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

/*

Things to do:

  - create switch for methods (seq,omp,pthread etc)
  - maybe create switch to change initialising matrices A and B
  - finish pthread

Changes to the sequential Version:

  - new datatyp "matrix" contains name, dimensions and the pointer to the matrix
  - all functions now works with the new datatyp
  - whole programm now is structured in folders and files,
    --> /log/ for everthing containing with log-files
    --> /source/ the whole sourcecode, seperated in semantik files

*/

#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "header.h"

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

/*
 * main  A*B=C
 */

//char method[10];

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

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

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

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

    (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);
  }    { // 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);
```

```
  }

    (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);
  }    { // 4th arg = printer option
      print=strtol(argv[4], NULL, 10);
      printf(" print = %d \n",print);
  }

  // future method switch
    (argc<6){
     // printf("\nChoose method of matrix computing\n\n 0 = sequential\n 1 = parallel (openMP)\n \
     2 = compare sequential and openMP\n method = ");

     //  scanf("%d",&method);
  }
      {
    //  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 = {"OMP",m,p};
  //matrix E = {"OMP2",m,p};


  A.matrix=mallocMatrix(A); // returns the allocated Matrix now
  B.matrix=mallocMatrix(B); // A.matrix[height value][hight value] is available
  C.matrix=mallocMatrix(C);
  D.matrix=mallocMatrix(D);
  //E.matrix=mallocMatrix(E);

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

  multiReturn seq=matrixMultiSEQ(A,B,C); // from seq.c

  multiReturn omp=matrixMultiOMP(A,B,D); // from omp.c
  //multiReturn omp2=matrixMultiOMP2(A,B,E);

  matrixCompare(C,D); // compares the seq. and the omp-marices-results (from lib.c)
  //matrixCompare(C,E);

  printMatrix(A,print); // prints the matrices depending on the print parameter...
  printMatrix(B,print); // nothing, screen, file, both  (from lib.c)

  printMatrix(D,print);
  //printMatrix(E,print);

  logger(seq); // from log.c
  logger(omp);
  //logger(omp2);



 // matrix F = {"PTHREAD",m,p};
 // F.matrix=mallocMatrix(F);
 // multiReturn pth=matrixMultiPTH(A,B,F);
 // matrixCompare(C,F);
 // printMatrix(F,print);
 // logger(pth);

  freeMatrix(A); // deallocate the matrices
  freeMatrix(B); // from lib.c
  freeMatrix(C);
  freeMatrix(D);
```

```
 // freeMatrix(E);
 // freeMatrix(F);

  printf("\n");

          (EXIT_SUCCESS);
}
```