```
1 /********************************
2 * FILE: pmms.c
3 * AUTHOR: Connor Beardsmore - 15504319
4 * UNIT: OS200 Assignment S1 - 2016
5 * PURPOSE: Matrix multiplication using multithreading and POSIX mutexs
6 * LAST MOD: 07/05/16
7 * REQUIRES: pmms.h
8 ***********************************
9
10 #include "pmms.h"
11
12 //-----
13
14 int main(int argc, char* argv[])
15 {
16
      // ENSURE ONLY 6 COMMAND LINE ARGUMENTS ENTERED
17
     if ( argc != 6 )
18
19
         printf( "Usage: ./pmms[Matrix A File] [Matrix B File] [M] [N] [K]\n" );
         printf( "Please see README for detailed steps on how to run!\n" );
20
         return -1;
21
22
     }
23
     // RENAME COMMAND LINE ARGUMENTS FOR CODE READABILITY
24
25
     char* fileA = argv[1];
     char* fileB = argv[2];
26
27
    M = atoi(argv[3]);
28
    N = atoi(argv[4]);
29
    K = atoi(argv[5]);
30
     status = 0;
31
32
     // VALIDATE THAT M,N,K ARE ALL 1 OR MORE
33
     if ( ( M < 1 ) || ( N < 1 ) || ( K < 1 ) )
34
35
         printf( "ERROR - Matrix dimensions must bee positive value.\n" );
36
         return -1;
37
38
39
     // MAP MATRICES STRUCT TO ADDRESS SPACE, ASSIGN TO POINTERS
     first = (int*)malloc( M * N * sizeof(int) );
40
    second = (int*)malloc( N * K * sizeof(int) );
41
     product = (int*)malloc( M * K * sizeof(int) );
42
43
     // READ DATA FROM FILE INTO MATRIX SHARED MEMORY
44
     // ERROR CHECK TO CONFIRM THAT BOTH WORKED AS EXPECTED
4.5
46
     status = readFile( fileA, first, M, N );
47
     if ( status != 0 )
48
         freeMatrices( first, second, product );
49
50
         return -1;
51
52
     status = readFile( fileB, second, N, K );
53
     if ( status != 0 )
54
55
         freeMatrices( first, second, product );
56
         return -1;
57
58
     // INITIAL SUBTOTAL FIELDS TO "EMPTY"
59
     subtotal.value = SUBTOTAL EMPTY;
60
    subtotal.threadID = SUBTOTAL EMPTY;
61
     subtotal.rowNumber = SUBTOTAL EMPTY;
62
63
64
     // CREATE M THREADS IN A MALLOC'D ARRAY
65
     pthread t* producers = (pthread t*)malloc( sizeof(pthread t) * M );
66
     // INITIALISE THE SEMAPHORES
67
```

```
68
      status = createLocks(locks);
69
      if ( status != 0 )
70
          fprintf( stderr, "ERROR - creating POSIX mutex + conditions\n");
71
72
          freeMatrices( first, second, product );
73
          free( producers );
74
          return -1;
75
      }
76
     // THE 'M' CREATED THREADS EXECUTE PRODUCER FUNCTION
77
78
      // NO THREAD SPECIFIC DATA IS REQUIRED
79
      for ( int ii = 0; ii < M; ii++ )</pre>
80
81
          pthread create( &producers[ii], NULL, producer, NULL );
82
          // AUTOMATICALLY RELEASE SYSTEM RESOURCES UPON THREAD EXITING
83
          pthread detach( producers[ii] );
84
      }
85
86
      // PARENT THREAD EXECUTES CONSUMER FUNCTION
87
      consumer(NULL);
88
89
      // PARENT DESTORYS ALL SEMAPHORES
90
      status = destroyLocks(locks);
91
      if ( status != 0 )
92
93
          fprintf( stderr, "ERROR - destroying POSIX mutex + conditions\n");
94
          freeMatrices( first, second, product );
95
          free( producers );
96
          return -1;
97
     }
98
     // OUTPUT FINAL TOTAL
99
100
      printf( "Total: %d\n", grandTotal );
101
102
       // FREE ALL MALLOC'D MEMORY
103
       freeMatrices( first, second, product );
104
       free( producers );
105
106
       return 0;
107 }
108 //----
109 // FUNCTION: producer
110 // PURPOSE: Parent process consumes the subtotal + childPID create by children.
111
112 void* producer()
113 {
114
       int rowNumber = 0;
115
       int total = 0;
116
       int value;
117
118
       // THREAD DETERMINES WHICH ROW TO CALCULATE
119
       // MUTEX REQUIRED TO ACCESS rowNumber, SO EACH THREAD HAS DISTINCT VALUE
120
      pthread mutex lock( &locks.mutex );
121
           rowNumber = subtotal.rowNumber;
122
            subtotal.rowNumber = subtotal.rowNumber + 1;
123
       pthread mutex unlock( &locks.mutex );
124
125
       // CALCULATE OFFSETS TO CONVERT 1D ARRAYS TO VIRTUAL 2D
126
       int offsetA = rowNumber * N;
127
       int offsetC = rowNumber * K;
128
       for ( int ii = 0; ii < K; ii++ )</pre>
129
130
131
           value = 0;
132
133
            // CALCULATE ROW DATA
134
           for ( int jj = 0; jj < N; jj++ )</pre>
135
                value += first[offsetA + jj] * second[jj * K + ii];
136
137
           product[offsetC + ii] = value;
```

```
138
139
      // CALCULATE TOTAL OF ALL ELEMENTS IN ROW
140
      for ( int kk = 0; kk < K; kk++ )
141
142
          total += product[offsetC + kk];
143
144
      // WAIT FOR LOCK BEFORE ACCESSING SHARED DATA
     pthread mutex lock( &locks.mutex );
145
146
      while ( subtotal.value != 0 )
147
          // GIVE UP MUTEX LOCK WHILE WAITING FOR CONDITION
148
          pthread cond wait( &locks.empty, &locks.mutex );
149
150
          subtotal.value = total;
151
          subtotal.threadID = pthread_self();
152
     pthread_cond_signal( &locks.full );
153
154
     pthread mutex unlock( &locks.mutex );
155
156
      // THREAD FINISHES ONCE ROW CALCULATED
157
      pthread exit(0);
158 }
159
160 //----
161 // FUNCTION: consumer
162 // PURPOSE: Parent process consumes the subtotal + threadID create by thread.
164 void* consumer()
165 {
166
      grandTotal = 0;
167
168
       for ( int ii = 0; ii < M; ii++ )</pre>
169
170
       // WAIT FOR LOCK BEFORE ACCESSING SHARED DATA
171
         pthread mutex lock( &locks.mutex );
172
          while ( subtotal.value == 0 )
173
              // GIVE UP MUTEX LOCK WHILE WAITING FOR CONDITION
174
              pthread cond wait( &locks.full, &locks.mutex );
175
176
              // OUTPUT ROW TOTAL AND RESET SUBTOTAL VALUES
177
              printf( "Subtotal produced by thread with ID " );
178
              printf( "%ld: %d\n", subtotal.threadID, subtotal.value );
179
              grandTotal += subtotal.value;
180
              subtotal.value = SUBTOTAL EMPTY;
181
              subtotal.threadID = SUBTOTAL EMPTY;
182
183
             pthread cond signal ( &locks.empty );
184
          pthread mutex unlock( &locks.mutex );
185
      }
186
187
      return NULL;
188 }
189
190 //-----
191 // FUNCTION: createLocks
192 // EXPORT: status (int)
193 // PURPOSE: Initialise the Mutex and Conditions used for locks
194
195 int createLocks()
196 {
       // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
197
198
       int status = 0;
199
      status += pthread mutex init( &locks.mutex, NULL );
200
      status += pthread cond init( &locks.full, NULL );
201
      status += pthread cond init( &locks.empty, NULL );
202
       return status;
203 }
204
205 //-----
206 // FUNCTION: destroyLocks
207 // EXPORT: status (int)
```

```
208 // PURPOSE: Destroy the Mutex and Conditions used for locks
209
210 int destroyLocks()
211 {
      // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
212
213
      int status = 0;
214
     status += pthread mutex destroy( &locks.mutex );
215
     status += pthread_cond_destroy( &locks.full );
     status += pthread_cond destroy( &locks.empty );
216
217
     return status;
218 }
219
220 //-----
221 // FUNCTION freeMatrices
222 // IMPORT: first (int*), second (int*), third (int*)
223 // PURPOSE: Free's the malloc'd member associated with the matrices imported
224
225 void freeMatrices(int* first, int* second, int* product)
226 {
    free(first);
227
    free(second);
228
229
     free (product);
230 }
231
232 //-----
233 // FUNCTION: printMatrix()
234 // IMPORT: newMatrix (Matrix*)
235 // PURPOSE: Print matrix contents to std out for debugging purposes
236
237 void printMatrix(int* matrix, int rows, int cols)
238 {
239
      // OFFSET TO CALCULATE "ROWS" OF THE 1D ELEMENT ARRAY
240
     int offset = 0;
     printf("\n");
241
242
243
     // ITERATE OVER ENTIRE MATRIX AND PRINT EACH ELEMENT
     for ( int ii = 0; ii < rows; ii++ )</pre>
244
    {
245
      offset = ii * cols;
246
         for ( int jj = 0; jj < cols; jj++ )</pre>
247
248
            printf("%d ", matrix[ offset + jj ] );
249
250
         printf("\n");
251
252
     }
253 }
254
255 //-----
256 // FUNCTION: printMatrices
257 // IMPORT: first (int*), second (int*), product (int*)
258 // PURPOSE: Prints the contents of three different Matrices to std out
260 void printMatrices (int* first, int* second, int* third, int M, int N, int K)
261 {
        printMatrix(first, M, N);
262
        printMatrix(second, N, K);
263
        printMatrix(third, M, K);
264
265 }
266
267 //-----
268
```