Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра Информатики
Дисциплина «Конструирование программ»

ОТЧЕТ
к лабораторной работе №7
на тему:
«ИНТЕГРАЦИЯ АССЕМБЛЕРНЫХ ПРЕРЫВАНИЙ В ПРОЕКТЫ НА C++»
БГУИР 6-05-0612-02 67

Выполнил студент группы 353503
КОХАН Артём Игоревич

_____
(дата, подпись студента)

Проверил ассистент каф. Информатики
РОМАНЮК Максим Валерьевич

_____
(дата, подпись преподавателя)

Минск 2024

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Задание 1. Вариант 7. Реализовать управление функциями ассемблера: инициализация дерева, добавление и удаление элементов, балансировка дерева, чтобы обеспечить оптимальное время поиска. При каждой операции с деревом (добавление, удаление) автоматически создается резервная копия дерева в файле. Добавить функционал восстановления дерева из последнего бэкапа. Реализацию всего дерева и его функционала выполнить в assembler-коде. Сохранения дерева в файл надо произвести с использованием сериализации. Загрузка дерева из файла надо произвести с использованием десериализации.

2 ВЫПОЛНЕНИЕ РАБОТЫ

Для реализации бинарного дерева поиска в памяти выделим 8 байт под значения узла, 8 байт под адрес левого потомка и 8 байт под адрес правого потомка. Балансировка реализована с помощью отсортированного массива элементов, а далее средний элемент массива выбран в качестве корня, левое поддерево создано из левой части массива, правое поддерево создано из правой части массива. Сложность также вызвала реализация удаления элементов из дерева, так как при удалении элемента необходимо было обновлять указатели на потомков, а также смещать элементы, которые были расположены за удалённым элементом в памяти. При вставке и удалении реализован алгоритм сериализации бинарного дерево поиска в файл, при необходимости можно загрузить дерево из файла путём десериализации дерева из него.

Листинг кода 1 – Функционал программы на языке ассемблера

```
global bst           ; метка bst будет видимой извне

section .data
    bufferInput: times 21 db 0, 0Ah
    bufferInt db 21 dup(0)
    bufferFileNum db 21 dup(0)

    msRoot: db "Enter value root in range from -9 223 372 036 854 775 808 until
9 223 372 036 854 775 807: ", 0 ; 91 символов
    invalid_input_msg: db "Invalid input", 0Ah, 0 ; 14 символов
    info: db "Choise the comand:", 0Ah, "0 - to close program", 0Ah, "1 - insert
node to tree", 0Ah, "2 - delete node from tree", 0Ah, "3 - output tree", 0Ah,
"4 - back up tree", 0Ah, "5 - balance the tree", 0Ah  ; 144 символов
    insert_value: db "Enter to insert value node in range from -9 223 372 036
854 775 808 until 9 223 372 036 854 775 807: ", 0   ; 101 символов
    delete_value: db "Enter to delete value node in range from -9 223 372 036
854 775 808 until 9 223 372 036 854 775 807: ", 0   ; 101 символов
    myTree: db "Root -> Left -> Right:", 0Ah ; 23 символов
```

```
    full_tree_msg: db "TREE IS FULL !!!", 0Ah ; 17 символов
    not_found_msg: db "Element not found :(", 0Ah ; 21 символ
    balance_success_msg: db "Success balance tree", 0Ah ; 21 символов

    file_name: db "tree.txt", 0 ; 8 символов
    file_error_msg: db "Error to open file", 0Ah ; 22 символа
    file_invalid_msg: db "Incorrect data in file", 0Ah ; 23 символа
    file_success_msg: db "Success back up of tree", 0Ah ; 25 символов
    size_file dq 0
    key_file dq 0

    separator: db 0Ah

    node_count dw 0            ; Число узлов дерева
    last_node dq 0             ; Адрес последней node
    parent_in_delete dq 0

    max_nodes_count equ 256

    node_count_balance  dw  0       ;  будем  сохранять  кол-во  элементов  перед
балансировкой
    average_index dq 0

section .bss
    treeNode resb 6144         ; Выделение 6144 байт для 256 узлов
    bufferFile resb 5120

    sortedArray resq 256      ; Массив для хранения элементов (для балансировки)
    arrayIndex resq 0          ; Индекс текущего элемента (для балансировки)

section .text

;;;--------------------------------MAIN PROGRAM---------------------------------
---------------Starting

bst:
    mov rax, 1             ; 1 - номер системного вызова функции write
    mov rdi, 1             ; 1 - дескриптор файла стандартного вызова stdout
    mov rsi, msRoot        ; адрес строки для вывода
    mov rdx, 91            ; количество байтов
    syscall

    mov rax, 0             ; sys_read
    mov rdi, 0             ; stdin
    mov rsi, bufferInput
    mov rdx, 21
    syscall

    mov rsi, bufferInput
    call validate_input
    cmp bl, 2
    je invalid_input_root

    mov [rel treeNode], rax
    inc byte [rel node_count]  ; Увеличиваем счетчик узлов

    lea rax, [rel treeNode] ; mov rax, treeNode
    mov [rel last_node], rax

continue_execution:
    mov ax, [rel node_count]
    cmp ax, 0
```

```
    je root_insert

    mov rax, 1
    mov rdi, 1
    mov rsi, info
    mov rdx, 144
    syscall

    mov rax, 0              ; sys_read
    mov rdi, 0              ; stdin
    mov rsi, bufferInput
    mov rdx, 21
    syscall

    xor ax, ax
    mov al, [rel bufferInput]
    add al, [rel bufferInput + 1] ; num + 0Ah

    cmp al, 58
    jb .invalid_input_main
    cmp al, 63
    ja .invalid_input_main

    cmp al, 59
    je insertCall

    cmp al, 60
    je deleteCall

    cmp al, 61
    je start_pre_order_traversal

    cmp al, 62
    je restore_tree_from_file

    cmp al, 63
    je balance_tree

    mov rax, 60               ; Завершение программы
    syscall

.invalid_input_main:
    mov rax, 1
    mov rdi, 1
    mov rsi, invalid_input_msg
    mov rdx, 14
    syscall

    jmp continue_execution

;;;--------------------------------MAIN PROGRAM---------------------------------
---------------Ending

;;;--------------------------------ROOT INSERT----------------------------------
--------------Strting

root_insert:
    mov rax, 1              ; sys_write
    mov rdi, 1              ; stdout
    mov rsi, msRoot
    mov rdx, 91
    syscall
```

```
    mov rax, 0              ; sys_read
    mov rdi, 0              ; stdin
    mov rsi, bufferInput
    mov rdx, 21
    syscall

    mov rsi, bufferInput
    call validate_input
    cmp bl, 2
    je invalid_input_node

    mov [rel treeNode], rax

    inc byte [rel node_count]  ; Увеличиваем счетчик узлов

    mov rcx, [rel last_node]
    add rcx, 24              ; Узлы занимают 24 байт
    mov [rel last_node], rcx

    jmp continue_execution

;;;--------------------------------ROOT INSERT---------------------------------
--------------Ending

;;;--------------------------------CHECK INPUT---------------------------------
---------------Starting

invalid_input_node:
    mov rax, 1
    mov rdi, 1
    mov rsi, invalid_input_msg
    mov rdx, 14
    syscall

    jmp insertCall

invalid_input_root:
    mov rax, 1
    mov rdi, 1
    mov rsi, invalid_input_msg
    mov rdx, 14
    syscall

    jmp bst

validate_input:
    xor rax, rax
    xor rbx, rbx
    xor rcx, rcx
    xor rdx, rdx
    mov bl, 1                ; Флаг для отрицательного числа

    lodsb                    ; Загрузить первый символ из буфера в al
    cmp al, '-'              ; Проверка на знак минуса
    je check_digits          ; Если минус, перейти к проверке цифр
    cmp al, 0Ah
    je not_number
    mov bl, 0                ; Флаг для положительного числа
```

```
back_to_digits:
    cmp al, 0Ah                ; Проверка на конец строки (новая строка)
    je end_check               ; Конец строки
    cmp al, '0'                ; Проверка, является ли это цифрой
    jb not_number
    cmp al, '9'
    ja not_number

    cmp bl, 1
    je convertNEG_to_number

convert_to_number:
    imul rdx, rdx, 10          ; Умножение текущего числа на 10
    jo overflow
    sub al, '0'                ; Конвертация ASCII в число
    add rdx, rax               ; Добавление к общему
    jo overflow

    jmp check_digits

convertNEG_to_number:
    imul rdx, rdx, 10
    jo overflow
    sub al, '0'
    sub rdx, rax
    jo overflow

    jmp check_digits

check_digits:
    lodsb                      ; Загрузить следующий символ из буфера

    jmp back_to_digits         ; Повторить проверку цифр

not_number:
    mov bl, 2                  ; Индикатор некорректного ввода

    ret

end_check:
    mov rax, rdx               ; Результат в rax

    ret

overflow:
    mov bl, 2                  ; Индикатор переполнения

    ret

;;;--------------------------------CHECK INPUT---------------------------------
---------------Ending

;;;-------------------------------INSERT NODE----------------------------------
---------------Starting

insertCall:
    call start_pre_order_traversal_to_file

    mov ax, [rel node_count]
    cmp ax, max_nodes_count
    je .full_tree
```

```
    mov rax, 1              ; sys_write
    mov rdi, 1              ; stdout
    mov rsi, insert_value
    mov rdx, 101
    syscall

    mov rax, 0             ; sys_read
    mov rdi, 0             ; stdin
    mov rsi, bufferInput
    mov rdx, 21
    syscall

    mov rsi, bufferInput
    call validate_input
    cmp bl, 2
    je invalid_input_node

    call insert

    jmp continue_execution

.full_tree:
    mov rax, 1
    mov rdi, 1
    mov rsi, full_tree_msg
    mov rdx, 17
    syscall

    jmp continue_execution

insert:
    lea rsi, [rel treeNode] ;mov rsi, treeNode , Адрес node(начинаем с корня)
    call insert_recursive

    ret

insert_recursive:
    mov rdx, rsi
    lea rcx, [rel treeNode]
    cmp rsi, rcx
    jne no_root_insert
    mov rdx, [rsi]
    mov rdi, [rsi]

    cmp rax, rdi
    jl .left_branch           ; Идем влево, если значение < текущего узла
    jge .right_branch         ; Идем вправо, если значение > текущего узла

    ret

.left_branch:
    ; Переход к левому потомку
    add rsi, 8                    ; Левый потомок

    mov rdi, [rsi]

    test rdi, rdi
    jz create_node

    call insert_recursive

    ret
```

```
.right_branch:
    ; Переход к правому потомку
    add rsi, 16                    ; Правый потомок

    mov rdi, [rsi]

    test rdi, rdi
    jz create_node

    call insert_recursive

    ret



no_root_insert:
    mov rdi, [rdx] ; значение ноды
    test rdi, rdi
    jz create_node_no_root

    mov rbx, [rdi]
    mov rdi, rbx

    cmp rax, rdi
    jl .left_branch_no_root            ; Идем влево, если значение < текущего
узла
    jge .right_branch_no_root          ; Идем вправо, если значение > текущего
узла

.left_branch_no_root:
    mov rdx, [rsi] ; адрес ноды
    add rdx, 8
    mov rsi, rdx

    call no_root_insert

    ret

.right_branch_no_root:
    mov rdx, [rsi] ; адрес ноды
    add rdx, 16
    mov rsi, rdx

    call no_root_insert

    ret

create_node:
    mov rcx, [rel last_node]
    add rcx, 24              ; Узлы занимают 24 байт
    mov [rel last_node], rcx

    mov [rcx], rax           ; Сохраняем значение
    mov [rsi], rcx

    inc byte [rel node_count]    ; Увеличиваем счетчик узлов

    ret

create_node_no_root:
    mov rcx, [rel last_node]
```

```
    add rcx, 24              ; Узлы занимают 24 байт
    mov [rel last_node], rcx

    mov [rcx], rax           ; Сохраняем значение
    mov [rdx], rcx

    inc byte [rel node_count]    ; Увеличиваем счетчик узлов

    ret

;;;--------------------------------INSERT NODE---------------------------------
---------------Ending

;;;--------------------------------OUTPUT TREE---------------------------------
---------------Starting

start_pre_order_traversal:
    mov rax, 1
    mov rdi, 1
    mov rsi, myTree
    mov rdx, 23
    syscall

    xor rdi, rdi
    lea rdi, [rel treeNode]

    call .pre_order_traversal

    jmp continue_execution

.pre_order_traversal:
    test rdi, rdi
    jz done_pre_order_traversal

    ; Обрабатываем текущий узел
    mov rax, [rdi]
    call print_node_value

    ; Обходим левое поддерево
    add rdi, 8
    mov rsi, [rdi]
    sub rdi, 8

    push rdi

    test rsi, rsi
    jz .skip_left

    mov rdi, rsi
    call .pre_order_traversal

.skip_left:
    pop rdi
    ; Обходим правое поддерево
    mov rsi, [rdi + 16]
    test rsi, rsi
    jz done_pre_order_traversal
    mov rdi, rsi
    call .pre_order_traversal

done_pre_order_traversal:
```

```
    ret

print_node_value:
    mov r8, rdi

    ; Преобразуем число в строку
    mov rsi, bufferInt         ; Указатель на буфер
    call int_to_string

    ; Выводим строку на экран
    mov rdx, rax            ; Длина строки
    mov rax, 1              ; sys_write
    mov rdi, 1              ; stdout
    syscall

    mov rax, 1              ; sys_write
    mov rdi, 1              ; stdout
    mov rsi, separator
    mov rdx, 1
    syscall

    mov rdi, r8

    ret

int_to_string:
    ; Преобразует число в строку
    ; Вход: rax = значение
    ; Выход: rsi = строка, rax = длина строки

    xor rbx, rbx            ; по умолчанию число положительное
    mov rcx, 10             ; Основание системы счисления
    lea rdi, [rsi + 20]     ; Конец буфера
    mov byte [rdi], 0       ; Null-terminator

    test rax, rax
    jns .convert_loop

    neg rax
    mov bl, '-'

.convert_loop:
    dec rdi
    xor rdx, rdx
    div rcx
    add dl, '0'
    mov [rdi], dl
    test rax, rax
    jnz .convert_loop

    ; Если число было отрицательным, добавляем минус
    cmp bl, '-'
    jne .done
    dec rdi
    mov [rdi], bl

.done:
    add rsi, 21
    mov rax, rsi
    sub rsi, 21
    sub rax, rdi            ; Длина строки
    mov rsi, rdi            ; Указатель на начало строки
```

```
    ret

;;;-------------------------------OUTPUT TREE-------------------------------
---------------Ending
```

C++ позволяет вызывать функции assembler-a NASM в программе. Это позволяет оптимизировать код, написанный на C++. Для этого нам необходимо скомпилировать файл кода *.asm в объектный файл: nasm -f elf64 BST.asm -o BST.o. С помощью компилятора gcc линкуем и запускаем программу на языке C++: gcc main.cpp BST.o -o main.

Листинг кода 2 – Функционал программы на языке C++

```
#include <iostream>

extern "C" void bst();

int main()
{
    bst();
    return 0;
}
```

ВЫВОД

В ходе лабораторной работы получено понимание принципов работы для интеграции ассемблерных вставок в проекты, написанные на C++ и принципы работы с прерываниями на операционной системе linux.