```cpp
//-----------------------------------------------------------------------------
// 7A_Floyd_shortest_path.cpp
//-----------------------------------------------------------------------------
#ifdef _WIN32
   #pragma warning(disable:4996)
   #include <windows.h>
   #include <conio.h>
   #include <direct.h>
#elif (defined __linux__) || (defined _AIX)
   #include <stdlib.h>
   #include <sys/types.h>
   #include <sys/stat.h>
   #include <unistd.h>
   typedef char _TCHAR;
   #define _tmain main
#endif

#include <mpi.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
using namespace std;
//-----------------------------------------------------------------------------
#define MIN(A,b)                ((A)<(b)?(A):(b))
// napr. nech mame 3 procesy a 6 vrcholov grafu ciest
// (cislovanie riadkov matice ciest zacina od 0)
#define BLOCK_LOW(rank,size,n)   ((rank)*(n)/(size))
// napr. pre stredny pruh dostaneme 1 * 6 / 3 = 2
#define BLOCK_HIGH(rank,size,n)  (BLOCK_LOW((rank)+1,size,n)-1)
// 2 * 6 / 3 - 1 = 2 * 2 - 1 = 3
#define BLOCK_SIZE(rank,size,n)  (BLOCK_HIGH(rank,size,n)-BLOCK_LOW(rank,size,n)+1)
// 3 - 2 + 1 = 2

#define BLOCK_OWNER(j,size,n)    (((size)*((j)+1)-1)/(n))
// napr. nech j = 1, potom (3 * (1 + 1 -1)) / 6 = 3 / 6 = 0 (celociselne delenie)
// nech j = 2. potom (3 * (2 + 1 -1)) / 6 = 6 / 6 = 1
#define MAXLINE 255
// nech j = 4. potom (3 * (4 + 1 -1)) / 6 = 12 / 6 = 2
// nech j = 5. potom (3 * (5 + 1 -1)) / 6 = 15 / 6 = 2 (celociselne delenie)
//-----------------------------------------------------------------------------
void compute_shortest_paths(int, int, int[][6], int);
void GetNamesOfIOFiles(const char[], char[]);
//-----------------------------------------------------------------------------
int main(int argc, char* argv[])
{
   int A[6][6];   /* Doubly-subscripted array */
   int i, j, k;
   int size;      /* Number of processes */
   int rank;      /* Process rank */
   int m=0;       /* Rows in matrix */
   int n=0;       /* Columns in matrix */
   int tmp;
   FILE* in;
```

```
54     char nameoffile[MAXLINE];
55     double my_time;
56     MPI_Datatype my_type1;
57
58     MPI_Init(&argc, &argv);
59     MPI_Barrier(MPI_COMM_WORLD);
60     my_time = -MPI_Wtime();
61
62     MPI_Comm_size(MPI_COMM_WORLD, &size);
63     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
64     if (6 % size != 0) {
65        if (rank == 0)
66           printf("\nThe number of processes should be 1, 2, 3 or 6!\n");
67        goto end;
68     }
69     if (rank == 0)
70        cout << "\nThere are " << size << " processes.\n";
71
72     GetNamesOfIOFiles("distances.txt", nameoffile);
73     if (rank == 0) {
74        in = fopen(nameoffile, "r");
75        tmp = fscanf(in, "%i %i", &m, &n);
76        for (i = 0; i < m; i++)
77           for (j = 0; j < n; j++)
78              tmp = fscanf(in, "%i", &A[i][j]);
79        fclose(in);
80
81        if (m != n) {
82           cout << "Matrix must be square\n";
83           goto end;
84        }
85
86        printf("\nThe original graph of distances was:\n\n");
87        for (i = 0; i < m; i++) {
88           for (j = 0; j < n; j++)
89              printf("%4i", A[i][j]);
90           printf("\n");
91        }
92        printf("\n");
93     }
94     MPI_Bcast(&m, 1, MPI_INT, 0, MPI_COMM_WORLD);
95     MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
96
97     MPI_Type_vector(BLOCK_SIZE(rank, size, n) * 6, 1, 1, MPI_INT, &my_type1);
98     MPI_Type_commit(&my_type1);
99
100    if (rank == 0)
101       for (k = 1; k < size; k++) // posielame vodorovne pruhy matice A
102          MPI_Ssend(&(A[BLOCK_LOW(k, size, n)][0]), 1, my_type1, k, k,
103             MPI_COMM_WORLD);
104    else
105       MPI_Recv(&(A[0][0]), 1, my_type1, 0, rank, MPI_COMM_WORLD,
106          MPI_STATUSES_IGNORE);
```

```
107
108     compute_shortest_paths(rank, size, A, n);
109
110     if (rank > 0)
111        MPI_Ssend(&(A[0][0]), 1, my_type1, 0, rank, MPI_COMM_WORLD);
112     if (rank == 0)
113        for (k = 1; k < size; k++)
114           MPI_Recv(&(A[BLOCK_LOW(k, size, n)][0]), 1, my_type1, k, k, MPI_COMM_WORLD,
115              MPI_STATUSES_IGNORE);
116
117     if (rank == 0) {
118        printf("\nThe graph of the shortest distances is:\n\n");
119        for (i = 0; i < m; i++) {
120           for (j = 0; j < n; j++)
121              printf("%4i", A[i][j]);
122           printf("\n");
123        }
124        printf("\n");
125     }
126
127  end:
128     my_time += MPI_Wtime();
129     if (rank == 0)
130        printf("Time = %10.6f s\n", my_time);
131
132     fflush(stdout);
133     MPI_Finalize();
134
135     return 0;
136  }
137  //--------------------------------------------------------------------------
138  void compute_shortest_paths(int rank, int size, int A[][6], int n)
139  {
140     int  i, j, k;
141     int  offset;     /* Local index of broadcast row */
142     int  root;       /* Process controlling row to be beast */
143     int  tmp[6];     /* Holds the broadcast row */
144
145     for (j = 0; j < n; j++)
146        tmp[0] = 0;
147
148     for (k = 0; k < n; k++) {
149        root = BLOCK_OWNER(k, size, n);
150        if (root == rank) {
151           offset = k - BLOCK_LOW(rank, size, n);
152           // napr. ak rank = 1 a size = 3, potom k nadobuda hodnoty 0 a 1
153           for (j = 0; j < n; j++)
154              tmp[j] = A[offset][j];
155           // lebo kopie pruhov matice A sme ukladali na &A[0][0]
156           // tmp je kopia riadku matice A
157        }
158        // MPI_Bcast synchronizuje procesy ako kazde collective
159        // communication, t.j., vsetky procesy budu mat rovnake k
```

```
160         MPI_Bcast(tmp, n, MPI_INT, root, MPI_COMM_WORLD);
161         for (i = 0; i < BLOCK_SIZE(rank, size, n); i++)
162            for (j = 0; j < n; j++)
163               A[i][j] = MIN(A[i][j], A[i][k] + tmp[j]);
164         // hladame minimum z A[i][j] a vsetkych moznych A[i][k] + A[k][j]
165         // A[i][j] = MIN(A[i][j],A[i][k]+A[k][j]) !!!!!!!!!!!!!!!
166      }
167   }
168   //-------------------------------------------------------------------------
169   void GetNamesOfIOFiles(const char name_of_input_file[], char path_to_input_file[])
170   {
171      char current_path[MAXLINE];
172      current_path[0] = '\0';
173
174   #ifdef _WIN32
175      TCHAR exePath[MAXLINE];
176
177      HMODULE hModule = GetModuleHandle(NULL);
178      if (hModule != NULL) {
179         if (!GetModuleFileName(hModule, exePath, MAXLINE)) {
180            cout << "Nepodarila sa zistit cesta k exe-suboru.\n";
181            exit(1);
182         }
183      }
184      else {
185         cout << "Module handle is NULL.\n" << endl;
186         exit(1);
187      }
188
189      int iii;
190      bool flag = false;
191      for (iii = (int)wcslen(exePath); iii >= 0; iii--) {
192         if (!flag && exePath[iii] == '\\') {
193            current_path[iii + 1] = '\0';
194            flag = true;
195         }
196         if (flag)
197            current_path[iii] = (char)exePath[iii];
198      }
199   #elif (defined __linux__)
200      unsigned iii;
201      char line[MAXLINE];
202      FILE* fp;
203      if ((fp = popen("/bin/pwd", "r")) == NULL) {
204         perror("popen error");
205         exit(1);
206      }
207      if (fgets(line, MAXLINE, fp) == NULL) {
208         perror("fgets error");
209         exit(1);
210      }
211      pclose(fp);
212
```

```
213     iii = 0;
214     while (line[iii] != '\r' && line[iii] != '\n') {
215        current_path[iii] = line[iii];
216        iii++;
217     }
218     current_path[iii] = '\0';
219  #elif (defined _AIX)
220     unsigned iii;
221     char line[MAXLINE];
222     FILE* fp;
223     if ((fp = popen("user/bin/pwd", "r")) == NULL) {
224        perror("popen error");
225        exit(1);
226     }
227     if (fgets(line, MAXLINE, fp) == NULL) {
228        perror("fgets error");
229        exit(1);
230     }
231     pclose(fp);
232
233     iii = 0;
234     while (line[iii] != '\r' && line[iii] != '\n') {
235        current_path[iii] = line[iii];
236        iii++;
237     }
238     current_path[iii] = '\0';
239  #endif
240
241     path_to_input_file[0] = '\0';
242     strcat(path_to_input_file, current_path);
243  #if (defined __linux__) || (defined _AIX)
244     strcat(path_to_input_file, "/inputs/");
245  #elif (defined _WIN32)
246     strcat(path_to_input_file, "inputs\\");
247  #endif
248     strcat(path_to_input_file, name_of_input_file);
249  }
```