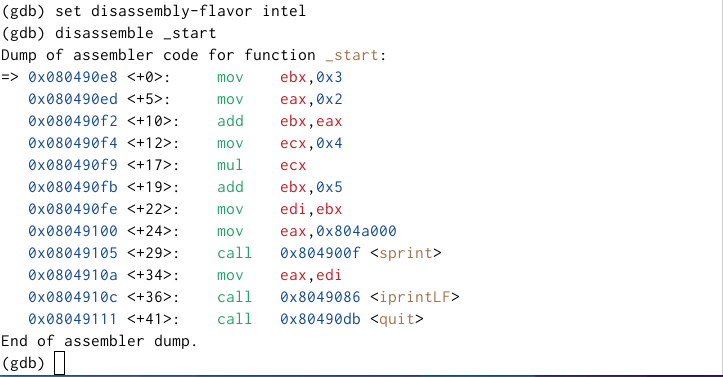


Рис. 31: Запуск отладчика

Исправляем ошибки в регистрах (например, в одной строчке они перепутаны): (рис. 32).

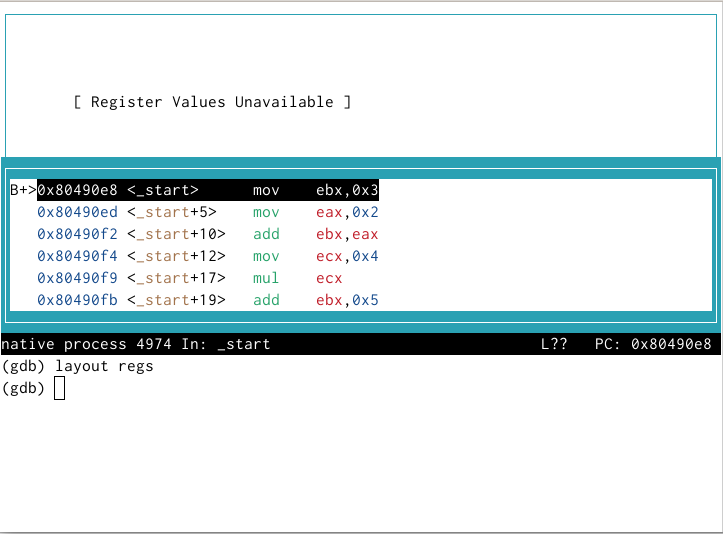


Рис. 32: Исправление ошибок

Запускаем программу и получаем правильный ответ: (рис. 33).

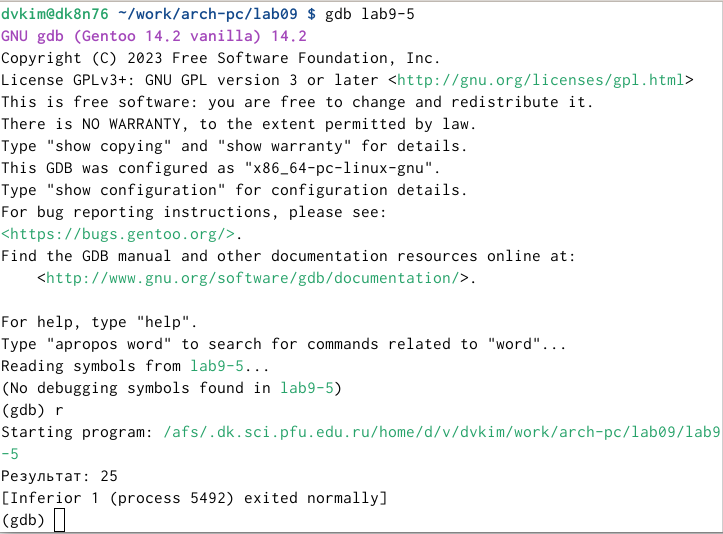


Рис. 33: Запуск программы и получение правильного ответа

# 5 Выводы

В ходе данной работы я приобрёл навыки написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.

# Список литературы

1. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.

2. Robbins A. Bash Pocket Reference. O’Reilly Media, 2016. 156 с.

3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.

4. Newham C. [Learning the bash Shell: Unix Shell Programming](http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658). O’Reilly Media, 2005. 354 с.