

Министерство науки и Высшего образования Российской Федерации
Севастопольский государственный университет
Кафедра ИС

Отчет
по лабораторной работе №4
«Исследование взаимодействий распределенных процессов типа Клиент-Сервер»
по дисциплине
«ТЕОРИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ И ПАРАЛЛЕЛЬНЫХ
ВЫЧИСЛЕНИЙ»

Выполнил студент группы ИС/б-17-2-о

Горбенко К. Н.

Проверил

Дрозин А.Ю.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

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

2 ПОСТАНОВКА ЗАДАЧИ

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 (путем передачи сообщения) в использование клиентам. Реализовать серверный процесс, который разграничивает доступ клиентов к этой процедуре (процедурам) и к ресурсам. Реализацию сервера выполнять в соответствии со схемой управления, использующую рассылку сообщений.

3 ХОД РАБОТЫ

Текст программы:

```

1 #include <mpi.h>
2 #include <iostream>
3 #include <sstream>
4 #include <iomanip>
5
6 #define CS_CONNECT 0
7 #define CS_DISCONNECT 1
8 #define CS_TAKE 2
9 #define CS_RETURN 3
10
11 #define SC_FREE 10
12 #define SC_INUSE 11
13 #define SC_SET 12
14 #define SC_NO_RESOURCE 13
15 #define SC_NO_ACTION 14
16
17 #define RES_COUNT 3
18
19 using namespace std;
20

```

```

21 void server() {
22     double resources[] = {1234.56, 12, 500};
23     int inUse[] = {0, 0, 0};
24
25     bool firstClientConnected = false;
26     int clients = 0, outTag, r_index;
27     double r_value;
28
29     double *inBuf = new double[2];
30     double *outBuf = new double[1];
31     MPI_Status status;
32     while (!firstClientConnected || clients) {
33         outBuf[0] = 0;
34         MPI_Recv(inBuf, 2, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG,
35                 MPI_COMM_WORLD, &status);
36         switch (status.MPI_TAG) {
37             case CS_CONNECT:
38                 firstClientConnected = true;
39                 clients++;
40                 break;
41             case CS_DISCONNECT:
42                 clients--;
43                 break;
44             case CS_TAKE:
45                 r_index = (int) inBuf[0];
46                 if (r_index < RES_COUNT) {
47                     if (inUse[r_index] != 0) {
48                         outTag = SC_INUSE;
49                     } else {
50                         outBuf[0] = resources[r_index];
51                         inUse[r_index] = status.MPI_SOURCE;
52                         outTag = SC_FREE;
53                     }
54                 } else {
55                     outTag = SC_NO_RESOURCE;
56                 }
57                 MPI_Send(outBuf, 1, MPI_DOUBLE, status.MPI_SOURCE, outTag,
58                         MPI_COMM_WORLD);
59                 break;
60             case CS_RETURN:
61                 r_index = (int) inBuf[0];
62                 r_value = inBuf[1];
63                 if (r_index < RES_COUNT) {
64                     if (inUse[r_index] != status.MPI_SOURCE) {
65                         outTag = SC_INUSE;
66                     } else {
67                         resources[r_index] = r_value;
68                         inUse[r_index] = 0;

```

```

67             outTag = SC_SET;
68         }
69     } else {
70         outTag = SC_NO_RESOURCE;
71     }
72     MPI_Send(outBuf, 1, MPI_DOUBLE, status.MPI_SOURCE, outTag,
73             MPI_COMM_WORLD);
74     break;
75 default:
76     MPI_Send(outBuf, 1, MPI_DOUBLE, status.MPI_SOURCE, SC_NO_ACTION
77             , MPI_COMM_WORLD);
78 }
79 }
80
81 stringstream ss;
82 ss << "\t" << "Server" << endl;
83 for (r_index = 0; r_index < RES_COUNT; r_index++) {
84     ss << "Resource #" << r_index
85     << " is " << fixed << setw(8) << setprecision(2) << resources[
86         r_index]
87     << endl;
88 }
89 ss << endl;
90 cout << ss.str();
91
92 delete[] inBuf;
93 delete[] outBuf;
94 }
95
96 double get_resource(int n) {
97     MPI_Status status;
98     double *inBuf = new double[1];
99     double *outBuf = new double[1];
100     outBuf[0] = n;
101     bool received = false;
102     while (!received) {
103         MPI_Send(outBuf, 1, MPI_DOUBLE, 0, CS_TAKE, MPI_COMM_WORLD);
104         MPI_Recv(inBuf, 1, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status)
105         ;
106         if (status.MPI_TAG == SC_FREE) {
107             received = true;
108         }
109     }
110     double ret_val = inBuf[0];
111     delete[] inBuf;
112     delete[] outBuf;
113     return ret_val;

```

```

111 }
112
113 bool set_resource(int n, double value) {
114     MPI_Status status;
115     double *inBuf = new double[1];
116     double *outBuf = new double[2];
117     outBuf[0] = n;
118     outBuf[1] = value;
119     MPI_Send(outBuf, 2, MPI_DOUBLE, 0, CS_RETURN, MPI_COMM_WORLD);
120     MPI_Recv(inBuf, 1, MPI_DOUBLE, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
121
122     delete[] inBuf;
123     delete[] outBuf;
124
125     return status.MPI_TAG == SC_SET;
126 }
127
128 void client(int rank) {
129     double *outBuf = new double[1];
130     outBuf[0] = 0;
131     MPI_Send(outBuf, 1, MPI_DOUBLE, 0, CS_CONNECT, MPI_COMM_WORLD);
132
133     stringstream ss;
134     ss << "\t" << "Client #" << rank << endl;
135     double r_value, r_value_changed;
136     for (int r_index = 0; r_index < RES_COUNT; r_index++) {
137         r_value = get_resource(r_index);
138         r_value_changed = r_value + (r_index + 10) * rank;
139         set_resource(r_index, r_value_changed);
140
141         ss << "Resource #" << r_index
142             << " was " << fixed << setw(8) << setprecision(2) << r_value
143             << " set " << fixed << setw(8) << setprecision(2) << r_value_changed
144             << endl;
145     }
146     ss << endl;
147     cout << ss.str();
148
149     MPI_Send(outBuf, 1, MPI_DOUBLE, 0, CS_DISCONNECT, MPI_COMM_WORLD);
150
151     delete[] outBuf;
152 }
153
154 int main(int argc, char **argv) {
155     int rank, size;
156     MPI_Init(&argc, &argv);
157     MPI_Comm_size(MPI_COMM_WORLD, &size);
158     MPI_Comm_rank(MPI_COMM_WORLD, &rank);

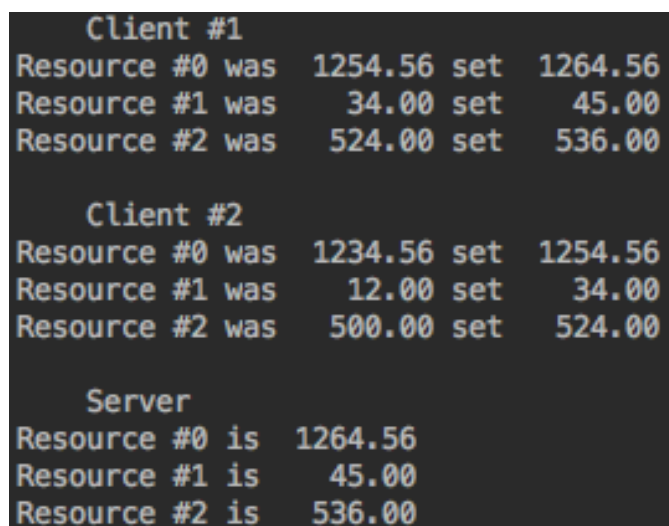
```

```

159     if (size != 3) {
160         if (rank == 0) {
161             cout << "Use only with 3 processes" << endl;
162             cout << "Exit..." << endl;
163         }
164         MPI_Finalize();
165         return 1;
166     }
167     !rank ? server() : client(rank);
168
169     MPI_Finalize();
170     return 0;
171 }

```

На рисунке 1 представлен результат работы программы:



```

Client #1
Resource #0 was 1254.56 set 1264.56
Resource #1 was 34.00 set 45.00
Resource #2 was 524.00 set 536.00

Client #2
Resource #0 was 1234.56 set 1254.56
Resource #1 was 12.00 set 34.00
Resource #2 was 500.00 set 524.00

Server
Resource #0 is 1264.56
Resource #1 is 45.00
Resource #2 is 536.00

```

Рисунок 1 – Результат работы программы

ВЫВОДЫ

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