Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Ефимов Александр Владимирович

Группа: М80 – 201Б-18

Вариант: 6

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2019

1. **Постановка задачи**

Произвести распараллеленный поиск по ненаправленному графу в ширину. Граф задается набором значений, что хранятся в вершинах, и набором пар связей. Информация по графу хранится в отдельном файле. Необходимо определить есть ли в графе циклы.

1. **Описание программы**

Вместе с программой подаются два параметра – файл, из которого читается граф и максимальное допустимое количество потоков, при этом последний является дополнительным, но не может быть меньше 1 (иначе у нас не будет потоков кроме *main*) и больше размера графа (иначе у нас будет излишние потоки, занимающие стек после выполнения и ожидающие чтения их результата).

После чтения графа, создается очереди потоков и пар. В пары записываются вершины графов, которые далее необходимо сравнить. Если вершина не была прочитана, то она помечается прочитанной. Если же она была прочитана, то возвращается флаг, указывающий, что цикл найден.

Очередь потоков позволяет упорядочить результаты потоков, а также работать с возможно добавленным ограничением потоков.

1. **Набор testcases**

test1.txt

|  |
| --- |
| 7  0 1  1 2  2 3  2 4  3 4  3 5  4 5  5 6 |

test2.txt

|  |
| --- |
| 9  0 1  1 4  1 5  0 2  2 6  2 7  0 3  3 8 |

test3.txt

|  |
| --- |
| 3  0 1  1 2  2 0 |

test4.txt

|  |
| --- |
| 5  0 1  1 2  2 3  3 4  4 2 |

test5.txt

|  |
| --- |
| 6  0 1  1 2  2 3  3 4  4 5  5 0 |

1. **Результаты выполнения тестов.**

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ make

gcc PairQueue.c -c

gcc -lpthread ThreadQueue.c -c

gcc -lpthread threads.c -c

gcc threads.o PairQueue.o ThreadQueue.o -lpthread -g -o GraphScan

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ ./GraphScan test1.txt

0 1 0 0 0 0 0

1 0 1 0 0 0 0

0 1 0 1 1 0 0

0 0 1 0 1 1 0

0 0 1 1 0 1 0

0 0 0 1 1 0 1

0 0 0 0 0 1 0

The graph contains loop

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ ./GraphScan test1.txt 1

0 1 0 0 0 0 0

1 0 1 0 0 0 0

0 1 0 1 1 0 0

0 0 1 0 1 1 0

0 0 1 1 0 1 0

0 0 0 1 1 0 1

0 0 0 0 0 1 0

The graph contains loop

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ ./GraphScan test1.txt 2

0 1 0 0 0 0 0

1 0 1 0 0 0 0

0 1 0 1 1 0 0

0 0 1 0 1 1 0

0 0 1 1 0 1 0

0 0 0 1 1 0 1

0 0 0 0 0 1 0

The graph contains loop

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ ./GraphScan test2.txt

0 1 1 1 0 0 0 0 0

1 0 0 0 1 1 0 0 0

1 0 0 0 0 0 1 1 0

1 0 0 0 0 0 0 0 1

0 1 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0

0 0 0 1 0 0 0 0 0

No loops found

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ ./GraphScan test3.txt

0 1 1

1 0 1

1 1 0

The graph contains loop

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ ./GraphScan test4.txt

0 1 0 0 0

1 0 1 0 0

0 1 0 1 1

0 0 1 0 1

0 0 1 1 0

The graph contains loop

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ ./GraphScan test5.txt

0 1 0 0 0 1

1 0 1 0 0 0

0 1 0 1 0 0

0 0 1 0 1 0

0 0 0 1 0 1

1 0 0 0 1 0

The graph contains loop

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$ make clean

rm \*.o GraphScan

rookstar@Refrigerator:~/Git/OS-Labs/os\_lab\_03/src$

1. **Листинг программы**

* **ThreadsQueue.h**

#ifndef THREAD\_QUEUE\_H

#define THREAD\_QUEUE\_H

#define MAX 100

#include <pthread.h>

typedef struct ThreadQueue {

int size;

int first;

pthread\_t array[MAX];

} ThreadQueue;

void threadsInit(ThreadQueue\*);

int threadsSize(ThreadQueue\*);

int threadsEmpty(ThreadQueue\*);

int threadsFull(ThreadQueue\*);

pthread\_t\* threadsPeek(ThreadQueue\*);

pthread\_t threadsPop(ThreadQueue\*);

pthread\_t\* threadsPush(ThreadQueue\*);

#endif

* **ThreadsQueue.c**

#include "ThreadQueue.h"

void threadsInit(ThreadQueue \*Q)

{

Q->size = Q->first = 0;

}

int threadsSize(ThreadQueue \*Q)

{

return Q->size;

}

int threadsEmpty(ThreadQueue \*Q)

{

return Q->size == 0 ? 1 : 0;

}

int threadsFull(ThreadQueue \*Q)

{

return Q->size == MAX ? 1 : 0;

}

pthread\_t\* threadsPeek(ThreadQueue \*Q)

{

return &Q->array[Q->first];

}

pthread\_t threadsPop(ThreadQueue \*Q)

{

int selected = Q->first;

--Q->size;

Q->first = (Q->first + 1) % MAX;

return Q->array[selected];

}

pthread\_t\* threadsPush(ThreadQueue \*Q)

{

// ++Q->size;

return &Q->array[(Q->first + Q->size++) % MAX];

}

* **PairsQueue.h**

#ifndef PAIR\_QUEUE\_H

#define PAIR\_QUEUE\_H

#define MAX 100

typedef struct adjacent

{

int i, j;

} adjacent;

typedef struct PairQueue {

int size;

int first;

adjacent array[MAX];

} PairQueue;

void pairsInit(PairQueue\*);

int pairsSize(PairQueue\*);

int pairsEmpty(PairQueue\*);

int pairsFull(PairQueue\*);

adjacent\* pairsPeek(PairQueue\*);

adjacent pairsPop(PairQueue\*);

void pairsPush(PairQueue\*, adjacent);

#endif

* **PairsQueue.c**

#include "PairQueue.h"

void pairsInit(PairQueue \*Q)

{

Q->size = Q->first = 0;

}

int pairsSize(PairQueue \*Q)

{

return Q->size;

}

int pairsEmpty(PairQueue \*Q)

{

return Q->size == 0 ? 1 : 0;

}

int pairsFull(PairQueue \*Q)

{

return Q->size == MAX ? 1 : 0;

}

adjacent\* pairsPeek(PairQueue \*Q)

{

return &Q->array[Q->first];

}

adjacent pairsPop(PairQueue \*Q)

{

int selected = Q->first;

--Q->size;

Q->first = (Q->first + 1) % MAX;

return Q->array[selected];

}

void pairsPush(PairQueue \*Q, adjacent add)

{

Q->array[(Q->first + Q->size) % MAX] = add;

++Q->size;

}

* **threads.c**

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

#include <fcntl.h>

#include "PairQueue.h"

#include "ThreadQueue.h"

#include <stdio.h>

typedef enum bool {false, true} bool;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

PairQueue pairs;

typedef struct vertex {

bool adj;

bool visited;

} vertex;

vertex \*\*AdjMatr = NULL;

int MatrSize = 0;

void FillAdjacency (int fd)

{

int ByteRead = 0;

char x = '\0';

if ( (ByteRead = read(fd, &x, 1)) <= 0)

{

write(STDOUT\_FILENO, "Failed to read size\n", 21);

exit(1);

}

do

{

MatrSize = MatrSize \* 10 + (x - '0');

} while ( (ByteRead = read(fd, &x, 1) > 0) && x != '\n');

int i = 0, j = 0;

AdjMatr = malloc(sizeof(vertex \*) \* MatrSize);

for (i = 0; i < MatrSize; ++i)

AdjMatr[i] = malloc(sizeof(vertex) \* MatrSize);

for (i = 0; i < MatrSize; ++i)

for (j = 0; j < MatrSize; ++j)

{

AdjMatr[i][j].adj = false;

AdjMatr[i][j].visited = false;

}

while ( (ByteRead = read(fd, &x, 1)) > 0)

{

i = j = 0;

if (x == '\n' || x == ' ') continue;

do {

i = i \* 10 + (x - '0');

} while ( (ByteRead = read(fd, &x, 1) > 0) && x != ' ');

read(fd, &x, 1); // Skip whitespace

do {

j = j \* 10 + (x - '0');

} while ( (ByteRead = read(fd, &x, 1) > 0) && x != '\n');

AdjMatr[i][j].adj = AdjMatr[j][i].adj = true;

}

}

void \* BreadthFirstSearch(void \*args)

{

adjacent \*pair = (adjacent \*) args;

int iT = pair->j; //as in i Transpose

int jT = pair->i; //as in j Transpose

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

{

if (i == iT) continue;

if (AdjMatr[i][jT].adj)

{

pthread\_mutex\_lock(&mutex);

if (AdjMatr[i][jT].visited)

{

pthread\_mutex\_unlock(&mutex);

pthread\_exit((void \*) true);

}

AdjMatr[i][jT].visited = true;

adjacent pair = {i, jT};

pairsPush(&pairs, pair);

pthread\_mutex\_unlock(&mutex);

}

}

pthread\_exit((void \*) false);

}

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

{

int fd;

if (argc > 1)

fd = open(argv[1], O\_RDONLY);

else

{

write(STDOUT\_FILENO, "Specify file as a first parameter\n", 35);

exit(1);

}

if (fd < 0)

{

write(STDOUT\_FILENO, "File failed to open\n", 21);

exit(-1);

}

FillAdjacency(fd);

close(fd);

int i;

for (i = 0; i < MatrSize; ++i)

{

for (int j = 0; j < MatrSize; ++j)

printf("%d ", AdjMatr[i][j].adj);

printf("\n");

}

pairsInit(&pairs);

adjacent pair;

int MaxThreads = 0;

if (argc > 2)

{

i = 0;

while (argv[2][i] != '\0')

{

MaxThreads = MaxThreads \* 10 + (argv[2][i] - '0');

++i;

}

}

else

MaxThreads = MatrSize;

if (MaxThreads <= 0 || MaxThreads > MatrSize)

{

write(STDOUT\_FILENO, "Invalid thread amount\n", 23);

exit(-1);

}

for (i = 0; i < MatrSize; ++i)

{

if (AdjMatr[i][0].adj)

{

pair.i = i;

pair.j = 0;

pairsPush(&pairs, pair);

}

}

AdjMatr[0][0].visited = true;

ThreadQueue threads;

threadsInit(&threads);

while (!pairsEmpty(&pairs) && threadsSize(&threads) < MaxThreads)

{

pthread\_create(threadsPush(&threads), NULL, BreadthFirstSearch, (void \*) pairsPeek(&pairs));

pairsPop(&pairs);

}

void \*status;

bool looped = false;

while (!threadsEmpty(&threads))

{

pthread\_join(threadsPop(&threads), &status);

if ((bool \*) status) looped = true;

while (!pairsEmpty(&pairs) && threadsSize(&threads) < MaxThreads)

{

pthread\_create(threadsPush(&threads), NULL, BreadthFirstSearch, (void \*) pairsPeek(&pairs));

pairsPop(&pairs);

}

}

for (i = 0; i < MatrSize; ++i)

free(AdjMatr[i]);

free(AdjMatr);

pthread\_mutex\_destroy(&mutex);

if (looped)

write(STDOUT\_FILENO, "The graph contains loop\n", 25);

else if (pairsEmpty(&pairs))

write(STDOUT\_FILENO, "No loops found\n", 16);

else

{

write(STDOUT\_FILENO, "what\n", 6);

return 1;

}

return 0;

}

1. **Вывод**

Потоки позволяют понизить время ожидания программы за счет разделения работы между разными потоками, в то же время используя максимальный потенциал стека, выделенного на программу. Например, если однопоточная программа обходит массив построчно, в цикле вызывая функцию, то многопоточная программа позволит одновременно обрабатывать сразу несколько строк.