Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
«Высшая школа программной инженерии»

КУРСОВАЯ РАБОТА

Выбор оптимальной опции оптимизации
по дисциплине: «Введение в профессиональную деятельность»

Выполнил
студент гр.в3530904/00021 С.Н. Юлдашев

Руководитель
Ст. преподаватель А.В. Петров

«15» июня 2021 г.

Санкт-Петербург
2021

Содержание

1 Цель работы

Определить оптимальную опцию оптимизации.

2 Задачи

1. Написать программу на языке C++.
2. Написать скрипт который выполняет оптимизацию написанной программы.
3. Выбрать вариант оптимизации дающий наибольшую производительность.
4. Оформить отчет с использованием LATEX.

3 Введение

Курсовая работа представляет полное описание выполнения задания по написанию сценария по поиску наиболее подходящей оптимизации. Целью выполнения сценария — опции оптимизации, оптимальные для заранее созданного приложения. На этой основе продемонстрированы различные уровни оптимизации, оформленные в виде сценария bash. Bash - усовершенствованная и модернизированная вариация командной оболочки Bourne shell. Bash в основном соответствует стандарту POSIX, но с рядом расширений.[1]Основной критерий упешной оптимизации - время выполнения программы.

[1]Advanced Bash-Scripting Guide - Расширенное руководство по написанию bash-скриптов.

4 Основная часть

Особенности выполнения

- Приложение без оптимизации обрабатывается 25,645 с.
- Вычисление занимаемого исполняемым файлом дискового пространства (в байтах) .
- Сценарий должен принимать только имя исходного файла программы

Вывод сценария должен содержать следующую информацию:

- Текущие опции оптимизации.
- Время затраченное программой на выполнение.
- Занимаемое программой дисковое пространство.

Программный код сценария start.sh

```
#!/bin/bash

filename=$1

for i in "-O0" "-Os" "-O1" "-O2" "-O3" \
"-O2␣-march=native" "-O3␣-march=native" \
"-O2␣-march=native␣-funroll-loops" "-O3␣-march=native␣-funroll-loops" \
"-O3␣-march=native␣-funroll-loops␣-fipa-cp␣-flto" \
"-O3␣-march=native␣-funroll-loops␣-fprofile-generate" \
"-O3␣-march=native␣-funroll-loops␣-fipa-cp␣-flto␣-fprofile-generate"
do
  echo "----------------------"
  echo "␣␣Optimization:␣$i:"
  echo "......................"
  c++ -Wall -Wextra $i $filename -o prg.veg
  echo "␣␣Time:"
  time ./prg.veg 150 20
  echo "......................"
  echo "␣␣Disk␣usage:"
  du -b $filename
  echo "----------------------"
done
```

Для выполнения задания наиболее подходящее решение - это использование цикла, поочередный перебор различных оптимизаций. Для наглядности алгоритма и удобства восприятия представлена блок - схема, описывающая все шаги выполнения задания.

Optimization	Time	Size, B
-O0	0m25,645s	634
-oS	0m4,985s	634
-O1	0m3,705s	634
-O2	0m3,745s	634
-O3	0m3,611s	634
-O2 -march=native	0m3,775s	634
-O3 -march=native	0m3,619s	634
-O2 -march=native -funroll-loops	0m3,722s	634
-O3 -march=native -funroll-loops	0m3,452s	634
-O3 -march=native -funroll-loops -fipa-cp -flto	0m3,548s	634
-O3 -march=native -funroll-loops -fprofile-generate	0m4,714s	634
-O3 -march=native -funroll-loops -fipa-cp -flto -fprofile-generate	0m4,709s	634

Результат программы представлены в таблице 1

5 Заключение

Для отимизации времени выполнения программы были использованы 12 методов и комбинаций методов. Наиболее эффективным оказался комбинированный метод оптимизации с оптимальной опцией, межпроцедурной оптимизацией, оптимизацией времени компоновки и с оптимизацией с обратной связью. Разница во времени выполнения программы с худшей и лучшей оптимизации составляет более 3,5 секунд, очень существенна. Выбор подходящей оптимизации - один из наиболее результативных факторов ее успешности.

6 Список Литературы

1. Уорд Б. "Внутреннее устройство LINUX СПБ.:Питер, 2016. - 384 с.: ил -(Серия "Для профессионалов")

2. Шотс У. "Командная строка LINUX. Полное руководство. СПБ.:Питер, 2017. - 480 с.: ил -(Серия "Для профессионалов")

3. Cooper M. "Advanced Bash-Scripting Guide"Revision 10 Mar 2014 Revised by: 'PUBLICDOMAIN' release-910 c

4. URL : https://ru.wikipedia.org/wiki/Bash - статья в Википедии

7 Приложение 1. Исходный код программы для тестирования.

```
#include <iostream>
#include <algorithm>
#include <stdlib.h>

    const size_t MB = 1024 * 1024;
    size_t MOD = 0;

    unsigned char uniqueNumber() {
        static unsigned char number = 0;
        return ++number % MOD;
    }

    int main(int argc, char** argv) {
        if (argc < 3) {
            return 1;
        }

        size_t BLOCK_SIZE = atoi(argv[1]) * MB;
        MOD = atoi(argv[2]);

        unsigned char* garbage = (unsigned char*)malloc(BLOCK_SIZE);

        std::generate_n(garbage, BLOCK_SIZE, uniqueNumber);
        std::sort(garbage, garbage + BLOCK_SIZE);

        free(garbage);

        return 0;
    }
```

8 Приложение 2. Блок-схема алгоритма.