****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №2 по курсу**

**«Операционные системы»**

Группа: М80 – 206Б-18

Студент: Касимов М.М.

Преподаватель: Соколов А.А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2019.

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Демонстрация работы программы
6. Вывод

**Постановка задачи**.

Произвести распараллеленный поиск по дереву общего вида. Первоначальные данные задаются в файле в виде простых команд (например, "+ 1 /" добавить элемент к корню дерева, "+ 2 1/3/" добавить элемент 2 по пути 1->3 в дереве).

**Общие сведения о программе**

Программа компилируется с ключом - lpthread. Используются заголовочные файлы stdio.h, unistd.h, stdbool.h, stdlib.h, pthread.h, queue.h (очередь, которую я написал для данной работы). В программе используются следующие системные вызовы:

1. **pthread\_mutex\_lock** – для захвата мьютекса потоком
2. **pthread\_create** – для создания потока
3. **pthread\_join** – для ожидания завершения потока
4. **pthread\_mutex\_unlock** – для освобождения мьютекса потоком
5. **pthread\_exit** – для завершения потока

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Создание общего дерева используя данные из файла.
2. Используя системный вызов создать «дочерний» поток.
3. В функции поиска для каждого узла конкретного уровня дерева создавать потоки для поиска.
4. Узлы, которым не хватило потоков, мы помещаем в очередь и производим поиск в ширину.
5. Когда мы найдем наш элемент в дереве, мы изменим булево поле в структуре на true.

**Основные файлы программы**.

**Файл queue.h**

#ifndef OS\_LAB\_3\_QUEUE\_H

#define OS\_LAB\_3\_QUEUE\_H

#include <stdbool.h>

#include <stdlib.h>

struct node {

int val;

struct node \* son;

struct node \* bro;

};

typedef struct node Tree;

typedef struct QueueItem queue\_item;

struct QueueItem {

struct QueueItem\* next;

struct QueueItem\* prev;

Tree\* value;

};

typedef struct Queue queue;

struct Queue {

queue\_item\* head;

queue\_item\* tail;

size\_t size;

};

**Файл queue.c**

#include "queue.h"

void q\_init(queue\* q) {

q->head = NULL;

q->tail = NULL;

q->size = 0;

}

Tree\* q\_top(queue\* q) {

return q->head->value;

}

Tree\* q\_pop(queue\* q) {

Tree\* temp = q\_top(q);

queue\_item\* ptr\_to\_free = q->head;

q->head = q->head->next;

if (q->head == NULL) {

q->tail = NULL;

}

free(ptr\_to\_free);

q->size--;

return temp;

}

void q\_push(queue\* q, Tree\* elem) {

queue\_item\* new\_elem = malloc(sizeof(queue\_item));

new\_elem->value = elem;

new\_elem->next = 0;

if (q->head == NULL) {

q->head = new\_elem;

q->tail = new\_elem;

new\_elem->prev = 0;

} else {

q->tail->next = new\_elem;

new\_elem->prev = q->tail;

q->tail = new\_elem;

}

q->size++;

}

bool q\_empty(queue\* q) {

return q->head == NULL;

}

size\_t q\_size(queue\* q) {

return q->size;

}

void q\_destroy(queue\* q) {

queue\_item\* start = q->head;

while (start != NULL) {

queue\_item\* next = start->next;

free(start);

start = next;

}

q->head = NULL;

q->tail = NULL;

q->size = 0;

}

**Файл main.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdbool.h>

#include <pthread.h>

#include "queue.h"

typedef struct ThreadParams thread\_params;

void pre\_check(bool\* flag) {

FILE\* file = fopen("in.txt","r");

char temp = fgetc(file);

putchar(temp);

while (temp != EOF) {

if ( temp < '0' && temp != '+' && temp != '/' && temp != ' ' && temp != '\n' && temp != EOF || temp != '\n'&& temp != EOF && temp > '9' && temp != '+' && temp != '/' && temp != ' ' ) {

\*flag = true;

fclose(file);

return;

}

temp = fgetc(file);

if (temp != EOF)

putchar(temp);

}

fclose(file);

}

Tree\* Tree\_create(int v) {

Tree\* tree = (Tree \*) malloc(sizeof(Tree));

if (!tree){

printf("Out of memory\n");

exit(0);

}

tree->val = v;

tree->son = NULL;

tree->bro = NULL;

return tree;

}

Tree\* find\_in\_son (Tree\* root,int v) {

if (root ->son == NULL)

return root;

Tree\* tmp = root->son;

while (tmp->val != v) {

tmp = tmp->bro;

}

return tmp;

}

struct ThreadParams {

Tree\* root;

int num;

bool\* found;

int\* thread\_count;

pthread\_mutex\_t\* count\_mutex;

};

void\* tree\_find(void\* arg) {

thread\_params\* params = (thread\_params\*) arg;

bool\* found = params->found;

int num = params->num;

Tree\* root = params->root;

int\* thread\_count = params->thread\_count;

pthread\_mutex\_t\* count\_mutex = params->count\_mutex;

if (root == NULL){

pthread\_exit(NULL);

}

if (root->val == num) {

\*(params->found) = true;

pthread\_exit(NULL);

}

if (root->son == NULL) {

pthread\_exit(NULL);

}

int son\_count = 0;

Tree\* son = root->son;

while(son != NULL) {

if (son->val == num) {

\*(params->found) = true;

pthread\_exit(NULL);

}

son\_count++;

son = son->bro;

}

son = root->son;

Tree\* sons[son\_count];

int counter = 0;

while(son != NULL) {

sons[counter] = son;

counter++;

son = son->bro;

}

pthread\_mutex\_lock(count\_mutex);

int create\_threads = (\*thread\_count > son\_count) ? son\_count : \*thread\_count;

(\*thread\_count) -= create\_threads;

pthread\_mutex\_unlock(count\_mutex);

pthread\_t threads[create\_threads];

thread\_params\* new\_params = malloc(sizeof(thread\_params) \* create\_threads);

for (int i = 0; i < create\_threads; ++i) {

new\_params[i] = \*params;

new\_params[i].root = sons[i];

pthread\_create(&threads[i], NULL, tree\_find, &new\_params[i]);

}

queue q;

q\_init(&q);

for (int i = create\_threads; i < son\_count; ++i) {

q\_push(&q, sons[i]);

}

while (!q\_empty(&q)) {

Tree\* item = q\_pop(&q);

if (item->val == num) {

\*found = true;

break;

}

Tree\* item\_son = item->son;

while (item\_son != NULL) {

q\_push(&q, item\_son);

item\_son = item\_son->bro;

}

}

for (int i = 0; i < create\_threads; ++i) {

pthread\_join(threads[i], NULL);

}

pthread\_mutex\_lock(count\_mutex);

(\*thread\_count) += create\_threads;

pthread\_mutex\_unlock(count\_mutex);

free(new\_params);

q\_destroy(&q);

pthread\_exit(NULL);

}

void pars(Tree\*\* root,char\* filename) {

FILE\* in = fopen (filename,"r");

char tmp;

int status;

while ((tmp = fgetc(in)) != EOF) {

if (tmp == ' ' || tmp == '\n') {

tmp = fgetc(in);

continue;

}

if (tmp == '+') {

Tree\*\* ptr = root;

int v = 0,trash = 0,n = 0;

while ((tmp = fgetc(in)) != ' ') {

v \*= 10;

v += tmp - '0';

}

//fgetc(in);

while ((tmp = fgetc(in)) != '\n') {

if ( tmp != '/') {

n \*= 10;

n += tmp - '0';

}

else {

while (\*ptr != NULL) {

if ((\*ptr)->val == n)

ptr = &(\*ptr) -> son;

else {

ptr = &(\*ptr)->bro;

if ((\*ptr)->val == n)

ptr = &(\*ptr) ->son;

}

break;

}

n = 0;

}

}

while (\*ptr != NULL)

ptr = &(\*ptr) -> bro;

\*ptr = (Tree\*) malloc(sizeof(Tree));

(\*ptr)->val = v;

(\*ptr)->son = NULL;

(\*ptr)->bro = NULL;

}

}

}

void tabs(int n) {

printf("%d: ",n);

for (int i = 0; i < n; ++i)

printf("\t");

}

void print\_tree(Tree\* root,int lvl) {

if (!root)

return;

Tree\* temp = root;

while (temp != NULL) {

tabs(lvl);

printf("%d\n",temp -> val);

print\_tree(temp -> son,lvl + 1);

temp = temp->bro;

}

}

int main(int argc,char\*\* argv)

{

bool flag = false;

Tree \* root = NULL;

if (flag) {

printf("error\n");

return 0;

}

pars(&root,argv[1]);

print\_tree(root,0);

printf("enter key to find\n");

int num;

scanf("%d",&num);

bool\* found = malloc(sizeof(bool));

\*found = false;

pthread\_mutex\_t\* count\_mutex = malloc(sizeof(pthread\_mutex\_t));

pthread\_mutex\_init(count\_mutex, NULL);

printf("enter max thread count\n");

int\* thread\_count = malloc(sizeof(int));

scanf("%d",thread\_count);

thread\_params params = {

.found = found,

.num = num,

.root = root,

.thread\_count = thread\_count,

.count\_mutex = count\_mutex

};

pthread\_t first\_thread;

pthread\_create(&first\_thread, NULL, tree\_find, &params);

pthread\_join(first\_thread, NULL);

printf("Node is%s found", \*found ? "" : " not");

printf("\n");

return 0;

}

**Демонстрация работы программы.**

magomed@DESKTOP-PG5DLO1:~/os3$ cat in2.txt

+3 /

+4 3/

+5 3/

+11 3/4/

+777 3/4/11/

+42 3/4/11/

magomed@DESKTOP-PG5DLO1:~/os3$ ./a.out in2.txt

0: 3

1: 4

2: 11

3: 777

3: 42

1: 5

enter key to find

11

enter max thread count

3

Node is found

magomed@DESKTOP-PG5DLO1:~/os3$ ./a.out in2.txt

0: 3

1: 4

2: 11

3: 777

3: 42

1: 5

enter key to find

42

enter max thread count

1

Node is found

magomed@DESKTOP-PG5DLO1:~/os3$ ./a.out in2.txt

0: 3

1: 4

2: 11

3: 777

3: 42

1: 5

enter key to find

14

enter max thread count

1000

Node is not found

magomed@DESKTOP-PG5DLO1:~/os3$ ./a.out in2.txt

0: 3

1: 4

2: 11

3: 777

3: 42

1: 5

enter key to find

90

enter max thread count

1

Node is not found

**Вывод.**

Я научился создавать потоки и взаимодействовать с ними. Узнал что такое мютекс и как работают потоки в ОС Linux. Улучшил навыки программирования на языке программирования Си..