

```
1 # Makefile For Matrix Multiplication w. Multi-threading
2 # OS200 Assignment
3 # Last Modified: 07/05/16
4 # Connor Beardsmore - 15504319
5
6 # MAKE VARIABLES
7 EXEC1 = pmms
8 OBJ1 = pmms.o fileIO.o
9 CFLAGS = -Wall -Wextra -std=c99 -lrt -pthread -D _XOPEN_SOURCE=500
10 CC = gcc
11
12
13 # RULES + DEPENDENCIES
14 $(EXEC1) : $(OBJ1)
15     $(CC) $(OBJ1) -o $(EXEC1) $(CFLAGS)
16
17 pmms.o : pmms.c pmms.h fileIO.h
18     $(CC) -c pmms.c $(CFLAGS)
19
20 fileIO.o : fileIO.c fileIO.h
21     $(CC) -c fileIO.c $(CFLAGS)
22
23 clean:
24     rm -f $(EXEC1) $(OBJ1)
25
```

```

1  /*****
2  *   FILE: pmms.h
3  *   AUTHOR: Connor Beardsmore - 15504319
4  *   UNIT: OS200 Assignment S1 - 2016
5  *   PURPOSE: Header file for pmms.c
6  *   LAST MOD: 07/05/16
7  *   REQUIRES: stdlib.h, pthread.h, fileIO.h
8  *****/
9
10 #pragma once
11
12 #include <stdlib.h>
13 #include <pthread.h>
14 #include "fileIO.h"
15
16 //-----
17 // CONSTANTS
18
19 #define SUBTOTAL_EMPTY 0
20
21 //-----
22 // STRUCT: Stores the value of subtotal and the ID of the thread that
23 //          created it. Also stores row number that the thread calculated.
24
25 typedef struct
26 {
27     int value;
28     long threadID;
29     int rowNumber;
30 } Subtotal;
31
32 //-----
33 // STRUCT: Stores 3 locks for use in producer-consumer problem.
34 //          Mutex provides mutual exclusion to data.
35 //          Full and empty are conditions that the producer and consumer
36 //          wait until they are met.
37
38 typedef struct
39 {
40     pthread_mutex_t mutex;
41     pthread_cond_t full;
42     pthread_cond_t empty;
43 } Synchron;
44
45 //-----
46 // GLOBAL VARIABLES FOR USE IN MULTITHREADS
47
48 Subtotal subtotal;
49 int grandTotal;
50 int status;
51 Synchron locks;
52
53 // MATRIX POINTERS AND DIMENSIONS
54 int* first;
55 int* second;
56 int* product;
57 int M;
58 int N;
59 int K;
60
61 //-----
62 // FUNCTION DECLARATIONS
63
64 void* producer();
65 void* consumer();
66 int destroyLocks();
67 int createLocks();

```

```
68 void freeMatrices(int*, int*, int*);
69 void printMatrix(int*, int, int);
70 void printMatrices(int*, int*, int*, int, int, int);
71
72 //-----
73
```

```

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" );
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 1 OR MORE
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 STRUCT 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(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
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, 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
129     for ( int ii = 0; ii < K; ii++ )
130     {
131         value = 0;
132
133         // CALCULATE ROW DATA
134         for ( int jj = 0; jj < N; jj++ )
135             value += first[offsetA + jj] * second[jj * K + ii];
136
137         product[offsetC + ii] = value;

```

```

138     }
139
140     // CALCULATE TOTAL OF ALL ELEMENTS IN ROW
141     for ( int kk = 0; kk < K; kk++ )
142         total += product[offsetC + kk];
143
144     // WAIT FOR LOCK BEFORE ACCESSING SHARED DATA
145     pthread_mutex_lock( &locks.mutex );
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
153     pthread_cond_signal( &locks.full );
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.
163
164 void* consumer()
165 {
166     grandTotal = 0;
167
168     for ( int ii = 0; ii < M; ii++ )
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 {
197     // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
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 {
212     // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
213     int status = 0;
214     status += pthread_mutex_destroy( &locks.mutex );
215     status += pthread_cond_destroy( &locks.full );
216     status += pthread_cond_destroy( &locks.empty );
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 {
227     free(first);
228     free(second);
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;
241     printf("\n");
242
243     // ITERATE OVER ENTIRE MATRIX AND PRINT EACH ELEMENT
244     for ( int ii = 0; ii < rows; ii++ )
245     {
246         offset = ii * cols;
247         for ( int jj = 0; jj < cols; jj++ )
248         {
249             printf("%d ", matrix[ offset + jj ] );
250         }
251         printf("\n");
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
259
260 void printMatrices(int* first, int* second, int* third, int M, int N, int K)
261 {
262     printMatrix(first, M, N);
263     printMatrix(second, N, K);
264     printMatrix(third, M, K);
265 }
266
267 //-----
268

```

```
1  /*****
2  *   FILE: fileIO.h
3  *   AUTHOR: Connor Beardsmore - 15504319
4  *   UNIT: OS200 Assignment S1 - 2016
5  *   PURPOSE: Header file for fileIO.c
6  *   LAST MOD: 07/05/16
7  *   REQUIRES: stdio.h
8  *****/
9
10 #pragma once
11 #include <stdio.h>
12
13 //-----
14 // FUNCTION DECLARATIONS
15
16 int readFile(char*, int*, int, int);
17
18 //-----
19
```



```

1  /*****
2  *  FILE: fileIO.c
3  *  AUTHOR: Connor Beardsmore - 15504319
4  *  UNIT: OS200 Assignment S1 - 2016
5  *  PURPOSE: Perform reading of matrix elements from a file
6  *  LAST MOD: 07/05/16
7  *  REQUIRES: fileIO.h
8  *****/
9
10 #include "fileIO.h"
11
12 //-----
13 // FUNCTION: readFile()
14 // IMPORT: filename (char*), matrix (int*), rows (int), cols (int)
15 // EXPORT: status (int)
16 // PURPOSE: Read matrix from file and store its elements in an int array
17
18 int readFile( char* filename, int* matrix, int rows, int cols)
19 {
20     int nRead;
21     int offset = 0;
22
23     // OPEN FILE AND CONFIRM NO ERRORS OCCURRED
24     FILE* f = fopen( filename, "r" );
25     if ( f == NULL )
26     {
27         perror( "ERROR - opening file!\n" );
28         return -1;
29     }
30
31     // ITERATE TO FILL ALL MATRIX ROWS
32     for ( int ii = 0; ii < rows; ii++ )
33     {
34         // ITERATE TO FILL ALL MATRIX COLS
35         offset = ii * cols;
36         for ( int jj = 0; jj < cols; jj++ )
37         {
38             nRead = fscanf( f, "%d", ( &matrix[offset + jj] ) );
39             if ( nRead < 0 )
40             {
41                 // CHECK THAT ENOUGH VALUES HAVE BEEN READ
42                 if ( (offset + jj) < (rows * cols - 1) )
43                 {
44                     fprintf( stderr, "ERROR - not enough matrix values\n" );
45                     return -1;
46                 }
47                 // CHECK THAT NO ERROR FORCED EARLY EXIT
48                 else if ( ferror(f) )
49                 {
50                     perror("ERROR - reading matrix file!\n");
51                     return -1;
52                 }
53             }
54         }
55     }
56
57     fclose(f);
58     return 0;
59 }
60 //-----
61

```

```
1 #! /bin/bash
2
3 # AUTHOR: Connor Beardsmore
4 # DATE: 07/04/16
5
6 # COMPILE PROGRAM
7 make tmms
8
9 # RUN PMMS PROGRAM FOR COMBINATION OF M N K VALUES
10 # FROM 1 TO 100
11
12 for i in {1..100};
13 do
14     for j in {1..100};
15     do
16         for k in {1..100};
17         do
18             echo $i $j $k
19             ./pmms allOnes.txt allOnes.txt $i $j $k
20         done
21     done
22 done
23
```