Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Теория распределенных систем и параллельных вычислений”

Лабораторная работа №4

“Исследование взаимодействий распределенных

процессов типа «Клиент-Сервер»”

Выполнил:

ст.гр. ИС/б-17-2

Черняев Н.Г.

Проверил:

Кротов К.В.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Исследовать механизм взаимодействия распределено выполняющихся параллельных процессов типа «клиент-сервер».

2 ПОСТАНОВКА ЗАДАЧИ

**Вариант №1.** В состав вычислительного кластера входит три хоста, один из которых реализует функции сервера, два остальных – клиентов.

Сервер разграничивает доступ к трем общим ресурсам – нерешенным, хранящим общую вырученную сумму от продажи товаров, общее количество товаров и остальных товаров. Доступ к ресурсам осуществляется в произвольном порядке, все ресурсы разделяются между клиентами по отдельности. Реализована процедура, выделяющая ресурсы в использование клиентам. Реализовать серверный процесс, который разграничивает доступ клиентов к этой процедуре (процедурам) и к ресурсам.

3 КОД ПРОГРАММЫ

#include <iostream>

#include <time.h>

#include <Windows.h>

#include "mpi.h"

using namespace std;

// Тэг для обмена между клиентами и сервером

#define CLIENT\_SERVER\_MSG\_TAG 99

// Общее кол-во заявок на ресурс

#define MAX\_MESSAGES 10

const int countResources = 3;

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

{

int rank, processes;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &processes);

MPI\_Status status;

int serverProcess = 0;

int resourceProcess = 3;

// Клиент, запросивший ресурс, указывает номер необходимого ресурса в эту переменную

int message[1];

if (rank == serverProcess)

{

cout << "Server started" << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

// Результат ответа от ресурсного процесса, т.е. запрошенный ресурс для клиента

int result[1];

MPI\_Status status;

for (size\_t i = 0; i < MAX\_MESSAGES; i++)

{

// Ожидаем поступления сообщения от клиентов

// т.к. перед приемом сообщения нам необходимо знать номер процесса, который к нам обращается,

// то мы с помощью фукции зондирования входящих сообщений Probe узнаем номер источника сообщения

MPI\_Probe(MPI\_ANY\_SOURCE, CLIENT\_SERVER\_MSG\_TAG, MPI\_COMM\_WORLD, &status);

// Узнаем номер входящего процесса из полученной с помощью зондирования информации

int client = status.MPI\_SOURCE;

// Зная номер процесса, отправившего нам сообщение, можем принять его

MPI\_Recv(&message, 1, MPI\_INT, client, CLIENT\_SERVER\_MSG\_TAG, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

// Полученное сообщение мы помещаем в переменную \_message,

// в которую клиентский процесс, запросивший ресурс, указывает номер необходимого ресурса

// Выводим сообщение о том, что клиентский процесс с номером "client"

// запрашивает ресурс с номером, указанным в теле сообщения

cout << "\nServer : processing the client " << client << "th accessing the resource " << message[0] + 1 << endl;

cout << "-------------------------------------------------------------------------" << endl;

// Отправляем сообщение ресурсу и в теле сообщения указываем номер ресурса, который нам неообходим

MPI\_Send(message, 1, MPI\_INT, resourceProcess, 88, MPI\_COMM\_WORLD);

// Ожидаем ответа от ресурсного процесса

MPI\_Recv(result, 1, MPI\_INT, resourceProcess, 88, MPI\_COMM\_WORLD, &status);

cout << "\nServer : receive resourse " << result[0] << " from resourse process" << endl;

// Отправка запрошенного ресура \_result клиенту "client"

MPI\_Send(result, 1, MPI\_INT, client, CLIENT\_SERVER\_MSG\_TAG, MPI\_COMM\_WORLD);

// Получение подтверждения от клиента об освобождении ресурса

MPI\_Recv(&message, 1, MPI\_INT, client, CLIENT\_SERVER\_MSG\_TAG, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

// Признаком освобождения ресурса является сообщение,

// в котором в качестве тела запроса указывается -99 (-CLIENT\_SERVER\_MSG\_TAG)

if (message[0] == -CLIENT\_SERVER\_MSG\_TAG)

{

cout << "\nServer : " << client << "th client free resource" << endl;

}

}

}

else if (rank == resourceProcess)

{

/\*

resource[0] - общая сумма от продажи товаров

resource[1] - общее кол-во товаров

resource[2] - общее кол-во остальных товаров

\*/

int resource[countResources];

resource[0] = 524334;

resource[1] = 3233;

resource[2] = 232;

// Номер необходимого ресурса, который будет указан в сообщении от сервера

int numResource[1];

// Необходимый ресурс и отправим его в теле сообщения к серверу

int result[1];

for (int i = 0; i < MAX\_MESSAGES; ++i)

{

// Ждем поступления сообщения от сервера

// С помощью фукции зондирования входящих сообщений Probe узнаем, поступило нам сообщение или нет

// Выход из функции Probe произойдет, как только поступит сообщение

MPI\_Status status;

MPI\_Probe(MPI\_ANY\_SOURCE, 88, MPI\_COMM\_WORLD, &status);

// Принимаем сообщение от сервера с номером необходимого ресурса \_num\_resource

MPI\_Recv(numResource, 1, MPI\_INT, serverProcess, 88, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

cout << "\nResource : receive request from server for giving resource number " << numResource[0] + 1 << endl;

// В тело ответного сообщения поместим необходимый ресурс и отправим его в теле сообщения к серверу

result[0] = resource[numResource[0]];

cout << "\nResource : send to server resource number " << numResource[0] + 1 << " = " << result[0] << endl;

// Отправка ответа серверу с необходимым ресурсом

MPI\_Send(result, 1, MPI\_INT, serverProcess, 88, MPI\_COMM\_WORLD);

}

}

else if (rank == 1 || rank == 2)

{

// Генератор случайных чисел для выбора случайного номера ресурса от 1 до countResources

srand(time(0));

int rez[1];

// Определяем кол-во запросов, которое клиент отправит серверу

// для простоты, MAX\_MESSAGES = 10. processes = общее кол-во процессов.

// и того каждый из клиентов отправит по 5 сообщений на запрос ресурса

int countMessageForClient = (MAX\_MESSAGES) / (processes - 2);

// Цикл на отправку сообщений клиентов к серверу

for (size\_t i = 0; i < countMessageForClient; i++)

{

// Случайно выбираем номер ресурса

int num\_resource = rand() % countResources;

// Сообщение клиента о том, что он будет отправлять серверу запрос на ресурс с номером num\_resource

cout << "Client " << rank << ": requests resource number " << num\_resource + 1 << endl;

// Помещаем в тело сообщения номер ресурса, который нам необходим

message[0] = num\_resource;

// Отправка сообщения с запросом серверу

MPI\_Send(message, 1, MPI\_INT, serverProcess, CLIENT\_SERVER\_MSG\_TAG, MPI\_COMM\_WORLD);

// Получение ответа от сервера с запрошенным ресурсом

MPI\_Recv(rez, 1, MPI\_INT, serverProcess, CLIENT\_SERVER\_MSG\_TAG, MPI\_COMM\_WORLD, &status);

// Сообщение от клиента о получении ресурса rez, полученного от сервера

cout << "Client " << rank << ": receive resource " << rez[0] << endl;

// Задержка обработку ресура

Sleep((DWORD)1000);

// Помещаем в тело сообщения тег -99 (-CLIENT\_SERVER\_MSG\_TAG),

// который будет обозначать, что мы освободили этот ресурс

// и он может быть выдан следующему клиенту

message[0] = -CLIENT\_SERVER\_MSG\_TAG;

// Отправка сообщения серверу об освобождении ресурса

MPI\_Send(message, 1, MPI\_INT, serverProcess, CLIENT\_SERVER\_MSG\_TAG, MPI\_COMM\_WORLD);

}

}

MPI\_Finalize();

return 0;

}

4 РЕЗУЛЬТАТЫ

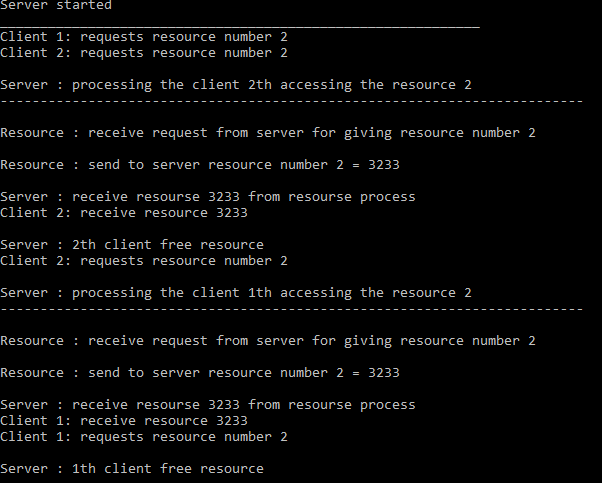


Рисунок 1 – Старт сервера, выполнение запросов 1-2, 2-2

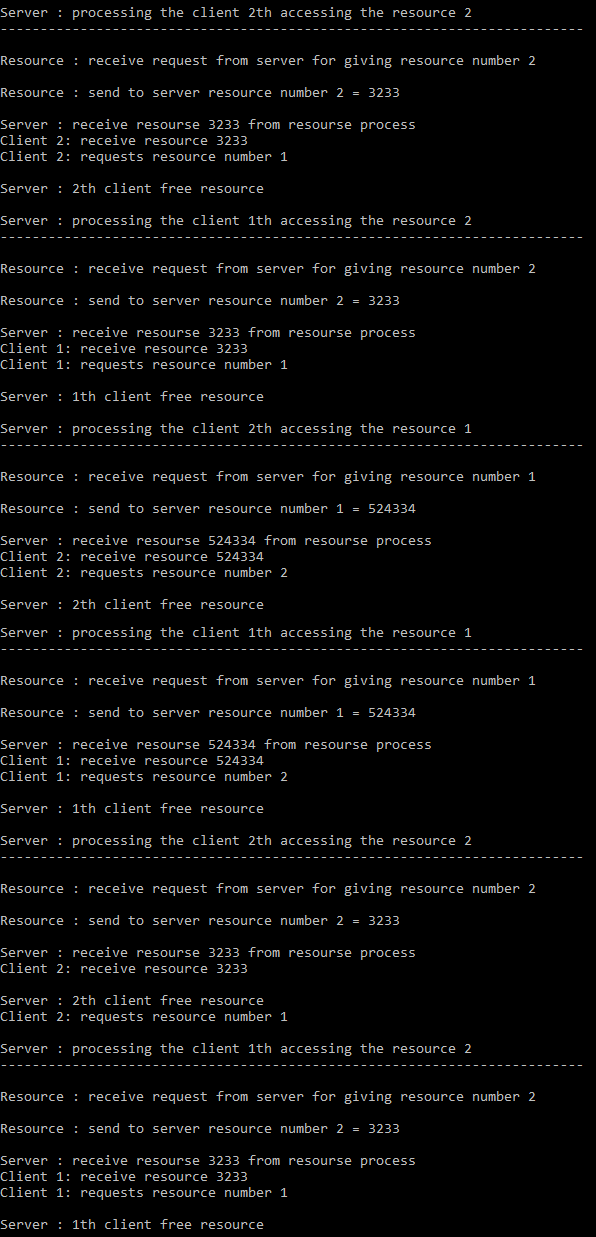


Рисунок 2 – Выполнение запросов 2-2, 1-2, 2-1, 1-1, 2-2, 1-2

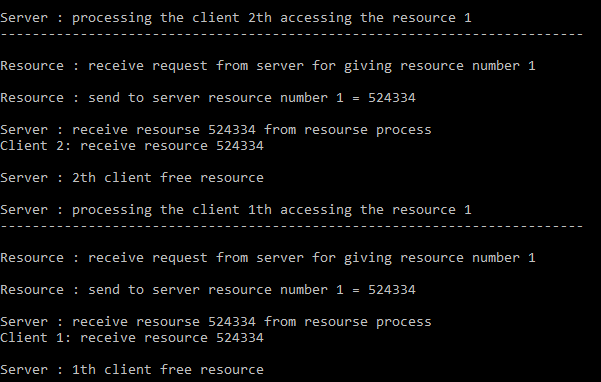


Рисунок 3 – Выполнение запросов 2-1, 1-1

ВЫВОДЫ

В ходе выполнения данной лабораторной работы был исследован механизм взаимодействия распределено выполняющихся параллельных процессов типа «клиент-сервер».

Функционирование сервера было реализовано с помощью концепции рандеву. Реализация рандеву предполагает инициализацию выполнения некоторой процедуры (процедурного процесса), доступ к которой регулируется сервером, в ответ на запрос клиента.

Как указывалось, в методических указаниях к данной лабораторной работе, существует два способа для решения задачи синхронизации (диспетчеризации) при организации сервера. В лабораторной работе был использован второй способ. Он заключается в том, что сервер является посредником между клиентом и ресурсом (процессом, являющимся ресурсом); клиенты вызывают сервер, указывая в нем требуемый вид обработки ресурса, далее сервер сам обращается к ресурсу, разграничивая доступ клиентов;