**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине «Программирование сетевых приложений»

на тему: **«Организация распределенных вычислений с использованием сокета TCP/IP средствами Linux API»**

Выполнила: студентка гр. ИТИ-41

Жукова А.А.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2021

**Цель работы:** понять принципы разработки сетевых приложений, разработанных средствами *Linux API*.

**Задание:**

Распределенное умножение матриц, распределение по столбцам (циклически).

**Ход выполнения**

В ходе выполнения данной лабораторной работы были разработаны сервер и клиент для операционной системы *Linux*. Серверное приложение создает квадратную матрицу заданной размерности и распределяет нагрузку на клиентов, а клиентские приложения в свою очередь получают от сервера данные и выполняют необходимые вычисления. Для передачи данных по сети используется специальный байтовый буфер.

На рисунке 1 представлена работа сервера.

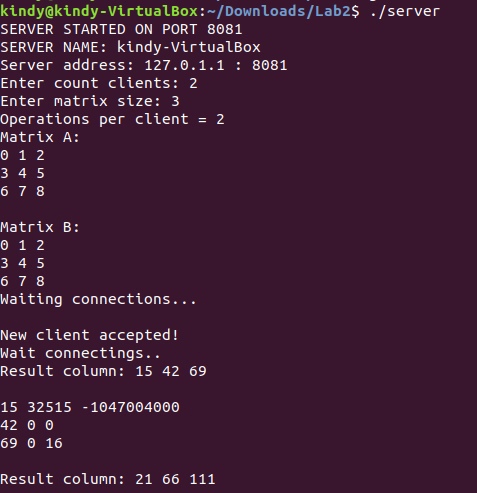


Рисунок 1 – Работа серверного приложения

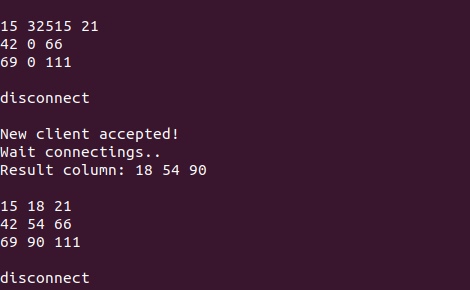


Рисунок 2 – Работа серверного приложения

Клиентское приложение получает от сервера данные в виде матрицы и столбца. После чего клиент производит необходимые умножения с использованием входных данных и в результате возвращает на сервер результат умножения.

На рисунке 3 и 4 представлена работа двух клиентских приложений.

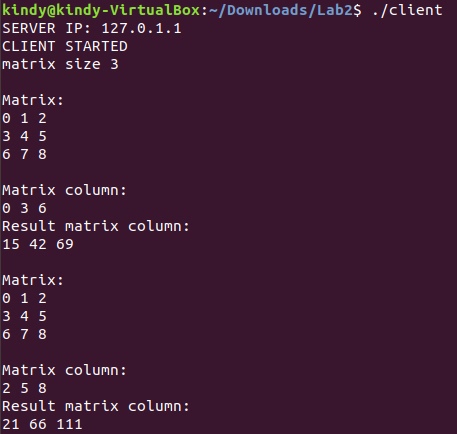


Рисунок 3 – Работа первого клиента

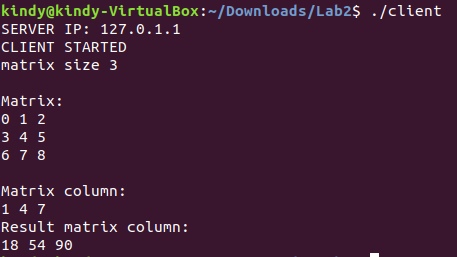


Рисунок 4 – Работа второго клиента

В дальнейшем корректность работы программ была проверена при помощи использования онлайн калькулятора. На рисунке 5 представлен результат проверки программы онлайн калькулятором.

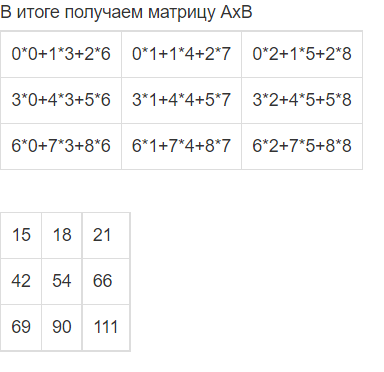


Рисунок 5 – Результат верификации

**Вывод**: в результате выполнения данной лабораторной работы были изучены основные принципы разработки распределенных вычислительных систем, а также полученные знания были применены на практике для создания клиент серверной распределенной среды вычисления на операционной системе *Linux*.

**Листинг server.cpp:**

#pragma warning( disable : 4996)

#include <stdio.h>

#include <string>

#include <sstream>

#include <time.h>

#include <fstream>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <stddef.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <netdb.h>

#include <arpa/inet.h>

#define MY\_PORT 8081

#define PRINTNUSERS if (nclients) printf("%d user onnline\n",nclients);else printf("No User on line\n");

void\* AcceptClient(void \* client\_socket);

void printSquareMatrix(int \* m, int size);

int\* createSquareMatrix(int size);

int\* m\_one;

int\* m\_two;

int\* m\_result;

int count\_calls = 0;

int matrix\_size;

int matrix\_col\_now;

int nclients;

int wasclients;

int max\_nclients;

int count\_operations;

clock\_t start;

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

char buff[1024];

printf("SERVER STARTED ON PORT %d\n", MY\_PORT);

char host[256];

char\* ip;

char\* ip\_i;

struct hostent \* host\_entry;

int hostname = gethostname(host, sizeof(host));

printf("SERVER NAME: %s\n", host);

host\_entry = gethostbyname(host);

int iter = 0;

while ((struct in\_addr\*) host\_entry -> h\_addr\_list[iter]) {

ip = inet\_ntoa(\*((struct in\_addr \* ) host\_entry -> h\_addr\_list[iter++]));

printf("Server address: %s : %d\n", ip, MY\_PORT);

}

int sock;

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("socket");

exit(1);

}

sockaddr\_in local\_addr;

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port = htons(MY\_PORT);

local\_addr.sin\_addr.s\_addr = 0;

if (bind(sock, (sockaddr \* ) & local\_addr, sizeof(local\_addr))) {

close(sock);

perror("bind");

exit(2);

}

if (listen(sock, 0x100)) {

printf("Error listen\n");

close(sock);

return 1;

}

printf("Enter count clients: ");

scanf("%d", & max\_nclients);

printf("Enter matrix size: ");

scanf("%d", & matrix\_size);

if (matrix\_size % max\_nclients > 0) {

count\_operations = matrix\_size / max\_nclients + 1;

} else {

count\_operations = matrix\_size / max\_nclients;

}

printf("Operations per client = %d\n", count\_operations);

m\_one = createSquareMatrix(matrix\_size);

m\_two = createSquareMatrix(matrix\_size);

m\_result = new int[matrix\_size];

printf("Matrix A:\n");

printSquareMatrix(m\_one, matrix\_size);

printf("\nMatrix B:\n");

printSquareMatrix(m\_two, matrix\_size);

printf("Waiting connections...\n\n");

int client\_socket;

sockaddr\_in client\_addr;

socklen\_t client\_addr\_size = sizeof(client\_addr);

while ((client\_socket = accept(sock, (sockaddr\*) & client\_addr, & client\_addr\_size))) {

if (wasclients >= max\_nclients) {

break;

}

nclients++;

wasclients++;

printf("New client accepted!\n");

pthread\_t threadId;

pthread\_create( & threadId, NULL, & AcceptClient, & client\_socket);

printf("Wait connectings..\n");

}

return 0;

}

void\* AcceptClient(void\* client\_socket) {

int sock;

sock = ((int\* ) client\_socket)[0];

char\* buff = new char[1024];

buff[0] = (char) matrix\_size;

buff[1] = (char) count\_operations;

send(sock, buff, sizeof(int) \* 2, 0);

count\_calls++;

for (int ii = 0; ii < count\_operations; ii++) {

if(count\_calls == max\_nclients) {

matrix\_col\_now -= max\_nclients + 1;

count\_operations--;

}

if (matrix\_col\_now >= matrix\_size) {

break;

}

int current\_col = matrix\_col\_now;

if(count\_calls == max\_nclients) { matrix\_col\_now += max\_nclients + 1; }

else { matrix\_col\_now += max\_nclients; }

int i, j;

for (i = 0; i < matrix\_size \* matrix\_size; i++) {

buff[i] = static\_cast<char>(m\_two[i]);

}

for (i, j = current\_col; i < (matrix\_size \* matrix\_size) + matrix\_size; i++, j+=matrix\_size) {

buff[i] = static\_cast<char>(m\_one[j]);

}

send(sock, buff, sizeof(int) \* (matrix\_size \* matrix\_size + matrix\_size), 0);

recv(sock, buff, sizeof(int) \* matrix\_size, 0);

printf("Result column: ");

for (i = 0, j = current\_col; i < matrix\_size; i++, j+=matrix\_size) {

m\_result[j] = static\_cast<int>(buff[i]);

printf("%d ", m\_result[j]);

}

printf("\n\n");

printSquareMatrix(m\_result, matrix\_size);

printf("\n");

}

nclients--;

printf("disconnect\n\n");

close(sock);

return 0;

}

int\* createSquareMatrix(int size) {

int\* arr = new int[size \* size];

for (int i = 0; i < size \* size; i++) {

arr[i] = i;

}

return arr;

}

void printSquareMatrix(int \* m, int size) {

for (int i = 0; i < size \* size; i++) {

printf("%d ", m[i]);

if ((i + 1) % size == 0) {

printf("\n");

}

}

}

**Листинг client.cpp:**

#pragma warning(disable: 4996)

#include <stdio.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <omp.h>

#include <iostream>

#include <string>

#include <sstream>

#include <fstream>

#include <sys/types.h>

#include <stddef.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 8081

char SERVERADDR[15] = "127.0.1.1";

void printSquareMatrix(int\* m, int size);

void printSquareMatrixCol(int\* m, int size);

int\* calculateMatrixCol(int\* m, int\* m\_row, int size);

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

printf("SERVER IP: %s\n", SERVERADDR);

char buff[1024];

printf("CLIENT STARTED\n");

int sock;

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0) {

printf("Socket error\n");

return 1;

}

struct sockaddr\_in dest\_addr;

dest\_addr.sin\_family = AF\_INET;

dest\_addr.sin\_port = htons(PORT);

dest\_addr.sin\_addr.s\_addr = inet\_addr(SERVERADDR);

int iResult = connect(sock, (struct sockaddr\*) & dest\_addr, sizeof(dest\_addr));

if (iResult < 0) {

close(sock);

printf("Unable to connect to the server\n");

return 1;

}

int nsize;

int matrixSize;

int count\_operations;

nsize = recv(sock, buff, sizeof(int) \* 2, 0);

matrixSize = (int)buff[0];

count\_operations = (int)buff[1];

printf("matrix size %d \n", matrixSize);

int\* matrixA = new int[matrixSize \* matrixSize];

int\* matrixBCol = new int[matrixSize];

int current\_op\_count = 0;

while (current\_op\_count < count\_operations &&

(nsize = recv(sock, buff, sizeof(int) \* (matrixSize \* matrixSize + matrixSize), 0)) > 0) {

current\_op\_count++;

int i, j;

for (i = 0, j = 0; i < (matrixSize \* matrixSize) + matrixSize; i++) {

if (i < matrixSize \* matrixSize) {

matrixA[i] = static\_cast<int>(buff[i]);

} else {

matrixBCol[j] = static\_cast<int>(buff[i]);

j++;

}

}

printf("\nMatrix: \n");

printSquareMatrix(matrixA, matrixSize);

printf("\nMatrix column: \n");

printSquareMatrixCol(matrixBCol, matrixSize);

int\* result = calculateMatrixCol(matrixA, matrixBCol, matrixSize);

printf("\nResult matrix column: \n");

printSquareMatrixCol(result, matrixSize);

printf("\n");

for (int i = 0; i < matrixSize; i++) {

buff[i] = static\_cast<char>(result[i]);

}

send(sock, buff, sizeof(int) \* matrixSize, 0);

}

close(sock);

delete[] matrixA;

delete[] matrixBCol;

return 0;

}

void printSquareMatrix(int \* m, int size) {

for (int i = 0; i < size \* size; i++) {

printf("%d ", \*(m + i));

if ((i + 1) % size == 0) {

printf("\n");

}

}

}

void printSquareMatrixCol(int \* m, int size) {

for (int i = 0; i < size; i++) {

printf("%d ", \*(m + i));

}

}

int\* calculateMatrixCol(int\* matrix, int\* column, int size) {

int\* res = new int[size];

for (int i = 0; i < size; i++) {

res[i] = 0;

}

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

res[i] += matrix[i \* size + j] \* column[j];

}

}

return res;

}