```
/*
 * File:   main.c
 * Author: Benno Waldhauer
 *
 * Created on 1. November 2010, 15:09
 */

/*

Things to do:

  - include MPI/OMP // --> dont work on sirius

Changes to the pthread Version:

  - include MPI multiplication
  - exclude OMP multiplication // --> dont work on sirius

*/

//#include <omp.h> // --> dont work on sirius
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>

#include "header.h"

#include "lib.c"
#include "logger.c"
#include "seq.c"
//#include "omp.c"
#include "pthread.c"
#include "mpi.c"

/*
 * main  A*B=C
 */

int main(int argc, char *argv[]) {

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  timestamp = time (NULL);  // gets the actuell time in s from 1.1.1970 as LogID in log file
  numOfProc=size; //number of slaves

  m=10;
  n=10;
  p=10;
  print =0;
  method =0;

if (rank == MASTER){ // Things the master has to do

  printf("\nWelcome\n\n A[m][n] * B[n][p] = C[m][p]\n\n");

  if(argc<4){ //if less then 3 arguments are providet, then ask vor the metrices dimensions
          printf("Please define m n and p!\n");
          printf("\n m = "); // ask for user input
          scanf("%d",&m); // scan user input
          printf(" n = ");
          scanf("%d",&n);
          printf(" p = ");
          scanf("%d",&p);
  }else{ // when min 3 arguments are providest, then use them
          m=strtol(argv[1], NULL, 10); // argument string to long int
          printf(" m = %d\n",m);
          n=strtol(argv[2], NULL, 10);
          printf(" n = %d\n",n);
          p=strtol(argv[3], NULL, 10);
```

```
        printf(" p = %d\n",p);
  }

  if(argc<5){ // if there is no 4th argument, then ask vor the printer options
        printf("\nChoose print option\n\n 0 = nothing\n 1 = print to screen\n \
2 = print to file\n 3 = print to screen and to file\n\n print = ");
        scanf("%d",&print);
  }else{ // 4th arg = printer option
        print=strtol(argv[4], NULL, 10);
        printf(" print = %d \n",print);
  }

  // method switch
  if(argc<6){
       printf("\nCompare results of parallel multiplications to\n\n 0 = nothing\n 1 = \n\n method =
");
        scanf("%d",&method);
  }
  else{
      method=strtol(argv[5], NULL, 10);
      printf(" method = %d",method);
  }

  printf("\n\nStart malloc of matrices ... \n\n");

  matrix A = {"A",m,n};
  matrix B = {"B",n,p};
  matrix C = {"SEQ",m,p};
  matrix D = {"MPI",m,p};
  matrix F = {"PTHREAD",m,p};

  A.matrix=mallocMatrix(A);     // returns the allocated Matrix now
  B.matrix=mallocMatrix(B); // A.matrix[height value][hight value] is available
  if(method == 1){
    C.matrix=mallocMatrix(C);
  }
  D.matrix=mallocMatrix(D);
  F.matrix=mallocMatrix(F);

  matrixInitRowPlusCol(A); // form lib.c
  matrixInitRowPlusCol(B);



  multiReturn seq;

  if(method == 1){
    seq=matrixMultiSEQ(A,B,C); // from seq.c
  }


  multiReturn mpi=matrixMultiMPI(A,B,D); // from mpi.c

  if(method == 1){
     matrixCompare(C,D); // compares the seq. and the mpi-marices-results (from lib.c)
  }


  multiReturn pth=matrixMultiPTH(A,B,F); // from pthread.c

  if(method == 1){
    matrixCompare(C,F);
  }

  printMatrix(A,print); // prints the matrices depending on the print parameter...
  printMatrix(B,print); // nothing, screen, file, both  (from lib.c)
  if(method == 1){
    printMatrix(C,print);
  }
  printMatrix(D,print);
  printMatrix(F,print);
```

```
  if(method == 1){
    logger(seq); // from log.c
  }
  logger(mpi);
  logger(pth);

  freeMatrix(A); // deallocate the matrices
  freeMatrix(B); // from lib.c
  if(method == 1){
    freeMatrix(C);
  }
  freeMatrix(D);
  freeMatrix(F);
  printf("\n");

}else{ // Things the slaves has to solve

  beMySlave(); // from mpi.c

}

  MPI_Finalize(); // END

  return (EXIT_SUCCESS);
}
```