

```

1  /*****
2  *   FILE: pmms.c
3  *   AUTHOR: Connor Beardsmore - 15504319
4  *   UNIT: OS200 Assignment S1 - 2016
5  *   PURPOSE: Matrix multiplication using multithreading and POSIX locks
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" );
20         printf( "Please see README for detailed steps on how to run!\n" );
21         return -1;
22     }
23
24     // RENAME COMMAND LINE ARGUMENTS FOR CODE READABILITY
25     char* fileA = argv[1];
26     char* fileB = argv[2];
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 POSITIVE VALUES
33     if ( ( M < 1 ) || ( N < 1 ) || ( K < 1 ) )
34     {
35         printf( "ERROR - Matrix dimensions must be positive value\n" );
36         return -1;
37     }
38
39     // MAP MATRICES ARRAYS TO ADDRESS SPACE, ASSIGN TO POINTERS
40     first = (int*)malloc( M * N * sizeof(int) );
41     second = (int*)malloc( N * K * sizeof(int) );
42     product = (int*)malloc( M * K * sizeof(int) );
43
44     // READ DATA FROM FILE INTO MATRIX SHARED MEMORY
45     // ERROR CHECK TO CONFIRM THAT BOTH WORKED AS EXPECTED
46     status = readFile( fileA, first, M, N );
47     if ( status != 0 )
48     {
49         freeMatrices( first, second, product );
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
59     // INITIAL SUBTOTAL FIELDS TO "EMPTY"
60     subtotal.value = SUBTOTAL_EMPTY;
61     subtotal.threadID = SUBTOTAL_EMPTY;
62     subtotal.rowNumber = SUBTOTAL_EMPTY;
63
64     // CREATE M THREADS IN A MALLOC'D ARRAY
65     pthread_t* producers = (pthread_t*)malloc( sizeof(pthread_t) * M );
66
67     // INITIALISE THE SEMAPHORES

```

```

68 status = createLocks(locks);
69 if ( status != 0 )
70 {
71     fprintf( stderr, "ERROR - creating POSIX mutex + conditions\n");
72     freeMatrices( first, second, product );
73     free( producers );
74     return -1;
75 }
76
77 // THE M CREATED THREADS EXECUTE PRODUCER FUNCTION
78 // NO THREAD SPECIFIC DATA IS REQUIRED
79 for ( int ii = 0; ii < M; ii++ )
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();
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
99 // OUTPUT FINAL TOTAL
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, ENSURES THREAD HAS DISTINCT VALUE
120     pthread_mutex_lock( &locks.mutex );
121
122     rowNumber = subtotal.rowNumber;
123     subtotal.rowNumber = subtotal.rowNumber + 1;
124
125     pthread_mutex_unlock( &locks.mutex );
126
127     // CALCULATE OFFSETS TO CONVERT 1D ARRAYS TO VIRTUAL 2D
128     int offsetA = rowNumber * N;
129     int offsetC = rowNumber * K;
130
131     // ACTUAL MULTIPLICATION CALCULATIONS
132     // SEE README FOR HOW THIS IS PERFORMED
133     for ( int ii = 0; ii < K; ii++ )
134     {
135         value = 0;
136
137         // CALCULATE ROW DATA

```

```

138     for ( int jj = 0; jj < N; jj++ )
139         value += first[offsetA + jj] * second[jj * K + ii];
140
141     product[offsetC + ii] = value;
142 }
143
144 // CALCULATE TOTAL OF ALL ELEMENTS IN ROW
145 for ( int kk = 0; kk < K; kk++ )
146     total += product[offsetC + kk];
147
148 // WAIT FOR LOCK BEFORE ACCESSING SHARED DATA
149 pthread_mutex_lock( &locks.mutex );
150 while ( subtotal.value != 0 )
151     // GIVE UP MUTEX LOCK WHILE WAITING FOR CONDITION
152     pthread_cond_wait( &locks.empty, &locks.mutex );
153
154     subtotal.value = total;
155     subtotal.threadID = pthread_self();
156
157     pthread_cond_signal( &locks.full );
158     pthread_mutex_unlock( &locks.mutex );
159
160 // THREAD FINISHES ONCE ROW CALCULATED
161 pthread_exit(0);
162 }
163
164 //-----
165 // FUNCTION: consumer
166 // PURPOSE: Parent process consumes the subtotal + threadID created by thread.
167
168 void* consumer()
169 {
170     grandTotal = 0;
171
172     // LOOP M TIMES FOR EACH ROW OF PRODUCT MATRIX
173     for ( int ii = 0; ii < M; ii++ )
174     {
175         // WAIT FOR LOCK BEFORE ACCESSING SHARED DATA
176         pthread_mutex_lock( &locks.mutex );
177         while ( subtotal.value == 0 )
178             // GIVE UP MUTEX LOCK WHILE WAITING FOR CONDITION
179             pthread_cond_wait( &locks.full, &locks.mutex );
180
181         // OUTPUT ROW TOTAL AND RESET SUBTOTAL VALUES
182         printf( "Subtotal produced by thread with ID " );
183         printf( "%ld: %d\n", subtotal.threadID, subtotal.value );
184         grandTotal += subtotal.value;
185         subtotal.value = SUBTOTAL_EMPTY;
186         subtotal.threadID = SUBTOTAL_EMPTY;
187
188         pthread_cond_signal( &locks.empty );
189         pthread_mutex_unlock( &locks.mutex );
190     }
191
192     return NULL;
193 }
194
195 //-----
196 // FUNCTION: createLocks
197 // EXPORT: status (int)
198 // PURPOSE: Initialise the Mutex and Conditions used for locks
199
200 int createLocks()
201 {
202     // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
203     int status = 0;
204     status += pthread_mutex_init( &locks.mutex, NULL );
205     status += pthread_cond_init( &locks.full, NULL );
206     status += pthread_cond_init( &locks.empty, NULL );
207     return status;

```

```

208 }
209
210 //-----
211 // FUNCTION: destroyLocks
212 // EXPORT: status (int)
213 // PURPOSE: Destroy the Mutex and Conditions used for locks
214
215 int destroyLocks()
216 {
217     // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
218     int status = 0;
219     status += pthread_mutex_destroy( &locks.mutex );
220     status += pthread_cond_destroy( &locks.full );
221     status += pthread_cond_destroy( &locks.empty );
222     return status;
223 }
224
225 //-----
226 // FUNCTION freeMatrices
227 // IMPORT: first (int*), second (int*), third (int*)
228 // PURPOSE: Free's the malloc'd arrays associated with the matrices imported
229
230 void freeMatrices(int* first, int* second, int* product)
231 {
232     free(first);
233     free(second);
234     free(product);
235 }
236
237 //-----
238 // FUNCTION: printMatrix()
239 // IMPORT: matrix (int*), rows (int), cols (int)
240 // PURPOSE: Print matrix contents to stdout for debugging purposes
241
242 void printMatrix(int* matrix, int rows, int cols)
243 {
244     // OFFSET TO CALCULATE "ROWS" OF THE 1D ELEMENT ARRAY
245     int offset = 0;
246
247     // ITERATE OVER ENTIRE MATRIX AND PRINT EACH ELEMENT
248     for ( int ii = 0; ii < rows; ii++ )
249     {
250         offset = ii * cols;
251         for ( int jj = 0; jj < cols; jj++ )
252         {
253             printf("%d ", matrix[ offset + jj ] );
254         }
255         printf("\n");
256     }
257 }
258
259 //-----
260 // FUNCTION: printMatrices
261 // IMPORT: first (int*), second (int*), product (int*), M,N,K (int)
262 // PURPOSE: Prints the contents of three different Matrices to stdout
263
264 void printMatrices(int* first, int* second, int* third, int M, int N, int K)
265 {
266     printMatrix(first, M, N);
267     printMatrix(second, N, K);
268     printMatrix(third, M, K);
269 }
270
271 //-----
272

```