Compiling ..\src\Lab2.cpp:
Source file timestamp: 2016-03-14 04:06:45
Compiled at: 2016-03-14 06:33:13

```
 1. /**
 2. ------------------------------------------------------
 3. ---------------- PARALLEL PROGRAMMING ----------------
 4. ---------------------- LAB #2 ------------------------
 5. - Win32. Semaphores, events, mutex and critical section -
 6. ------------ A = sort(a*B + b*C*(MO*MK)) ---------------
 7. -----------------CREATED ON 12.03.2016-----------------
 8. ----------------BY OLEG PEDORENKO, IP-31----------------
 9. ------------------------------------------------------
10. */
11.
12. #include <iostream>
13. #include <windows.h>
14. #include "data.h"
15.
16. using namespace std;
17.
18. int main(int argc, char** argv)
19. {
20.    int size = 8;
21.    if(argc > 1)
22.    {
23.        size = atoi(argv[1]);
24.    }
25.
26.     HANDLE* T = initThreads(size);
27.    WaitForMultipleObjects(4, T, true, INFINITE);
28.    cout << "All threads finished successfully" << endl;
29.    return 0;
30. }
```

30 lines: No errors

Compiling ..\src\data.h:
Source file timestamp: 2016-03-14 04:04:56
Compiled at: 2016-03-14 06:33:13

```
 1. #ifndef DATA_H
 2. #define DATA_H
 3. #define M_VALUE 1
 4. #include <windows.h>
 5. #include <ctime>
 6. #include "Eigen/Dense"
 7.
 8. using namespace std;
 9. using namespace Eigen;
10.
11. HANDLE* initThreads(int size);
12.
13. #endif
```

13 lines: No errors

Compiling ..\src\data.cpp:
Source file timestamp: 2016-03-14 06:30:15

Compiled at: 2016-03-14 06:33:13

```
 1. #include "data.h"
 2. #include <iostream>
 3. #include <fstream>
 4. #include <string>
 5.
 6. int M_SIZE = 8;
 7. int P = 4;
 8. int H = M_SIZE/P;
 9.
10. RowVectorXd A, B, C;
11. MatrixXd MO, MK;
12. double a, b;
13.
14. CRITICAL_SECTION CS_1;
15. HANDLE Evt_1, Evt_2, Sem_1, Sem_2, Sem_3;
16. HANDLE Mut_1;
17. typedef DWORD WINAPI (*FP)(LPVOID);
18.
19. DWORD WINAPI F1(LPVOID lparam);
20. DWORD WINAPI F2(LPVOID lparam);
21. DWORD WINAPI F3(LPVOID lparam);
22. DWORD WINAPI F4(LPVOID lparam);
23.
24. MatrixXd inputMatrix();
25. RowVectorXd inputVector();
26. double inputScalar();
27. MatrixXd sortMatrix(MatrixXd m);
28.
29. HANDLE* initThreads(int size = 8)
30. {
31.    M_SIZE = size;
32.    H = M_SIZE / P;
33.
34.    HANDLE* T = new HANDLE[P];
35.    FP F[] = {&F1, &F2, &F3, &F4};
36.    HANDLE* S[] = {&Sem_1, &Sem_2, &Sem_3};
37.    HANDLE* E[] = {&Evt_1, &Evt_2};
38.
39.    cout << "Matrix and vector size: ";
40.     cout << M_SIZE << endl;
41.
42.    for(int i = 0; i < 2; i++)
43.    {
44.        *(E[i]) = CreateEvent(NULL, true, false, NULL);
45.    }
46.
47.    for(int i = 0; i < 3; i++)
48.    {
49.        *(S[i]) = CreateSemaphore(NULL, 0, 1, NULL);
50.    }
51.    InitializeCriticalSection(&CS_1);
52.
53.    Mut_1 = CreateMutex(NULL, false, NULL);
54.
55.    for(int i = 0; i < P; i++)
56.    {
57.        T[i] = CreateThread(
58.         NULL,
59.         20000000,
60.         F[i],
```

```
61.         NULL,
62.         0,
63.         NULL);
64.    }
65.
66.    return T;
67. }
68.
69. //Th = aBh + bCh*(MO*MKh)
70. void calcTh(
71.        RowVectorXd &T,
72.        const double &a,
73.        const RowVectorXd &B,
74.        const double &b,
75.        const RowVectorXd &C,
76.        const MatrixXd &MO,
77.        const MatrixXd &MK,
78.        int k)
79. {
80.    int i = k-1;
81.    int size = H;
82.    if(k == P)
83.    {
84.        size = M_SIZE - i*H;
85.    }
86.    T.segment(i*H, size) =
87.        a * B.segment(i*H, size) +
88.        b * (C * (MO * MK.block(0, i*H, M_SIZE, size)));
89. }
90.
91. ////////////////////////////  TASK 1 ////////////////////////////////////
92. DWORD WINAPI F1(LPVOID lparam)
93. {
94.    int threadNo = 1;
95.    double a1, b1;
96.    MatrixXd MO1;
97.
98.    WaitForSingleObject(Mut_1, INFINITE);
99.        cout << "Thread " << threadNo << " started" << endl;
100.   ReleaseMutex(Mut_1);
101.
102.   //Ввод A, B, MO
103.   A = RowVectorXd::Zero(M_SIZE);
104.   B = inputVector();
105.   MO = inputMatrix();
106.   //Сигнал о завершении ввода S2,3,4.1 (Evt_1)
107.   SetEvent(Evt_1);
108.   //Ожидание ввода в T4 W4.1 (Evt_2)
109.   WaitForSingleObject(Evt_2, INFINITE);
110.   //Вход в КС
111.   EnterCriticalSection(&CS_1);
112.       //Копирование OP a->a1, b->b1, MO->MO1
113.       a1 = a;
114.       b1 = b;
115.       MO1 = MO;
116.   //Выход из КС
117.   LeaveCriticalSection(&CS_1);
118.   //Счет Ah
119.   calcTh(A, a1, B, b1, C, MO1, MK, 1);
120.   //Сортировка Ah
121.   auto begin = A.segment(0, H).data();
122.   sort(begin, begin+H);
```



```
123.   //Ожидание завершения сортировки в T2 W2.1 (Sem_1)
124.   WaitForSingleObject(Sem_1, INFINITE);
125.   //Слияние первой половины A
126.   auto begin2 = A.segment(0, 2*H).data();
127.   inplace_merge(begin2, begin2 + H, begin2 + 2*H);
128.   //Ожидание слияния второй половины A, W4.2 (Sem_3)
129.   WaitForSingleObject(Sem_3, INFINITE);
130.   //Слияние вектора A
131.   auto begin3 = A.data();
132.   inplace_merge(begin3, begin3 + 2*H, begin3 + M_SIZE);
133.   //Вывод вектора A
134.   WaitForSingleObject(Mut_1, INFINITE);
135.   if(M_SIZE < 9)
136.   {
137.       cout << A << endl;
138.   }
139.   ReleaseMutex(Mut_1);
140.   WaitForSingleObject(Mut_1, INFINITE);
141.   cout << "Thread " << threadNo << " finished" << endl;
142.   ReleaseMutex(Mut_1);
143.   return true;
144. }
145.
146. ////////////////////////////  TASK 2 ////////////////////////////////////
147. DWORD WINAPI F2(LPVOID lparam)
148. {
149.   int threadNo = 2;
150.   double a2, b2;
151.   MatrixXd MO2;
152.
153.   WaitForSingleObject(Mut_1, INFINITE);
154.   cout << "Thread " << threadNo << " started" << endl;
155.   ReleaseMutex(Mut_1);
156.
157.   //Ожидание ввода в T1 W1.1 (Evt_1)
158.   WaitForSingleObject(Evt_1, INFINITE);
159.   //Ожидание ввода в T4 W4.1 (Evt_2)
160.   WaitForSingleObject(Evt_2, INFINITE);
161.   //Вход в КС
162.   EnterCriticalSection(&CS_1);
163.       //Копирование OP a->a1, b->b1, MO->MO1
164.       a2 = a;
165.       b2 = b;
166.       MO2 = MO;
167.   //Выход из КС
168.   LeaveCriticalSection(&CS_1);
169.   //Счет Ah
170.   calcTh(A, a2, B, b2, C, MO2, MK, 2);
171.   //Сортировка Ah
172.   auto begin = A.segment(H, H).data();
173.   sort(begin, begin+H);
174.   //Сигнал о сортировке Ah S1.2 (Sem_1)
175.   ReleaseSemaphore(Sem_1, 1, NULL);
176.
177.   WaitForSingleObject(Mut_1, INFINITE);
178.   cout << "Thread " << threadNo << " finished" << endl;
179.   ReleaseMutex(Mut_1);
180.   return true;
181. }
182.
183. ////////////////////////////  TASK 3 ////////////////////////////////////
184. DWORD WINAPI F3(LPVOID lparam)
```

```
185. {
186.    int threadNo = 3;
187.    double a3, b3;
188.    MatrixXd MO3;
189.
190.    WaitForSingleObject(Mut_1, INFINITE);
191.    cout << "Thread " << threadNo << " started" << endl;
192.    ReleaseMutex(Mut_1);
193.
194.    //Ожидание ввода в T1 W1.1 (Evt_1)
195.    WaitForSingleObject(Evt_1, INFINITE);
196.    //Ожидание ввода в T4 W4.1 (Evt_2)
197.    WaitForSingleObject(Evt_2, INFINITE);
198.    //Вход в КС
199.    EnterCriticalSection(&CS_1);
200.        //Копирование ОР a->a1, b->b1, MO->MO1
201.        a3 = a;
202.        b3 = b;
203.        MO3 = MO;
204.    //Выход из КС
205.    LeaveCriticalSection(&CS_1);
206.    //Счет Ah
207.    calcTh(A, a3, B, b3, C, MO3, MK, 3);
208.    //Сортировка Ah
209.    auto begin = A.segment(2*H, H).data();
210.    sort(begin, begin+H);
211.    //Сигнал о сортировке Ah S4.2 (Sem_2)
212.    ReleaseSemaphore(Sem_2, 1, NULL);
213.
214.    WaitForSingleObject(Mut_1, INFINITE);
215.    cout << "Thread " << threadNo << " finished" << endl;
216.    ReleaseMutex(Mut_1);
217.    return true;
218. }
219.
220. ////////////////////////////  TASK 4 ////////////////////////////////////
221. DWORD WINAPI F4(LPVOID lparam)
222. {
223.    int threadNo = 4;
224.    double a4, b4;
225.    MatrixXd MO4;
226.
227.    WaitForSingleObject(Mut_1, INFINITE);
228.    cout << "Thread " << threadNo << " started" << endl;
229.    ReleaseMutex(Mut_1);
230.    //Ввод a, b, C, MK
231.    a = inputScalar();
232.    b = inputScalar();
233.    C = inputVector();
234.    MK = inputMatrix();
235.    //Сигнал о завершении ввода S1,2,3.2 (Evt_2)
236.    SetEvent(Evt_2);
237.    //Ожидание ввода в T1 W1.1 (Evt_1)
238.    WaitForSingleObject(Evt_1, INFINITE);
239.    //Вход в КС
240.    EnterCriticalSection(&CS_1);
241.        //Копирование ОР a->a1, b->b1, MO->MO1
242.        a4 = a;
243.        b4 = b;
244.        MO4 = MO;
245.    //Выход из КС
246.    LeaveCriticalSection(&CS_1);
```



```
247.    //Счет Ah
248.    calcTh(A, a4, B, b4, C, MO4, MK, 4);
249.    //Сортировка Ah
250.    auto begin = A.segment(3*H, (M_SIZE - 3*H)).data();
251.    sort(begin, begin+(M_SIZE - 3*H));
252.    //Ожидание завершения сортировки в T3 W3.1 (Sem_2)
253.    WaitForSingleObject(Sem_2, INFINITE);
254.    //Слияние второй половины A
255.    auto begin2 = A.segment(2*H, (M_SIZE - 2*H)).data();
256.    inplace_merge(begin2, begin2 + H, begin2 + (M_SIZE - 2*H));
257.    //Сигнал о слиянии второй половины A, S1.3 (Sem_3)
258.    ReleaseSemaphore(Sem_3, 1, NULL);
259.
260.    WaitForSingleObject(Mut_1, INFINITE);
261.    cout << "Thread " << threadNo << " finished" << endl;
262.    ReleaseMutex(Mut_1);
263.    return true;
264. }
265.
266.
267. MatrixXd inputMatrix()
268. {
269.     MatrixXd res;
270.     #ifdef M_VALUE
271.     res = MatrixXd::Constant(M_SIZE, M_SIZE, M_VALUE);
272.     #endif
273.     #ifdef M_RANDOM
274.     res = MatrixXd::Random(M_SIZE, M_SIZE);
275.     res = (res + MatrixXd::Constant(M_SIZE, M_SIZE, 1.2)) * 50;
276.     #endif
277.     return res;
278. }
279.
280. RowVectorXd inputVector()
281. {
282.     RowVectorXd res;
283.     #ifdef M_VALUE
284.     res = RowVectorXd::Constant(M_SIZE, M_VALUE);
285.     #endif
286.     #ifdef M_RANDOM
287.    srand(time(0));
288.     res = RowVectorXd::Random(M_SIZE);
289.     res = (res + RowVectorXd::Constant(M_SIZE, 1.2)) * 50;
290.     #endif
291.     return res;
292. }
293.
294. double inputScalar()
295. {
296.    double res;
297.    #ifdef M_VALUE
298.     res = M_VALUE;
299.     #endif
300.     #ifdef M_RANDOM
301.     res = 1 + rand() % 100;
302.     res = res / ((double) (1 + rand() % 10));
303.     #endif
304.     return res;
305. }
305 lines: No errors
```