Оглавление

Автоматизация тестирования

Цель: автоматизировать тестирование программ работающих со стандартным потоком вывода и файлами разных типов

Структура тестирующей системы:

```
.
├── build_coverage.sh
├── build_debug_asan.sh
├── build_debug_msan.sh
├── build_debug.sh
├── build_debug_ubsan.sh
├── build_realese.sh
├── check_scripts.sh
├── clean.sh
├── codecheck.sh
├── collect_coverage.sh
├── func_tests
│   ├── data
│   ├── readme.md
│   └── scripts
│       ├── build_conv.sh
│       ├── comparator.sh
│       ├── conv.c
│       ├── func_tests.sh
│       ├── neg_case.sh
│       └── pos_case.sh
└── testing.sh
```

Основные скрипты:

- **build_release.sh** – скрипт компилирует программу релизной сборки
- **build_debug.sh** – скрипт компилирует программу с отладочной информацией
- **build_debug_asan.sh** – скрипт компилирует программу с адрессными санитайзерами
- **build_debug_msan.sh** – скрипт компилирует программу с санитайзерами памяти
- **build_debug_ubsan.sh** – скрипт компилирует программу с санитайзерами неопределенного поведения
- **build_coverage.sh** - скрипт компилирует программу с сборкой информации о покрытии кода
- **clean.sh** – скрипт очищает тестирующую область
- **collect_coverage.sh** – скрипт запускает все тесты, и проверят процент покрытия кода после тестирования
- **testing.sh** – скрипт проверяет программу на всех санитайзерах и выводит проверяет процент покрытия кода
- **func_tests/readme.md** – описание тестовых данных
- **func_tests/scripts/func_tests.sh** – скрипт проверяет программу на правильность работы на тестовых данных, находящихся в каталоге **func_tests/data**
- **func_tests/scripts/comparator.sh** – скрипт сравнивает предполагаемые выходные данные программы с фактическими выходными данными по заданной маске
- **func_tests/scripts/pos_case.sh** – скрипт проверяет позитивные тесты
- **func_tests/scripts/neg_case.sh** – скрипт проверяет негативные тесты

Вывод: тестирующая система была написана и интегрирована в лабораторные работы

Этапы компиляции

Цель: изучить этапы компиляции программ, написанных на языке **Си**

Этапы компиляции:

1. Препроцессирование

   Удаляет комментарии, выполняет текстовые замены, вставку файлов и условную компиляцию

2. Трансляция на язык ассемблера

   Переводит файл, полученный препроцессором, с языка **Си** на язык **Ассемблера**

3. Ассемблирование

   С языка ассемблера программа переводится в машинный код. На выходе этого транслятора получается двоичный файл (объектный файл).

4. Компоновка

   - объединяет несколько объектных файлов в единый исполняемый файл
   - выполняет связывание переменных и функций, которые требуются очередному объектному файлу, но находятся где-то в другом месте
   - добавляет специальный код, который подготавливает окружение для вызова функции main, а после ее завершения выполняет обратные действия

Вывод: этапы компиляции были изучены

Отладка и хранение данных в памяти

Цель: изучить отладчик **gdb** и с помощью него научиться изучать расположение разных типов данных в памяти

Результаты:

Отладчик **gdb**:

- Чтобы использовать отладчик нужно использовать ключ **-g** при компиляции. Если не добавлять отладочную информацию при компиляции и запустить утилиту **gdb**, то она запустится, но она сообщит об отсутствии отладочных символов
- Чтобы запустить программу под отладчиком нужно использовать команду **run**
- Для досрочного завершение программы нужно использовать команду **kill**
- Чтобы посмотреть в каком месте программы была совершена остановка, можно использовать команду **where**
- Значение переменной можно посмотреть с помощью команды **print <var>**
- Значение переменной можно поменять с помощью команды **set <var>=<val>**
- Программу можно выполнять пошагово с помощью команд **step** (заходит в функции) и **next**
- Чтобы посмотреть последовательность вызванных функций используется команда **backtrace** или сокращенно **bt**
- Точку останова можно установить с помощью команды **break**
- Временные точки останова могут быть установлены с помощью команды **tbreak**

- Чтобы временно включить/выключить точку останова используют команды **enable/disable** соответственно
- Для пропуска некоторого количества срабатываний точки останова используют команду **ignore <count> <num of breakpoint>**
- Условие остановки на точке останова можно задать с помощью команды **break <position> if <condition>**
- Точки останова используются для остановки программы, а точки наблюдения - для отслеживания изменений в значениях переменных. В **gdb** для создания точки наблюдения используется команда **whatch <var>**
- Точку наблюдения удобно использовать в цикле, в котором поитерационно нужно отслеживать значения переменных
- Для просмотра содержимого области памяти в отладчике gdb используется команда **x[/nfu] [address]**

С помощью отладчика было изучено расположение char, int, usignged, long long, многомерных массивов, строк и структур в памяти.

Типы даныых **char, int, usigned, long long** в памяти:

```
(gdb) info locals
char_var = 7 '\a'
int_var = 7
uns_var = 7
ll_var = 7
```

```
(gdb) x /1xb &char_var
0x7fffffffdd5f:     0x07
(gdb) x /4xb &int_var
0x7fffffffdd60:     0x07 0x00 0x00 0x00
(gdb) x /4xb &uns_var
0x7fffffffdd64:     0x07 0x00 0x00 0x00
(gdb) x /8xb &ll_var
0x7fffffffdd68:     0x07 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

Хранение одномерного массива в памяти:

```
int arr[] = {1, 2, 3, 4, 5};
```

```
(gdb) sizeof(arr)
$1 =20
(gdb) x /20xb arr
0x7fffffffdd50:     0x01 0x00 0x00 0x00 0x02 0x00 0x00 0x00
0x7fffffffdd58:     0x03 0x00 0x00 0x00 0x04 0x00 0x00 0x00
0x7fffffffdd60:     0x05 0x00 0x00 0x00
```

Вывод: Изучен отладчик **gdb**, а также изучены расположения известных типов данных в памяти

Постановка замерного эксперимента

Способы замеров времени

Цель: Изучить разные способы замера времени в языке **Си** и найти из них самый точный

Существует 4 известных способа замера времени в языке **Си**:

- gettimeofday
- clock_gettime
- clock
- rdtsc

Вывод: самый точный способ **rdtsc**, т.к. измерения не зависят от вариаций в частоте процессора и выполняется за один такт процессора без прерываний или задержек.

Способы обращения к элементам одномерного массива

Цель: узнать самый быстрый способ работы с одномерным массивом, с помощью сортировки

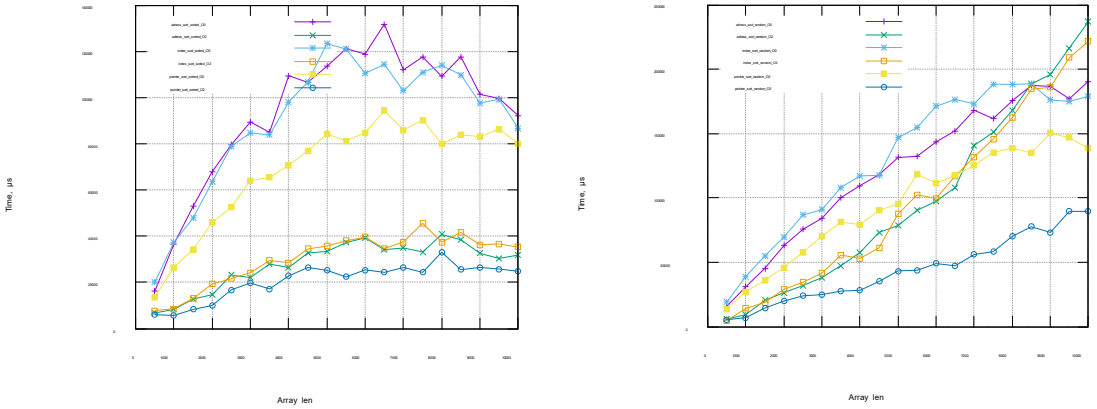
Способ сортировки: сортировка пузырьком с флагом

Количество запусков: 200

Способы работы с одномерным массивом:

- **address_sort** – обращение к элементу массива с использованием адресной арифметики
- **pointer_sort** – обращение к элементу массива по указателю
- **index_sort** – обращение к элементу массив по индексу

Графики сортировки одномерных массивов:



Вывод: графики показывают, что работа указателями является самым быстрым способом. Такой способ обращения к элементам массива является самым быстрым т.к. не тратит времени на операции чтения индексов

Способы обращения к элементам двумерного массива

Цель: узнать самый быстрый способ работы с двумерными массивами

Количество запусков: 2100

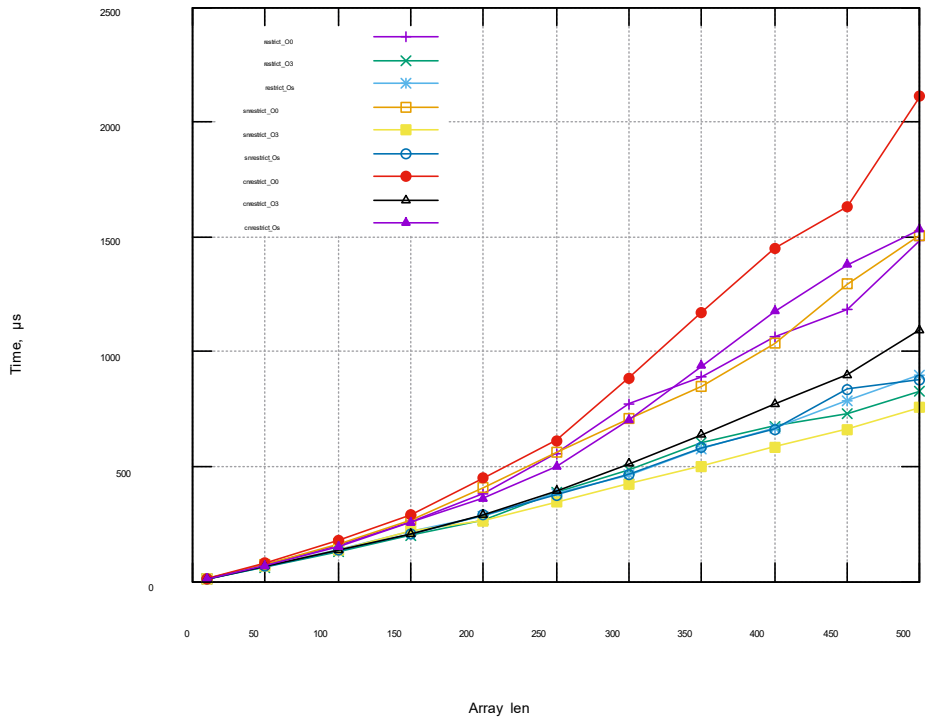
Способы работы с матрицами:

- **snrestrict** – матрица передается в функцию без использования **restrict** и со стандартным обходом элементов
- **cnrestrict** – матрица передается в функцию без использования **restrict** и с обходом элементов по столбцам
- **restrict** – матрица передается в функцию с использованием **restrict** и со стандартным обходом элементов

Таблицы:

	restrict_O3		snrestrict_O3		cnrestrict_O3	
length	t_i, мкс	RSE, %	t_i, мкс	RSE, %	t_i, мкс	RSE, %
10	11,12	2,01	10,95	1,98	10,64	0,68
50	62,91	0,62	68,36	0,65	66,49	0,52
100	129,25	0,44	135,42	0,58	135,79	0,51
150	201,55	1,71	220,21	0,65	206,36	0,39
200	267,52	0,37	265,61	0,69	290,62	0,64
250	387,91	0,58	344,98	0,36	393,05	0,37
300	485,67	0,65	426,57	0,47	511,02	0,67
350	604,65	1,72	503,70	0,29	637,78	0,47
400	677,79	0,98	587,15	0,81	772,13	0,42
450	730,01	0,43	664,20	0,25	900,37	0,41
500	828,27	0,47	756,66	1,16	1096,24	0,42

График:



Вывод: графики показывают, что использование restrict ускоряет работу программы. **restrict** ускоряет код, т.к. он сообщает компилятору, что нет никаких зависимостей между памятью, на которую указывают различные его указатели.

Литература

- Б.У. Керниган, Д.М. Ритчи «Язык программирования С.»
- https://e-learning.bmstu.ru/iu7/course/view.php?id=73
- https://www.opennet.ru/docs/RUS
- https://habr.com/ru/all/
- https://help.ubuntu.ru/wiki/bash
- http://www.gnuplot.info