```c
/***************************************************************************
*  FILE: pmms.c
*  AUTHOR: Connor Beardsmore - 15504319
*  UNIT: OS200 Assignment S1 - 2016
*  PURPOSE: Matrix multiplication using multiprocessing and shared memory
*  LAST MOD: 07/05/16
*  REQUIRES: pmms.h
***************************************************************************/

#include "pmms.h"

//---------------------------------------------------------------------------

int main(int argc, char* argv[])
{
    // ENSURE ONLY 6 COMMAND LINE ARGUMENTS ENTERED
    if ( argc != 6 )
    {
        printf( "Usage: ./pmms [MatrixA File] [MatrixB File] [M] [N] [K]\n" );
        printf( "Please see README for detailed steps on how to run!\n" );
        return 1;
    }

    // RENAME COMMAND LINE ARGUMENTS FOR CODE READABILITY
    char* fileA = argv[1];
    char* fileB = argv[2];
    int M = atoi( argv[3] );
    int N = atoi( argv[4] );
    int K = atoi( argv[5] );

    // VALIDATE THAT M,N,K ARE ALL POSITIVE VALUES
    if ( ( M < 1 ) || ( N < 1 ) ||  ( K < 1 ) )
    {
        fprintf(stderr, "ERROR - matrix dimensions must be positive values\n");
        return -1;
    }

    // VARIABLE DECLARATIONS
    int status = 0, total = 0, pid = -1;
    int parentPID = getpid();
    Subtotal* subtotal;
    Synchron* locks;

    // MATRIX POINTERS
    int *first, *second, *product;

    // FILE DESCRIPTORS
    int firstFD, secondFD, productFD, subtotalFD, locksFD;

    // CALCULATE TOTAL SIZE NEEDED FOR DATA OF THE 3 MATRICES
    size_t firstSize = sizeof(int) *  M * N ;
    size_t secondSize = sizeof(int) * N * K;
    size_t productSize = sizeof(int) * M * K;

    // CREATE SHARED MEMORY SEGMENTS FOR 3 MATRICES, SUBTOTAL AND LOCKS
    firstFD = shm_open( "matrixA", O_CREAT | O_RDWR, 0666 );
    secondFD = shm_open( "matrixB", O_CREAT | O_RDWR, 0666 );
    productFD = shm_open( "matrixC", O_CREAT | O_RDWR, 0666 );
    subtotalFD = shm_open( "subtotal", O_CREAT | O_RDWR, 0666 );
    locksFD = shm_open( "sync", O_CREAT | O_RDWR, 0666 );

    // ENSURE THAT ALL MEMORY SEGMENTS WERE CREATED CORRECTLY
    if ( (firstFD == -1) || (secondFD == -1) || (productFD == -1) ||
                            (subtotalFD == -1) || (locksFD == -1) )
    {
        fprintf( stderr, "ERROR - creating shared memory blocks\n" );
        return -1;
    }

    // TRUNCATE SEGMENTS TO THE CORRECT SIZES
    status += ftruncate( firstFD, firstSize );
    status += ftruncate( secondFD, secondSize );
    status += ftruncate( productFD, productSize );
    status += ftruncate( subtotalFD, sizeof(Subtotal) );
    status += ftruncate( locksFD, sizeof(Synchron) );
    if ( status != 0 )
    {
        fprintf( stderr, "ERROR - setting shared memory size\n" );
        return -1;
    }

    // MAP SHARED MEMORY SEGMENTS TO ADDRESS SPACE, ASSIGN TO POINTERS
    first = (int*)mmap( 0, firstSize, PROT_READ | PROT_WRITE,
                                                  MAP_SHARED, firstFD, 0 );
    second = (int*)mmap( 0, secondSize, PROT_READ | PROT_WRITE,
                                                  MAP_SHARED, secondFD, 0 );
    product = (int*)mmap( 0, productSize, PROT_READ | PROT_WRITE,
                                                  MAP_SHARED, productFD, 0 );
    subtotal = (Subtotal*)mmap( 0, sizeof(Subtotal), PROT_READ | PROT_WRITE,
                                                  MAP_SHARED, subtotalFD, 0 );
    locks = (Synchron*)mmap( 0, sizeof(Synchron), PROT_READ | PROT_WRITE,
                                                  MAP_SHARED, locksFD, 0 );

    // READ DATA FROM FILE INTO MATRIX SHARED MEMORY
    // ERROR CHECK TO CONFIRM THAT BOTH FILES WERE READ CORRECTLY
    status = readFile( fileA, first, M, N );
    if ( status != 0 )
    {
        // ERROR MESSAGE GIVEN WITHIN readFile()
        return -1;
    }
    status = readFile( fileB, second, N, K );
    if ( status != 0 )
    {
        return -1;
    }

    // INITIALIZE SUBTOTAL FIELDS TO "EMPTY" DEFAULT VALUE
    subtotal->value = SUBTOTAL_EMPTY;
    subtotal->childPID = SUBTOTAL_EMPTY;
    subtotal->rowNum = SUBTOTAL_EMPTY;

    // INITIALISE THE SEMAPHORES
    status = createLocks(locks);
    if ( status != 0 )
    {
        fprintf( stderr, "ERROR - creating POSIX semaphores\n" );
        return -1;
    }

    // CREATE 10 CHILDREN PROCESSES
    // SIGNAL TO AVOID CREATION OF ZOMBIE PROCESSES
    // SEE REPORT OR README.md FOR DETAILS ON HOW THIS IS ACHEIVED
    signal(SIGCHLD, SIG_IGN);
     for ( int ii = 0; ii < M; ii++ )
         if ( parentPID == getpid() )
             pid = fork();

    // CONSUMER. PARENT WAITS FOR SUBTOTAL TO NOT BE EMPTY.
    // ONLY PARENT WILL HAVE pid != 0, AS FORK RETURNS 0 TO CHILDREN.
    if ( pid != 0 )
    {
        consumer( locks, subtotal, &total, M );
    }

    // PRODUCER. CHILD STORES CALCULATION IN SUBTOTAL.
    // ONLY CHILD WILL HAVE pid = 0, FORK RETURNS childPID TO PARENT.

    if ( pid == 0 )
    {
        producer( locks, subtotal, first, second, product, N, K );
        // CHILD EXITS IMMEDIATELY AFTER IT DOES CACLULATIONS
        _exit(0);
    }

    // PARENT DESTORYS ALL SEMAPHORES
    status = destroyLocks(locks);
    if ( status != 0 )
    {
        fprintf( stderr, "ERROR - destroying POSIX semaphores\n" );
        return -1;
    }

    // UNLINK AND CLOSE SHARED MEMORY SEGMENTS
    status += close(firstFD);
    status += close(secondFD);
    status += close(productFD);
    status += close(subtotalFD);
    status += close(locksFD);
    if ( status != 0 )
    {
        fprintf( stderr, "ERROR - closing shared memory\n" );
        return -1;
    }

    // OUTPUT FINAL TOTAL
    printf( "Total: %d\n", total );
    return 0;
}

//---------------------------------------------------------------------------
// FUNCTION: producer
// IMPORT: locks (Synchron*), subtotal (Subtotal*),
//         first (Matrix*), second (Matrix*), product (Matrix*)
//         N (int), K (int)
// PURPOSE: Parent process consumes the subtotal + childPID create by children.
// NOTE: Numerous of imports is relatively high. For scope of this project and
//       readbility, I will leave as is. If extending: convert to matrix array.

void producer( Synchron* locks, Subtotal* subtotal,
                            int* first, int* second, int* product, int N, int K)
{
    int value, rowNumber;
    int offsetA, offsetC;
    int total = 0;

    // MUTEX REQUIRED TO FIND WHICH ROWNUMBER CHILD WILL CALCULATE
    sem_wait(&locks->mutex);

        rowNumber = subtotal->rowNum;
        subtotal->rowNum += 1;

    sem_post(&locks->mutex);

    // CALCULATE OFFSET TO MAKE VIRTUAL 2D ARRAY, FROM 1D ARRAY
    offsetA = rowNumber * N;
    offsetC = rowNumber * K;

    // NO LOCKS NEEDED FOR ACTUAL CALCULATION
    // VALUES READ / WRITTEN TO ARE INDEPENDENT BETWEEN CHILDREN
    // ACTUAL MATRIX MULTIPLICATION CALCULATION. SEE README.md FOR DETAILS
    for ( int ii = 0; ii < K; ii++ )
    {
        value = 0;

        for ( int jj = 0; jj < N; jj++ )
            value += first[offsetA + jj] * second[jj * K + ii];

        product[offsetC + ii] = value;
    }

    // SUM ALL VALUES IN THE CALCULATED ROW
    for ( int kk = 0; kk < K; kk++ )
        total += product[offsetC + kk];

    // WAIT FOR SUBTOTAL TO BE EMPTY AND GET SUBTOTAL LOCK
    sem_wait(&locks->empty);
        sem_wait(&locks->mutex);

            // STORE CALCULATED RESULTS IN SHARED SUBTOTAL
            subtotal->childPID = getpid();
            subtotal->value = total;

        sem_post(&locks->mutex);
    sem_post(&locks->full);
    // SIGNAL THAT SUBTOTAL IS NOW FULL AND RELEASE SUBTOTAL LOCK
}

//------------------------------------------------------------------------------
// FUNCTION: consumer
// IMPORT: locks (Synchron*), subtotal (Subtotal*),
//         total (int*), productRows (int)
// PURPOSE: Parent process consumes the subtotal + childPID create by children.

void consumer(Synchron* locks, Subtotal* subtotal, int* total, int productRows)
{
    for ( int ii = 0; ii < productRows; ii++ )
    {
        // WAIT FOR SUBTOTAL TO BE FULL AND GET SUBTOTAL LOCK
        sem_wait(&locks->full);
            sem_wait(&locks->mutex);

            printf( "Subtotal produced by process with ID " );
            printf( "%d: %d\n", subtotal->childPID, subtotal->value );
            *total += subtotal->value;

            // SET VALUES BACK TO EMPTY
            subtotal->value = SUBTOTAL_EMPTY;
            subtotal->childPID = SUBTOTAL_EMPTY;

            sem_post(&locks->mutex);
        sem_post(&locks->empty);
        // SIGNAL THAT SUBTOTAL IS NOW EMPTY AND RELEASE SUBTOTAL LOCK
    }
}

//------------------------------------------------------------------------------
// FUNCTION: createLocks
// IMPORT: locks (Synchron*)
// EXPORT: status (int)
// PURPOSE: Create the 3 POSIX semaphores required for locks.

int createLocks(Synchron* locks)
{
    // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
    int status = 0;
    status += sem_init( &locks->mutex, -1, 1 );
    status += sem_init( &locks->full, -1, 0 );
    status += sem_init( &locks->empty, -1, 1 );
    return status;
}

//------------------------------------------------------------------------------
// FUNCTION: destroyLocks
// IMPORT: locks (Synchron*)
// EXPORT: status (int)
// PURPOSE: Destroy the 3 POSIX semaphores created for locks.
```

```
int destroyLocks(Synchron* locks)
{
    // IF ANY METHOD FAILS, STATUS WILL BE NON-ZERO
    int status = 0;
    status += sem_destroy( &locks->mutex );
    status += sem_destroy( &locks->full );
    status += sem_destroy( &locks->empty );
    return status;
}

//---------------------------------------------------------------------------
// FUNCTION: printMatrix()
// IMPORT: newMatrix (Matrix*), rows (int), cols (int)
// PURPOSE: Print matrix contents to stdout for debugging purposes

void printMatrix(int* matrix, int rows, int cols)
{
    // OFFSET TO CALCULATE "ROWS" OF THE 1D ELEMENT ARRAY
    int offset = 0;

    // ITERATE OVER ENTIRE MATRIX AND PRINT EACH ELEMENT
    for ( int ii = 0; ii < rows; ii++ )
    {
        offset = ii * cols;
        for ( int jj = 0; jj < cols; jj++ )
        {
            printf( "%d ", matrix[ offset + jj ] );
        }
        printf("\n");
    }
}

//--------------------------------------------------------------------------
// FUNCTION: printMatrices
// IMPORT: first (int*), second (int*), product (int*), M, N ,K (ints)
// PURPOSE: Prints the contents of three different Matrices to stdout

void printMatrices(int* first, int* second, int* third, int M, int N, int K)
{
        printMatrix( first, M, N );
        printMatrix( second, N, K );
        printMatrix( third, M, K );
}

//--------------------------------------------------------------------------
```