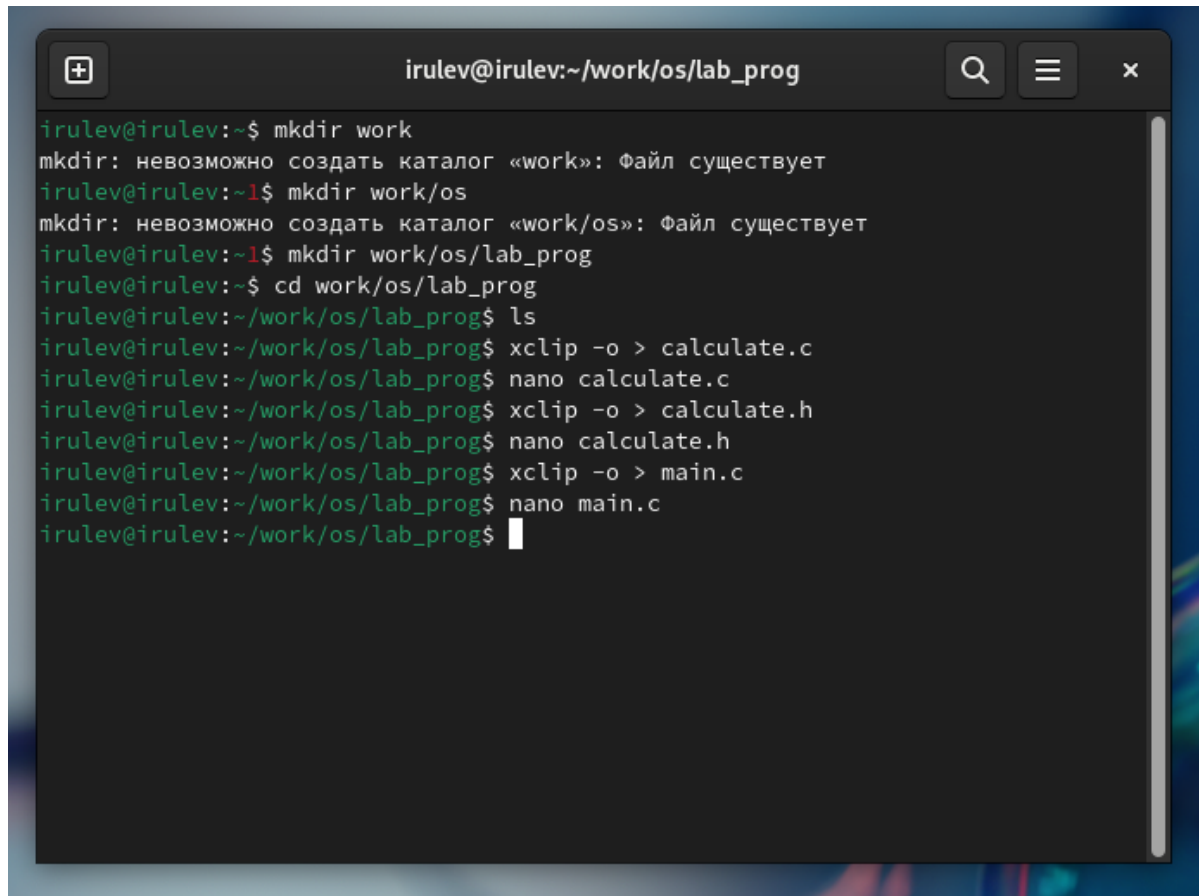


Цель работы

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.

Выполнение лабораторной работы

В домашнем каталоге создадим подкаталог `~/work/os/lab_prog`. Создадим в нём файлы: `calculate.h`, `calculate.c`, `main.c` из лабораторной работы.



```
irulev@irulev:~/work/os/lab_prog
irulev@irulev:~$ mkdir work
mkdir: невозможно создать каталог «work»: Файл существует
irulev@irulev:~$ mkdir work/os
mkdir: невозможно создать каталог «work/os»: Файл существует
irulev@irulev:~$ mkdir work/os/lab_prog
irulev@irulev:~$ cd work/os/lab_prog
irulev@irulev:~/work/os/lab_prog$ ls
irulev@irulev:~/work/os/lab_prog$ xclip -o > calculate.c
irulev@irulev:~/work/os/lab_prog$ nano calculate.c
irulev@irulev:~/work/os/lab_prog$ xclip -o > calculate.h
irulev@irulev:~/work/os/lab_prog$ nano calculate.h
irulev@irulev:~/work/os/lab_prog$ xclip -o > main.c
irulev@irulev:~/work/os/lab_prog$ nano main.c
irulev@irulev:~/work/os/lab_prog$
```

Выполним компиляцию программы посредством gcc:

```
irulev@irulev:~/work/os/lab_prog$ gcc -c calculate.c
irulev@irulev:~/work/os/lab_prog$ gcc -c main.c
irulev@irulev:~/work/os/lab_prog$ gcc calculate.o main.o -o calcul -lm
irulev@irulev:~/work/os/lab_prog$ ls
calcul calculate.c calculate.h calculate.o main.c main.o
irulev@irulev:~/work/os/lab_prog$
```

Создадим необходимый Makefile.

```
GNU nano 7.2 Makefile Изменён
# Makefile

CC = gcc
CFLAGS =
LIBS = -lm

calcul: calculate.o main.o
    gcc calculate.o main.o -o calcul $(LIBS)

calculate.o: calculate.c calculate.h
    gcc -c calculate.c $(CFLAGS)

main.o: main.c calculate.h
    gcc -c main.c $(CFLAGS)

clean:
    -rm calcul *.o *~
# End Makefile

^G Справка  ^O Записать  ^W Поиск    ^K Вырезать  ^T Выполнить ^C Позиция
^X Выход    ^R ЧитФайл  ^\ Замена   ^U Вставить  ^J Выводить  ^/_ К строке
```

Этот Makefile компилирует программу под названием «calcul» из двух исходных файлов: «calculate.c» и «main.c». Он использует компилятор `gcc` и связывается с математической библиотекой. `Makefile` определяет три скрипта:

- `calcul`: связывает `calculate.o` и `main.o` в исполняемый файл.

- `calculate.o` и `main.o`: скомпилирует `calculate.c` и `main.c` в объектные файлы соответственно.
- `clean`: удаляет исполняемый файл, объектные файлы и файлы резервных копий.

После запуска `make`, он проверит зависимости и перенастроит скрипты по мере необходимости.

С помощью `gdb` выполним отладку программы `calcul`.

```
irulev@irulev:~/work/os/lab_prog$ gdb ./calcul
GNU gdb (Fedora Linux) 13.2-8.fc39
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./calcul...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
(No debugging symbols found in ./calcul)
(gdb)
```

Выводы

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

Ответы на контрольные вопросы

1. Чтобы получить информацию о возможностях таких программ, как `gcc`, `make`, `gdb` и других, вы можете использовать опцию `--help` или `-h`, за которой следует имя программы. Например, `gcc --help` или `gdb -h`. Это отобразит использование и параметры для каждой программы.
2. Основными этапами разработки приложений в UNIX являются:
 - **Редактирование:** написание исходного кода с помощью текстового редактора.
 - **Компиляция:** перевод исходного кода в машинный код с использованием компилятора, такого как `gcc`.
 - **Связывание:** объединение объектных файлов в исполняемый файл с помощью компоновщика, такого как `ld`.
 - **Отладка:** выявление и исправление ошибок в программе с помощью отладчика, такого как `gdb`.

- **Тестирование:** проверка правильности работы программы.

3. В контексте языков программирования суффикс — это символ или набор символов, добавляемый в конец имени переменной или функции для обозначения ее типа или назначения. Например, `myVariable_i` может указывать на целочисленную переменную, а `myFunction_f` может указывать на функцию с плавающей запятой.

4. Основная цель компилятора C в UNIX — преобразовать исходный код C в машинный код, который может выполняться компьютером.

5. Утилита `make` используется для автоматизации процесса сборки путем выполнения серии команд, указанных в `Makefile`. Он проверяет зависимости между файлами и перестраивает только то, что необходимо.

6. Базовая структура `Makefile` состоит из:

- **Цели:** файлы, которые необходимо создать, например исполняемые файлы или объектные файлы.
- **Зависимости:** файлы, необходимые для сборки целевых объектов, например исходные файлы или библиотеки.
- **Команды:** действия, которые необходимо предпринять для создания целей, например компиляция или связывание.

7. Главным свойством, общим для всех программ-отладчиков, является возможность **пошагового выполнения кода**, проверяя состояние переменных и регистров на каждом этапе. Чтобы использовать эту функцию, вам необходимо скомпилировать программу с включенными символами отладки (флаг «-g») и запустить ее под отладчиком, например «gdb».

8. Основные команды `gdb`:

- `run`: запускает выполнение программы.
- `break`: устанавливает точку останова в определенном месте.
- `next`: выполняет следующую строку кода.
- `step`: выполняет следующую инструкцию, переходя к функциям.
- `print`: отображает значение переменной.
- `backtrace`: отображает стек вызовов.

9. Схема отладки, которую я использовал в этой лабораторной работе, включала:

- Компиляция программы с включенными символами отладки (`gcc -g`).
- Запуск программы под `gdb` (`gdb ./program`).
- Установка точек останова в определенных местах («`break main`»).
- Прохождение кода, проверка переменных и регистров («`next`», «`step`», «`print`»).
- Анализ стека вызовов («`backtrace`»).

10. Когда компилятор обнаруживает синтаксическую ошибку, он обычно отображает сообщение об ошибке с указанием номера строки и характера ошибки. Компилятор не создаст исполняемый файл, пока не будут исправлены все синтаксические ошибки.

11. Основными инструментами для понимания исходного кода являются:

- **Отладчики:** такие как gdb, которые позволяют пошагово выполнять код и проверять переменные.
- **Инструменты анализа кода:** например, «сплент», который проверяет наличие потенциальных проблем и предупреждает о возможных ошибках.
- **Документация:** комментарии и документация внутри самого кода.

12. Основными задачами, решаемыми "шиной", являются:

- **Обнаружение утечек памяти:** выявление потенциальных утечек памяти.
- **Обнаружение разыменования нулевого указателя:** предупреждение о потенциальных разыменованиях нулевого указателя.
- **Проверка границ массива:** проверка доступа к массиву за пределами границ.
- **Обнаружение несоответствия типов:** выявление несоответствия типов между переменными и параметрами функции.