Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе №4**

**Дисциплина**: Низкоуровневое программирование

**Тема**: Раздельная компиляция

Выполнил студент гр. 3530901/90004 М. Д. Балкин

(подпись)

Преподаватель А. О. Алексюк

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

Оглавление

[1. Техническое задание 3](#_Toc72888781)

[2. Метод Решения 3](#_Toc72888782)

[3. Решение 4](#_Toc72888783)

[4. Сборка программы «по шагам» 4](#_Toc72888784)

[5. Создание статической библиотеки и make-файлов 12](#_Toc72888785)

[6. Вывод 15](#_Toc72888786)

# Техническое задание

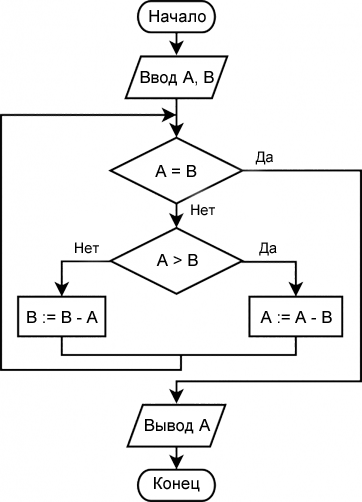
1. На языке С разработать функцию, реализующую нахождение НОД массива. Поместить определение функции в отдельный исходный файл, оформить заголовочный файл. Разработать тестовую программу на языке С.

2. Собрать программу «по шагам». Проанализировать выход препроцессора и компилятора. Проанализировать состав и содержимое секций, таблицы символов, таблицы перемещений и отладочную информацию, содержащуюся в объектных файлах и исполняемом файле.

3. Выделить разработанную функцию в статическую библиотеку. Разработать make-файлы для сборки библиотеки и использующей ее тестовой программы. Проанализировать ход сборки библиотеки и программы, созданные файлы зависимостей.

# Метод Решения

Используя алгоритм Евклида



будем искать НОД массива таким образом:

НОД (НОД (1 элемент массива, 2 элемент массива), 3 элемент массива)) и так далее.

# Решение

Заголовочный файл NOD.h:

#ifndef TEMP\_NOD\_H  
#define TEMP\_NOD\_H  
void NOD(int arr[], int counter);  
#endif //TEMP\_NOD\_H

Основной файл NOD.c:

#include "NOD.h"  
void NOD(int arr[], int size) {  
 int second;  
 int counter = 1;  
 while (counter < size) {  
 second = arr[counter];  
 while (second != \*arr) {  
 if (\*arr > second) {  
 \*arr = \*arr - second;  
 } else {  
 second = second - \*arr;  
 }  
 }  
 counter++;  
 }  
}

Тестовая программа main.c

#include <stdio.h>  
#include "NOD.h"  
  
int main() {  
 int arr[6] = {108, 171, 45, 27, 792, 153}; //{133, 266, 152, 114, 228, 38}; //  
 NOD(arr, sizeof(arr)/sizeof(int));  
 printf("NOD = %d", \*arr);  
 return 0;  
}

# Сборка программы «по шагам»

**Препроцессирование**

Препроцессирование выполняется следующими командами:

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -E main.c -o main.i

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 –E NOD.c -o NOD.i

Результат препроцессирования содержится в файлах main.i и NOD.i. По причине того, что main.c содержит заголовочный файл стандартной библиотеки языка С stdio.h, результат препроцессирования этого файла имеет достаточно много добавочных строк.

Файл main.i

# 1 "main.c"  
# 1 "<built-in>"  
# 1 "<command-line>"  
# 1 "main.c"

…

# 2 "main.c" 2  
# 1 "NOD.h" 1  
  
  
  
# 3 "NOD.h"  
void NOD(int arr[], int counter);  
# 3 "main.c" 2  
  
int main() {  
 int arr[6] = {108, 171, 45, 27, 792, 153};  
 NOD(arr, sizeof(arr)/sizeof(int));  
 printf("NOD = %d", \*arr);  
 return 0;  
}

Файл NOD.i

# 1 "NOD.c"  
# 1 "<built-in>"  
# 1 "<command-line>"  
# 1 "NOD.c"  
# 1 "NOD.h" 1  
  
  
void NOD(int arr[], int counter);  
# 2 "NOD.c" 2  
void NOD(int arr[], int size) {  
 int second;  
 int counter = 1;  
 while (counter < size) {  
 second = arr[counter];  
 while (second != \*arr) {  
 if (\*arr > second) {  
 \*arr = \*arr - second;  
 } else {  
 second = second - \*arr;  
 }  
 }  
 counter++;  
 }  
}

**Компиляция**

Компиляция осуществляется следующими командами:

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 –S main.i -o main.s

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -S NOD.i -o NOD.s

Наибольший интерес представляет файл main.s, так как в нем можно заметить обращение к подпрограмме NOD (значение регистра ra, содержащее адрес возврата из main, сохраняется в стеке).

Файл main.s

.file "main.c"  
 .option nopic  
 .attribute arch, "rv64i2p0\_a2p0\_c2p0"  
 .attribute unaligned\_access, 0  
 .attribute stack\_align, 16  
 .text  
 .section .rodata.str1.8,"aMS",@progbits,1  
 .align 3  
.LC1:  
 .string "NOD = %d"  
 .text  
 .align 1  
 .globl main  
 .type main, @function  
main:  
 addi sp,sp,-48  
 sd ra,40(sp)  
 lui a5,%hi(.LANCHOR0)  
 addi a5,a5,%lo(.LANCHOR0)  
 ld a4,0(a5)  
 sd a4,8(sp)  
 ld a4,8(a5)  
 sd a4,16(sp)  
 ld a5,16(a5)  
 sd a5,24(sp)  
 li a1,6  
 addi a0,sp,8  
 call NOD  
 lw a1,8(sp)  
 lui a0,%hi(.LC1)  
 addi a0,a0,%lo(.LC1)  
 call printf  
 li a0,0  
 ld ra,40(sp)  
 addi sp,sp,48  
 jr ra  
 .size main, .-main  
 .section .rodata  
 .align 3  
 .set .LANCHOR0,. + 0  
.LC0:  
 .word 108  
 .word 171  
 .word 45  
 .word 27  
 .word 792  
 .word 153  
 .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

Файл NOD.s

.file "NOD.c"  
 .option nopic  
 .attribute arch, "rv64i2p0\_a2p0\_c2p0"  
 .attribute unaligned\_access, 0  
 .attribute stack\_align, 16  
 .text  
 .align 1  
 .globl NOD  
 .type NOD, @function  
NOD:  
 li a5,1  
 ble a1,a5,.L1  
 addi a3,a0,4  
 addiw a2,a1,-2  
 slli a5,a2,32  
 srli a2,a5,30  
 addi a5,a0,8  
 add a2,a2,a5  
 j .L7  
.L4:  
 subw a4,a4,a5  
.L5:  
 lw a5,0(a0)  
 beq a4,a5,.L3  
.L6:  
 ble a5,a4,.L4  
 subw a5,a5,a4  
 sw a5,0(a0)  
 j .L5  
.L3:  
 addi a3,a3,4  
 beq a3,a2,.L1  
.L7:  
 lw a4,0(a3)  
 lw a5,0(a0)  
 bne a4,a5,.L6  
 j .L3  
.L1:  
 ret  
 .size NOD, .-NOD  
 .ident "GCC: (SiFive GCC-Metal 10.2.0-2020.12.8) 10.2.0"

**Ассемблирование**

Ассемблирование осуществляется следующими командами:

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c main.s -o main.o

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v -c NOD.s -o NOD.o

Заголовки секций файла main.o

riscv64-unknown-elf-objdump.exe -h main.o

Sections:

Idx Name Size VMA LMA File off Algn

0 .text 0000003e 0000000000000000 0000000000000000 00000040 2\*\*1

CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE

1 .data 00000000 0000000000000000 0000000000000000 0000007e 2\*\*0

CONTENTS, ALLOC, LOAD, DATA

2 .bss 00000000 0000000000000000 0000000000000000 0000007e 2\*\*0

ALLOC

3 .rodata.str1.8 00000009 0000000000000000 0000000000000000 00000080 2\*\*3

CONTENTS, ALLOC, LOAD, READONLY, DATA

4 .rodata 00000018 0000000000000000 0000000000000000 00000090 2\*\*3

CONTENTS, ALLOC, LOAD, READONLY, DATA

5 .comment 00000031 0000000000000000 0000000000000000 000000a8 2\*\*0

CONTENTS, READONLY

6 .riscv.attributes 00000026 0000000000000000 0000000000000000 000000d9 2\*\*0

CONTENTS, READONLY

Таблица символов файла main.o

riscv64-unknown-elf-objdump.exe –t main.o

SYMBOL TABLE:

0000000000000000 l df \*ABS\* 0000000000000000 main.c

0000000000000000 l d .text 0000000000000000 .text

0000000000000000 l d .data 0000000000000000 .data

0000000000000000 l d .bss 0000000000000000 .bss

0000000000000000 l d .rodata.str1.8 0000000000000000 .rodata.str1.8

0000000000000000 l d .rodata 0000000000000000 .rodata

0000000000000000 l .rodata 0000000000000000 .LANCHOR0

0000000000000000 l .rodata.str1.8 0000000000000000 .LC1

0000000000000000 l d .comment 0000000000000000 .comment

0000000000000000 l d .riscv.attributes 0000000000000000 .riscv.attributes

0000000000000000 g F .text 000000000000003e main

0000000000000000 \*UND\* 0000000000000000 NOD

0000000000000000 \*UND\* 0000000000000000 printf

В таблице символов main.o имеется запись: символ “NOD” типа \*UND\*. Эта запись означает, что символ “NOD” использовался в ассемблерном коде, из которого был получен данный объектный файл, но не был определен, ассемблер сделал вывод о том, что символ должен быть определен где-то еще, и отразил это в таблице символов. То же самое относится и к символу “printf”.

Таблица перемещений файла main.o

riscv64-unknown-elf-objdump.exe -d -M no-aliases -r main.o

main.o: file format elf64-littleriscv

Disassembly of section .text:

0000000000000000 <main>:

0: 7179 c.addi16sp sp,-48

2: f406 c.sdsp ra,40(sp)

4: 000007b7 lui a5,0x0

4: R\_RISCV\_HI20 .LANCHOR0

4: R\_RISCV\_RELAX \*ABS\*

8: 00078793 addi a5,a5,0 # 0 <main>

8: R\_RISCV\_LO12\_I .LANCHOR0

8: R\_RISCV\_RELAX \*ABS\*

c: 6398 c.ld a4,0(a5)

e: e43a c.sdsp a4,8(sp)

10: 6798 c.ld a4,8(a5)

12: e83a c.sdsp a4,16(sp)

14: 6b9c c.ld a5,16(a5)

16: ec3e c.sdsp a5,24(sp)

18: 4599 c.li a1,6

1a: 0028 c.addi4spn a0,sp,8

1c: 00000097 auipc ra,0x0

1c: R\_RISCV\_CALL NOD

1c: R\_RISCV\_RELAX \*ABS\*

20: 000080e7 jalr ra,0(ra) # 1c <main+0x1c>

24: 45a2 c.lwsp a1,8(sp)

26: 00000537 lui a0,0x0

26: R\_RISCV\_HI20 .LC1

26: R\_RISCV\_RELAX \*ABS\*

2a: 00050513 addi a0,a0,0 # 0 <main>

2a: R\_RISCV\_LO12\_I .LC1

2a: R\_RISCV\_RELAX \*ABS\*

2e: 00000097 auipc ra,0x0

2e: R\_RISCV\_CALL printf

2e: R\_RISCV\_RELAX \*ABS\*

32: 000080e7 jalr ra,0(ra) # 2e <main+0x2e>

36: 4501 c.li a0,0

38: 70a2 c.ldsp ra,40(sp)

3a: 6145 c.addi16sp sp,48

3c: 8082 c.jr ra

Заголовки секций файла NOD.o

riscv64-unknown-elf-objdump.exe -h NOD.o

NOD.o: file format elf64-littleriscv

Sections:

Idx Name Size VMA LMA File off Algn

0 .text 00000042 0000000000000000 0000000000000000 00000040 2\*\*1

CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE

1 .data 00000000 0000000000000000 0000000000000000 00000082 2\*\*0

CONTENTS, ALLOC, LOAD, DATA

2 .bss 00000000 0000000000000000 0000000000000000 00000082 2\*\*0

ALLOC

3 .comment 00000031 0000000000000000 0000000000000000 00000082 2\*\*0

CONTENTS, READONLY

4 .riscv.attributes 00000026 0000000000000000 0000000000000000 000000b3 2\*\*0

CONTENTS, READONLY

Таблица символов файла NOD.o

riscv64-unknown-elf-objdump.exe –t NOD.o

NOD.o: file format elf64-littleriscv

SYMBOL TABLE:

0000000000000000 l df \*ABS\* 0000000000000000 NOD.c

0000000000000000 l d .text 0000000000000000 .text

0000000000000000 l d .data 0000000000000000 .data

0000000000000000 l d .bss 0000000000000000 .bss

0000000000000040 l .text 0000000000000000 .L1

0000000000000036 l .text 0000000000000000 .L7

0000000000000030 l .text 0000000000000000 .L3

000000000000001e l .text 0000000000000000 .L4

0000000000000020 l .text 0000000000000000 .L5

0000000000000026 l .text 0000000000000000 .L6

0000000000000000 l d .comment 0000000000000000 .comment

0000000000000000 l d .riscv.attributes 0000000000000000 .riscv.attributes

0000000000000000 g F .text 0000000000000042 NOD

**Компоновка**

Компоновка осуществляется следующей командой:

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -v main.o NOD.o

Исполняемый файл a.out(фрагмент)

riscv64-unknown-elf-objdump.exe -j .text -d -M no-aliases a.out >a.ds

a.out: file format elf64-littleriscv

...

0000000000010156 <main>:  
 10156: 7179 c.addi16sp sp,-48  
 10158: f406 c.sdsp ra,40(sp)  
 1015a: 67f5 c.lui a5,0x1d  
 1015c: b2078793 addi a5,a5,-1248 # 1cb20 <\_\_clzdi2+0x46>  
 10160: 6398 c.ld a4,0(a5)  
 10162: e43a c.sdsp a4,8(sp)  
 10164: 6798 c.ld a4,8(a5)  
 10166: e83a c.sdsp a4,16(sp)  
 10168: 6b9c c.ld a5,16(a5)  
 1016a: ec3e c.sdsp a5,24(sp)  
 1016c: 4599 c.li a1,6  
 1016e: 0028 c.addi4spn a0,sp,8  
 10170: 018000ef jal ra,10188 <NOD>  
 10174: 45a2 c.lwsp a1,8(sp)  
 10176: 6575 c.lui a0,0x1d  
 10178: b1050513 addi a0,a0,-1264 # 1cb10 <\_\_clzdi2+0x36>  
 1017c: 1a2000ef jal ra,1031e <printf>  
 10180: 4501 c.li a0,0  
 10182: 70a2 c.ldsp ra,40(sp)  
 10184: 6145 c.addi16sp sp,48  
 10186: 8082 c.jr ra  
  
0000000000010188 <NOD>:  
 10188: 4785 c.li a5,1  
 1018a: 02b7df63 bge a5,a1,101c8 <NOD+0x40>  
 1018e: 00450693 addi a3,a0,4  
 10192: ffe5861b addiw a2,a1,-2  
 10196: 02061793 slli a5,a2,0x20  
 1019a: 01e7d613 srli a2,a5,0x1e  
 1019e: 00850793 addi a5,a0,8  
 101a2: 963e c.add a2,a5  
 101a4: a829 c.j 101be <NOD+0x36>  
 101a6: 9f1d c.subw a4,a5  
 101a8: 411c c.lw a5,0(a0)  
 101aa: 00f70763 beq a4,a5,101b8 <NOD+0x30>  
 101ae: fef75ce3 bge a4,a5,101a6 <NOD+0x1e>  
 101b2: 9f99 c.subw a5,a4  
 101b4: c11c c.sw a5,0(a0)  
 101b6: bfcd c.j 101a8 <NOD+0x20>  
 101b8: 0691 c.addi a3,4  
 101ba: 00c68763 beq a3,a2,101c8 <NOD+0x40>  
 101be: 4298 c.lw a4,0(a3)  
 101c0: 411c c.lw a5,0(a0)  
 101c2: fef716e3 bne a4,a5,101ae <NOD+0x26>  
 101c6: bfcd c.j 101b8 <NOD+0x30>  
 101c8: 8082 c.jr ra

# Создание статической библиотеки и make-файлов

Выделим из программы NOD.c функцию вычитания из большего числа

меньшее в отдельную программу sub.c. Объединим NOD.c и sub.c в

статическую библиотеку NODlib, тестовую программу main.c оставим без

изменений.

Для создания статической библиотеки получим объектные файлы всех

используемых программ: NOD.o и sub.o.

Для создания статической библиотеки получим объектные файлы всех

используемых программ: NOD.o и sub.o.

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -c NOD.c -o NOD.o

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 -c sub.c -o sub.o

Объединим получившиеся файлы в одну библиотеку следующей

командой:

riscv64-unknown-elf-ar.exe -rsc NODlib.a NOD.o sub.o

Используя получившуюся библиотеку, соберем исполняемый файл

программы следующей командой:

riscv64-unknown-elf-gcc.exe -march=rv64iac -mabi=lp64 -O1 --save-temps main.c NODlib.a

Таблица символов исполняемого файла (фрагмент)

riscv64-unknown-elf-objdump.exe -t a.out

a.out: file format elf64-littleriscv

SYMBOL TABLE:

00000000000100b0 l d .text 0000000000000000 .text

…

0000000000000000 l df \*ABS\* 0000000000000000 main.c

0000000000000000 l df \*ABS\* 0000000000000000 NOD.c

0000000000000000 l df \*ABS\* 0000000000000000 sub.c

…

0000000000010156 g F .text 0000000000000032 main

…

0000000000010188 g F .text 0000000000000056 NOD

…

00000000000101de g F .text 0000000000000016 sub

…

000000000001567c g F .text 0000000000000012 \_Bfree

Можно заметить, что в состав программы вошло содержимое объектных

файлов NOD.o и sub.o.

Процесс выполнения команд выше можно заменить make-файлами,

которые произведут создание библиотеки и сборку программы.

Makefile для создания статической библиотеки

# "Фиктивные" цели

.PHONY: all clean

# Исходные файлы, необходимые для сборки библиотеки

OBJS= NOD.c \

sub.c

#Вызываемые приложения

AR = riscv64-unknown-elf-ar.exe

CC = riscv64-unknown-elf-gcc.exe

# Файл библиотеки

MYLIBNAME = NODlib.a

# Параметры компиляции

CFLAGS= -march=rv64iac -mabi=lp64 -O1

# Включаемые файлы следует искать в текущем каталоге

INCLUDES+= -I .

# Make должна искать файлы \*.h и \*.c в текущей директории

vpath %.h .

vpath %.c .

# Построение объектного файла из исходного текста

# $< = %.c

# $@ = %.o

%.o: %.c

$(CC) -MD $(CFLAGS) $(INCLUDES) -c $< -o $@

# Чтобы достичь цели "all", требуется построить библиотеку

all: $(MYLIBNAME)

# $^ = (NOD.o, sub.o)

$(MYLIBNAME): NOD.o sub.o

$(AR) -rsc $@ $^

Makefile для сборки исполняемого файла

# "Фиктивные" цели

.PHONY: all clean

# Файлы для сборки исполнимого файла

OBJS= main.c \

NODlib.a

#Вызываемые приложения

CC = riscv64-unknown-elf-gcc.exe

# Параметры компиляции

CFLAGS= -O1 --save-temps

# Включаемые файлы следует искать в текущем каталоге

INCLUDES+= -I .

# Make должна искать файлы \*.c и \*.a в текущей директории

vpath %.c .

vpath %.a .

# Чтобы достичь цели "all", требуется собрать исполнимый файл

all: a.out

# Сборка исполнимого файла и удаление мусора

a.out: $(OBJS)

$(CC) $(CFLAGS)

Для запуска Makefile воспользуемся программой mingw32-make.exe

mingw32-make.exe -f Makelib

mingw32-make.exe -f Makeapp

Таблица символов исполняемого файла, созданного с помощью Makefile

a.out: file format elf64-littleriscv

SYMBOL TABLE:

00000000000100b0 l d .text 0000000000000000 .text

…

0000000000000000 l df \*ABS\* 0000000000000000 main.c

0000000000000000 l df \*ABS\* 0000000000000000 NOD.c

0000000000000000 l df \*ABS\* 0000000000000000 sub.c

…

0000000000010156 g F .text 0000000000000032 main

…

0000000000010188 g F .text 0000000000000056 NOD

…

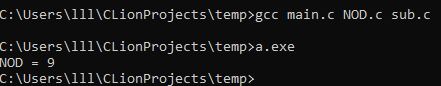
00000000000101de g F .text 0000000000000016 sub

…

000000000001567c g F .text 0000000000000012 \_Bfree

Видим, созданный исполняемый файл аналогичен тому, что был создан через терминал.

Попробуем собрать нашу программу с помощью обычного gcc:



Результат верный!

# Вывод

В ходе лабораторной работы изучена пошаговая компиляция программы

на языке C. Также была создана статическая библиотека и произведена сборка программы с помощью Makefile.