

```
1 # Makefile For Matrix Multiplication w. Multi-processing
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: stdio.h, stdlib.h, unistd.h, fcntl.h, semaphore.h, mman.h
8  *               stat.h, wait.h, fileIO.h
9  *****/
10
11 #pragma once
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <unistd.h>
16 #include <fcntl.h>
17 #include <semaphore.h>
18 #include <sys/mman.h>
19 #include <sys/stat.h>
20 #include <sys/wait.h>
21 #include "fileIO.h"
22
23 //-----
24 // CONSTANTS
25
26 #define SUBTOTAL_EMPTY 0
27
28 //-----
29 // STRUCT: Stores the value of subtotal and the ID of the child that
30 //          created it. Also stores row number that the child calculated.
31
32 typedef struct
33 {
34     int value;
35     int childPID;
36     int rowNum;
37 } Subtotal;
38
39 //-----
40 // STRUCT: Stores 3 locks for use in producer-consumer problem.
41 //          Mutex provides mutual exclusion to data.
42 //          Full and empty are semaphores to signal actions
43 //          from the producer and consumer.
44
45 typedef struct
46 {
47     sem_t mutex;
48     sem_t full;
49     sem_t empty;
50 } Synchron;
51
52 //-----
53 // FUNCTION PROTOTYPES
54
55 void producer(Synchron*, Subtotal*, int*, int*, int*, int, int);
56 void consumer(Synchron*, Subtotal*, int*, int);
57 int createLocks(Synchron*);
58 int destroyLocks(Synchron*);
59
60 //-----
61

```

```

1  /*****
2  *   FILE: pmms.c
3  *   AUTHOR: Connor Beardsmore - 15504319
4  *   UNIT: OS200 Assignment S1 - 2016
5  *   PURPOSE: Matrix multiplication using multiprocessing and shared memory
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 [MatrixA File] [MatrixB 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     int M = atoi( argv[3] );
28     int N = atoi( argv[4] );
29     int K = atoi( argv[5] );
30
31     // VALIDATE THAT M,N,K ARE ALL POSITIVE VALUES
32     if ( ( M < 1 ) || ( N < 1 ) || ( K < 1 ) )
33     {
34         fprintf(stderr, "ERROR - matrix dimensions must be positive values\n");
35         return -1;
36     }
37
38     // VARIABLE DECLARATIONS
39     int status = 0, total = 0, pid = -1;
40     int parentPID = getpid();
41     Subtotal* subtotal;
42     Synchron* locks;
43
44     // MATRIX POINTERS
45     int *first, *second, *product;
46
47     // FILE DESCRIPTORS
48     int firstFD, secondFD, productFD, subtotalFD, locksFD;
49
50     // CALCULATE TOTAL SIZE NEEDED FOR DATA OF THE 3 MATRICES
51     size_t firstSize = sizeof(int) * M * N ;
52     size_t secondSize = sizeof(int) * N * K;
53     size_t productSize = sizeof(int) * M * K;
54
55     // CREATE SHARED MEMORY SEGMENTS FOR 3 MATRICES, SUBTOTAL AND LOCKS
56     firstFD = shm_open( "matrixA", O_CREAT | O_RDWR, 0666 );
57     secondFD = shm_open( "matrixB", O_CREAT | O_RDWR, 0666 );
58     productFD = shm_open( "matrixC", O_CREAT | O_RDWR, 0666 );
59     subtotalFD = shm_open( "subtotal", O_CREAT | O_RDWR, 0666 );
60     locksFD = shm_open( "sync", O_CREAT | O_RDWR, 0666 );
61
62     // ENSURE THAT ALL MEMORY SEGMENTS WERE CREATED CORRECTLY
63     if ( (firstFD == -1) || (secondFD == -1) || (productFD == -1) ||
64         (subtotalFD == -1) || (locksFD == -1) )
65     {
66         fprintf( stderr, "ERROR - creating shared memory blocks\n" );
67         return -1;

```

```

68     }
69
70     // TRUNCATE SEGMENTS TO THE CORRECT SIZES
71     status += ftruncate( firstFD, firstSize );
72     status += ftruncate( secondFD, secondSize );
73     status += ftruncate( productFD, productSize );
74     status += ftruncate( subtotalFD, sizeof(Subtotal) );
75     status += ftruncate( locksFD, sizeof(Synchron) );
76     if ( status != 0 )
77     {
78         fprintf( stderr, "ERROR - setting shared memory size\n" );
79         return -1;
80     }
81
82     // MAP SHARED MEMORY SEGMENTS TO ADDRESS SPACE, ASSIGN TO POINTERS
83     first = (int*)mmap( 0, firstSize, PROT_READ | PROT_WRITE,
84                        MAP_SHARED, firstFD, 0 );
85     second = (int*)mmap( 0, secondSize, PROT_READ | PROT_WRITE,
86                        MAP_SHARED, secondFD, 0 );
87     product = (int*)mmap( 0, productSize, PROT_READ | PROT_WRITE,
88                        MAP_SHARED, productFD, 0 );
89     subtotal = (Subtotal*)mmap( 0, sizeof(Subtotal), PROT_READ | PROT_WRITE,
90                                MAP_SHARED, subtotalFD, 0 );
91     locks = (Synchron*)mmap( 0, sizeof(Synchron), PROT_READ | PROT_WRITE,
92                              MAP_SHARED, locksFD, 0 );
93
94     // READ DATA FROM FILE INTO MATRIX SHARED MEMORY
95     // ERROR CHECK TO CONFIRM THAT BOTH FILES WERE READ CORRECTLY
96     status = readFile( fileA, first, M, N );
97     if ( status != 0 )
98     {
99         // ERROR MESSAGE GIVEN WITHIN readFile()
100         return -1;
101     }
102     status = readFile( fileB, second, N, K );
103     if ( status != 0 )
104     {
105         return -1;
106     }
107
108     // INITIALIZE SUBTOTAL FIELDS TO "EMPTY" DEFAULT VALUE
109     subtotal->value = SUBTOTAL_EMPTY;
110     subtotal->childPID = SUBTOTAL_EMPTY;
111     subtotal->rowNum = SUBTOTAL_EMPTY;
112
113     // INITIALISE THE SEMAPHORES
114     status = createLocks(locks);
115     if ( status != 0 )
116     {
117         fprintf( stderr, "ERROR - creating POSIX semaphores\n" );
118         return -1;
119     }
120
121     // CREATE 10 CHILDREN PROCESSES
122     // SIGNAL TO AVOID CREATION OF ZOMBIE PROCESSES
123     // SEE REPORT OR README.md FOR DETAILS ON HOW THIS IS ACHIEVED
124     signal(SIGCHLD, SIG_IGN);
125     for ( int ii = 0; ii < M; ii++ )
126     {
127         if ( parentPID == getpid() )
128             pid = fork();
129
130         // CONSUMER. PARENT WAITS FOR SUBTOTAL TO NOT BE EMPTY.
131         // ONLY PARENT WILL HAVE pid != 0, AS FORK RETURNS 0 TO CHILDREN.
132         if ( pid != 0 )
133         {
134             consumer(locks, subtotal, &total, M);
135         }
136
137         // PRODUCER. CHILD STORES CALCULATION IN SUBTOTAL.
138         // ONLY CHILD WILL HAVE pid = 0, FORK RETURNS childPID TO PARENT.

```

```

138
139     if ( pid == 0 )
140     {
141         producer(locks, subtotal, first, second, product, N, K);
142         // CHILD EXITS IMMEDIATELY AFTER IT DOES CALCULATIONS
143         _exit(0);
144     }
145
146     // PARENT DESTORYS ALL SEMAPHORES
147     status = destroyLocks(locks);
148     if ( status != 0 )
149     {
150         fprintf( stderr, "ERROR - destroying POSIX semaphores\n");
151         return -1;
152     }
153
154     // UNLINK AND CLOSE SHARED MEMORY SEGMENTS
155     status += close(firstFD);
156     status += close(secondFD);
157     status += close(productFD);
158     status += close(subtotalFD);
159     status += close(locksFD);
160     if ( status != 0 )
161     {
162         fprintf( stderr, "ERROR - closing shared memory\n");
163         return -1;
164     }
165
166     // OUTPUT FINAL TOTAL
167     printf("Total: %d\n", total);
168
169     return 0;
170 }
171
172 //-----
173 // FUNCTION: producer
174 // IMPORT: locks (Synchron*), subtotal (Subtotal*),
175 //         first (Matrix*), second (Matrix*), product (Matrix*)
176 //         N (int), K (int)
177 // PURPOSE: Parent process consumes the subtotal + childPID create by children.
178 // NOTE: Numerous of imports is relatively high. For scope of this project and
179 //        readability, I will leave as is. If extending: convert to matrix array.
180
181 void producer( Synchron* locks, Subtotal* subtotal,
182               int* first, int* second, int* product, int N, int K)
183 {
184     int value, rowNumber, offsetA, offsetC;
185     int total = 0;
186
187     // MUTEX REQUIRED TO FIND WHICH ROWNUMBER CHILD WILL CALCULATE
188     sem_wait(&locks->mutex);
189
190     rowNumber = subtotal->rowNum;
191     subtotal->rowNum += 1;
192
193     sem_post(&locks->mutex);
194
195     // CALCULATE OFFSET TO MAKE VIRTUAL 2D ARRAY, FROM 1D ARRAY
196     offsetA = rowNumber * N;
197     offsetC = rowNumber * K;
198
199     // NO LOCKS NEEDED FOR ACTUALL CALCULATION
200     // VALUES READ / WRITTEN TO ARE INDEPENDENT BETWEEN CHILDREN
201     // ACTUAL MATRIX MULTIPLICATION CALCULATION. SEE README.md FOR DETAILS
202     for ( int ii = 0; ii < K; ii++ )
203     {
204         value = 0;
205
206         for ( int jj = 0; jj < N; jj++ )
207             value += first[offsetA + jj] * second[jj * K + ii];

```

```

208
209     product[offsetC + ii] = value;
210 }
211
212 // SUM ALL VALUES IN THE CALCULATED ROW
213 for ( int kk = 0; kk < K; kk++ )
214     total += product[offsetC + kk];
215
216 // WAIT FOR SUBTOTAL TO BE EMPTY AND GET SUBTOTAL LOCK
217 sem_wait(&locks->empty);
218     sem_wait(&locks->mutex);
219
220     // STORE CALCULATED RESULTS IN SHARED SUBTOTAL
221     subtotal->childPID = getpid();
222     subtotal->value = total;
223
224     sem_post(&locks->mutex);
225     sem_post(&locks->full);
226 // SIGNAL THAT SUBTOTAL IS NOW FULL AND RELEASE SUBTOTAL LOCK
227 }
228
229 //-----
230 // FUNCTION: consumer
231 // IMPORT: locks (Synchron*), subtotal (Subtotal*),
232 //         total (int*), productRows (int)
233 // PURPOSE: Parent process consumes the subtotal + childPID create by children.
234
235 void consumer(Synchron* locks, Subtotal* subtotal, int* total, int productRows)
236 {
237     for ( int ii = 0; ii < productRows; ii++ )
238     {
239         // WAIT FOR SUBTOTAL TO BE FULL AND GET SUBTOTAL LOCK
240         sem_wait(&locks->full);
241         sem_wait(&locks->mutex);
242
243         printf( "Subtotal produced by process with ID " );
244         printf( "%d: %d\n", subtotal->childPID, subtotal->value );
245         *total += subtotal->value;
246
247         // SET VALUES BACK TO EMPTY
248         subtotal->value = SUBTOTAL_EMPTY;
249         subtotal->childPID = SUBTOTAL_EMPTY;
250
251         sem_post(&locks->mutex);
252         sem_post(&locks->empty);
253         // SIGNAL THAT SUBTOTAL IS NOW EMPTY AND RELEASE SUBTOTAL LOCK
254     }
255 }
256
257 //-----
258 // FUNCTION: createLocks
259 // IMPORT: locks (Synchron*)
260 // EXPORT: status (int)
261 // PURPOSE: Create the 3 POSIX semaphores required for locks.
262
263 int createLocks(Synchron* locks)
264 {
265     // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
266     int status = 0;
267     status += sem_init( &locks->mutex, -1, 1 );
268     status += sem_init( &locks->full, -1, 0 );
269     status += sem_init( &locks->empty, -1, 1 );
270     return status;
271 }
272
273 //-----
274 // FUNCTION: destroyLocks
275 // IMPORT: locks (Synchron*)
276 // EXPORT: status (int)
277 // PURPOSE: Destroy the 3 POSIX semaphores created for locks.

```

```

278
279 int destroyLocks(Synchron* locks)
280 {
281     // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
282     int status = 0;
283     status += sem_destroy( &locks->mutex );
284     status += sem_destroy( &locks->full );
285     status += sem_destroy( &locks->empty );
286     return status;
287 }
288
289 //-----
290 // FUNCTION: printMatrix()
291 // IMPORT: newMatrix (Matrix*)
292 // PURPOSE: Print matrix contents to stdout for debugging purposes
293
294 void printMatrix(int* matrix, int rows, int cols)
295 {
296     // OFFSET TO CALCULATE "ROWS" OF THE 1D ELEMENT ARRAY
297     int offset = 0;
298
299     // ITERATE OVER ENTIRE MATRIX AND PRINT EACH ELEMENT
300     for ( int ii = 0; ii < rows; ii++ )
301     {
302         offset = ii * cols;
303         for ( int jj = 0; jj < cols; jj++ )
304         {
305             printf("%d ", matrix[ offset + jj ] );
306         }
307         printf("\n");
308     }
309 }
310
311 //-----
312 // FUNCTION: printMatrices
313 // IMPORT: first (int*), second (int*), product (int*), M, N ,K (ints)
314 // PURPOSE: Prints the contents of three different Matrices to stdout
315
316 void printMatrices(int* first, int* second, int* third, int M, int N, int K)
317 {
318     printMatrix(first, M, N);
319     printMatrix(second, N, K);
320     printMatrix(third, M, K);
321 }
322
323 //-----
324

```

```
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 //-----
62

```

```
1 #! /bin/bash
2
3 # AUTHOR: Connor Beardsmore
4 # DATE: 07/04/16
5
6 # COMPILE PROGRAM
7 make pmms
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
```