МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Севастопольский государственный университет»

кафедра Информационные системы

Бариев Эмин Юсуфович

Институт информационных технологий и управления в технических системах

курс 4 группа ИС/б-16-2

09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №5

По дисциплине: «ТРСиПВ»

По теме: «Исследование моделей взаимодействия распределенно выполняющихся процессов»

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(дата)

Руководитель практикума

ст.преподаватель Дрозин А. Ю.

(должность) (подпись) (инициалы,фамилия)

Севастополь 2019

1 ЦЕЛЬ РАБОТЫ

Исследовать алгоритмическое построение методов взаимодействия распределено выполняющихся процессов.

2 ПОСТАНОВКА ЗАДАЧИ

Вариант – 1

Осуществить построение топологии кластера требуемого вида (рис. 2.1); выполнить широковещательную рассылку вводимого с клавиатуры сообщения от узла S на все остальные узлы. На узле, инициирующем рассылку, выводить (в виде матрицы) топологию и остовное дерево, на остальных хостах кластра после получения сообщения выводить номер хоста и сам текст сообщения.



Рисунок 2.1 – Схема каналов взаимодействия процессов в кластере

3 ХОД РАБОТЫ

3.1 Код программы на языке С++

//

// main.cpp

// TRSPV\_5

//

// Created by Emin Bari on 16.11.2019.

// Copyright © 2019 Emin Bari. All rights reserved.

//

#include <iostream>

#include <mpi.h>

**using** **namespace** std;

**static** **int** processRank;

**static** **int** processCount;

**static** **const** **int** ROOT = 0;

**enum**{ECHO, ZOND};

MPI\_Comm comm;

MPI\_Comm graph;

MPI\_Group old\_group;

MPI\_Group new\_group;

MPI\_Status status;

**static** **int** nnodes;

//static int indexes[5] = {2, 5, 8, 10, 12};

//static int edges[12] = {1, 2, 0, 3, 4, 0, 3, 4, 1, 2, 1, 2};

**static** **int** clasterMatrix[5][5] = {

{0, 1, 1, 0, 0},

{1, 0, 0, 1, 1},

{1, 0, 0, 1, 1},

{0, 1, 1, 0, 0},

{0, 1, 1, 0, 0},

};

**static** **int** \*claster\_row = **new** **int**[nnodes];

**static** **int** \*topo\_row = **new** **int**[nnodes];

**static** **int** \*spanning\_row = **new** **int**[nnodes];

**static** **int** topologyMatrix[5][5];

**static** **int** tmp\_topology[5][5];

**static** **int** spanningTree[5][5];

**void** createSpanningTree();

**void** showTopology\_SpanningTree();

**void** rootProcess();

**void** nonRootProcess();

**void** sendMessage();

**void** resendMessage();

**char**\* message;

**int** lenMessage = 50;

**void** root\_message(){

message = "May the forth be with you";

**int** sendCount = 0;

**for**(**int** j = processRank; j < processCount; j++){

**if** (spanning\_row[j] == 1) {

MPI\_Send(message, lenMessage, MPI\_CHAR, j, ZOND, comm);

sendCount++;

}

}

**while** (sendCount--) {

MPI\_Recv(**NULL**, 0, MPI\_INT, MPI\_ANY\_SOURCE, ECHO, comm, &status);

}

}

**void** notRoot\_message(){

message = **new** **char**[lenMessage];

MPI\_Recv(message, lenMessage, MPI\_CHAR, MPI\_ANY\_SOURCE, ZOND, MPI\_COMM\_WORLD, &status);

cout << "Процесс " << status.MPI\_SOURCE << " послал процессу " << processRank << ": '" << message << "'" << endl;

**int** sendCount = 0;

**for**(**int** j = processRank; j < processCount; j++){

**if** (spanning\_row[j] == 1) {

MPI\_Send(message, lenMessage, MPI\_CHAR, j, ZOND, MPI\_COMM\_WORLD);

sendCount++;

}

}

**while** (sendCount--) {

MPI\_Recv(**NULL**, 0, MPI\_INT, MPI\_ANY\_SOURCE, ECHO, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

MPI\_Send(**NULL**, 0, MPI\_INT, status.MPI\_SOURCE, ECHO, MPI\_COMM\_WORLD);

}

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

MPI\_Init(&argc, &argv);

comm = MPI\_COMM\_WORLD;

MPI\_Comm\_size(comm, &processCount);

MPI\_Comm\_rank(comm, &processRank);

//cout<<processRank<<endl;

nnodes = processCount;

//MPI\_Graph\_create(comm, nnodes, indexes, edges, false, &graph );

// if (processRank == 0) {

// int nneighborsCount[1];

// int neighbors[nneighborsCount[0]];

// MPI\_Graph\_neighbors\_count(graph, processRank, nneighborsCount);

// MPI\_Graph\_neighbors(graph, processRank, nneighborsCount[0], neighbors);

//

// for (int i = 0; i < nneighborsCount[0]; i++) {

// cout<<"Neighbor"<<neighbors[i]<<endl;

// }

//

// }

MPI\_Scatter(clasterMatrix, processCount, MPI\_INT, claster\_row, processCount, MPI\_INT, 0, comm);

**if** (processRank == ROOT){

rootProcess();

}

**else**{

nonRootProcess();

}

MPI\_Barrier(comm);

MPI\_Scatter(spanningTree, processCount, MPI\_INT, spanning\_row, processCount, MPI\_INT, 0, comm);

**if**(processRank == ROOT){

root\_message();

}

**else**{

notRoot\_message();

}

MPI\_Finalize();

**return** 0;

}

**void** createSpanningTree(){

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

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

**if** (topologyMatrix[j][i] == 1){

spanningTree[j][i] = 1;

spanningTree[i][j] = 1;

**break**;

}

}

}

}

**void** showTopology\_SpanningTree(){

cout << endl << "Topology matrix: " << endl;

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

**for** (**int** j = 0; j < processCount; j++){

cout<<topologyMatrix[i][j]<<" ";

}

cout<<endl;

}

createSpanningTree();

cout << endl << "Spanning tree: " << endl;

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

**for**(**int** j = 0; j < processCount; j++){

cout<<spanningTree[i][j]<<" ";

}

cout<<endl;

}

}

**void** rootProcess(){

**int** sendCount = 0;

**for** (**int** j = 0; j < processCount; j++){

**if** (claster\_row[j] == 1) {

MPI\_Send(&processRank, 1, MPI\_INT, j, ZOND, comm);

cout << "P" << processRank << " послал P" << j << " зонд"<< endl;

sendCount++;

}

}

**for** (**int** j = 0; j < processCount;j++){

topologyMatrix[processRank][j] = claster\_row[j];

}

**while** (sendCount) {

MPI\_Recv(&tmp\_topology, processCount \* processCount, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, comm, &status);

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

**for** (**int** j = 0; j < processCount; j++){

topologyMatrix[i][j] = topologyMatrix[i][j] || tmp\_topology[i][j];

}

}

sendCount--;

}

showTopology\_SpanningTree();

}

**void** nonRootProcess(){

**int** sendCount = 0;

**int** zondsCount = 0;

**int** parentRank;

**for** (**int** j = 0; j < processRank; j++){

**if** (claster\_row[j] == 1) {

zondsCount++;

}

}

**while**(zondsCount){

MPI\_Recv(&parentRank, 1, MPI\_INT, MPI\_ANY\_SOURCE, ZOND, comm, &status);

cout << "P" << processRank << " принял от P" << parentRank << " zond"<< endl;

zondsCount--;

}

**for** (**int** j = processRank; j < processCount; j++){

**if** (claster\_row[j] == 1) {

MPI\_Send(&processRank, 1, MPI\_INT, j, ZOND, comm);

cout << "P" << processRank << " послал P" << j << " zond"<< endl;

sendCount++;

}

}

**for** (**int** j=0;j<processCount;j++)

topologyMatrix[processRank][j] = claster\_row[j];

**while** (sendCount) {

MPI\_Recv(&tmp\_topology, processCount \* processCount, MPI\_INT, MPI\_ANY\_SOURCE, ECHO, comm, &status);

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

**for** (**int** j = 0; j < processCount; j++){

topologyMatrix[i][j] = topologyMatrix[i][j] || tmp\_topology[i][j];

}

}

sendCount--;

}

**for** (**int** j = 0; j < processRank; j++){

**if** (claster\_row[j] == 1) {

MPI\_Send(&topologyMatrix, processCount\*processCount, MPI\_INT, j, ECHO, comm);

cout << "P" << processRank << " послал P" << j << " echo с топологией"<< endl;

}

}

}

3.2 Результаты выполнения программы

**P0 послал P1 зонд**

**P0 послал P2 зонд**

**P2 принял от P0 zond**

**P1 принял от P0 zond**

**P2 послал P3 zond**

**P2 послал P4 zond**

**P3 принял от P2 zond**

**P4 принял от P2 zond**

**P1 послал P3 zond**

**P1 послал PP3 принял от P1 zond**

**4 zond**

**P4 принял от P1 zond**

**P3 послал P1 echo с топологией**

**PP1 послал P0 echo с топологией**

**3 послал P2 echo с топологией**

**P4 послал P1 echo с топологией**

**P4 послал P2 echo с топологией**

**P2 послал P0 echo с топологией**

**Topology matrix:**

**0 1 1 0 0**

**1 0 0 1 1**

**1 0 0 1 1**

**0 1 1 0 0**

**0 1 1 0 0**

**Spanning tree:**

**0 1 1 0 0**

**1 0 0 1 1**

**1 0 0 0 0**

**0 1 0 0 0**

**0 1 0 0 0**

**Процесс 0 послал процессу 1: 'May the forth be with you'**

**Процесс 1 послал процессу 4: 'May the forth be with you'**

**Процесс 0 послал процессу 2: 'May the forth be with you'**

**Процесс 1 послал процессу 3: 'May the forth be with you'**

**Program ended with exit code: 0**

ВЫВОДЫ

В ходе выполнения лабораторной работы было исследовано алгоритмическое построение методов взаимодействия распределено выполняющихся процессов.

Была создана программа, реализующая взаимодействия распределено выполняющихся параллельных процессов методом «зонд-эхо».