Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №4

Win32.

Виконав:

студент групи ІП-32

Ковальчук О. М.

Перевірив:

Корочкін О. В.

Київ - 2015

Лабораторна робота №4. Win32

Мета роботи: вивчення засобів бібліотеки Win32 для роботи з потоками

Мова програмування: C++, використання бібліотеки Win32

Завдання: Розробити програму, що містить паралельні потоки, кожен з яких реалізує функції F1, F2, F3 з лабораторної роботи №1. Вимоги до потоків такі ж, як в лабораторній роботі № 2.

Функції:

F1: C = A - B * (MA * MD)

F2: o = Min(MK * MM)

F3: T = (MS * MZ) * (W + X)

Лістинг програми

cmdopts.cpp
```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#include "cmdopts.h"

char *getCmdOption(char **begin, char **end, const std::string &option) {
  char **itr = std::find(begin, end, option);
  if (itr != end && ++itr != end) {
    return *itr;
  }
  return 0;
}

bool cmdOptionExists(char **begin, char **end, const std::string &option) {
  return std::find(begin, end, option) != end;
}
```
functions.cpp
```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

```

```
#include "functions.h"

vector func1(vector a, vector b, matrix ma, matrix md) {
  return (a - b * (ma * md));
}

int func2(matrix mk, matrix mn) {
  return (min(mk*mn));
}

vector func3(matrix ms, matrix mz, vector w, vector x) {
  return ((ms * mz) * (w + x));
}
```

lab04.cpp

```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#include <iostream>
#include "functions.h"
#include "cmdopts.h"
#include "tasks.h"

int main(int argc, char* argv[]) {
  std::cout << "lab 04 started" << std::endl;
  int size, threadCount;

  {
    char *sizeopt = getCmdOption(argv, argv + argc, "-s");
    size = sizeopt ? atoi(sizeopt) : 4;
    char *threadopt = getCmdOption(argv, argv + argc, "-t");
    threadCount = threadopt ? atoi(threadopt) : 3;
  }

  std::vector<HANDLE> handles;
  HANDLE thread_;
  LPTHREAD_START_ROUTINE func;

  for (int i = 0; i < threadCount; ++i) {
    switch (i % 3) {
      case 0:
        func = (LPTHREAD_START_ROUTINE) task1;
        break;
      case 1:
        func = (LPTHREAD_START_ROUTINE) task2;
        break;
      default: // case 2
        func = (LPTHREAD_START_ROUTINE) task3;
        break;
    }
    thread_ = CreateThread(NULL, 0, func, (LPVOID)&size, CREATE_SUSPENDED, NULL);
    SetThreadPriority(thread_, THREAD_PRIORITY_NORMAL);
    handles.push_back(thread_);
  }

  for (int i = 0; i < handles.size(); ++i) {
    ResumeThread(handles[i]);
  }

  for (int i = 0; i < handles.size(); ++i) {
    WaitForSingleObject(handles[i], INFINITE);
    CloseHandle(handles[i]);
  }

  std::cout << "lab 04 finished" << std::endl;
        return 0;
}
```

matrix.cpp

```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#include "matrix.h"

matrix generateMatrix(int size, int filler) {
        return generateMatrix(size, size, filler);
}

matrix generateMatrix(int rows, int columns, int filler) {
        matrix result = matrix();
        for (int r = 0; r < rows; r++) {
                result.push_back(std::vector<int>(columns, filler));
        }
        return result;
}

matrix operator*(matrix left, matrix right) {
        assert(left.size() > 0);
        assert(right.size() > 0);
        assert(left[0].size() == right.size());

        matrix result = generateMatrix(left.size(), right[0].size(), 0);
        for (int i = 0; i < left.size(); ++i) {
                for (int j = 0; j < right[0].size(); ++j) {
                        for (int k = 0; k < left[0].size(); ++k) {
                                result[i][j] += left[i][k] * right[k][j];
                        }
                }
        }
        return result;
}

int min(matrix mtrx) {
        int result = mtrx[0][0];
        for (int r = 0; r < mtrx.size(); ++r) {
                for (int c = 0; c < mtrx[r].size(); ++c) {
                        if (mtrx[r][c] < result) {
                                result = mtrx[r][c];
                        }
                }
        }
        return result;
}
```

tasks.cpp

```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#include "tasks.h"

void task1(LPVOID lpSize) {
  std::cout << "task 1 in thread " << GetCurrentThreadId() << " started" << std::endl;
  int size = *((int *)lpSize);

  Sleep(500);

  vector a, b;
  matrix ma, md;

  a = generateVector(size);
  b = generateVector(size);
  ma = generateMatrix(size);
  md = generateMatrix(size);

  vector result = func1(a, b, ma, md);

  if (size < 8) {
    std::stringstream ss;
    ss << "task 1 result: ";
    ss << '[';
    for (int i = 0; i < result.size(); ++i) {
      ss << result[i] << " ";
    }
    ss << ']';

    std::cout << ss.str() << std::endl;
  }

  std::cout << "task 1 in thread " << GetCurrentThreadId() << " finished" << std::endl;
}

void task2(LPVOID lpSize) {
  std::cout << "task 2 in thread " << GetCurrentThreadId() << " started" << std::endl;
  int size = *((int *)lpSize);

  Sleep(500);

  matrix mk, mn;

  mk = generateMatrix(size);
  mn = generateMatrix(size);

  int result = func2(mk, mn);

  if (size < 8) {
    std::stringstream ss;
    ss << "task 2 result: " << result;
    std::cout << ss.str() << std::endl;
  }

  std::cout << "task 2 in thread " << GetCurrentThreadId() << " finished" << std::endl;
}

void task3(LPVOID lpSize) {
  std::cout << "task 3 in thread " << GetCurrentThreadId() << " started" << std::endl;
  int size = *((int *) lpSize);
  Sleep(500);

  vector w, x;
  matrix ms, mz;

  w = generateVector(size);
  x = generateVector(size);
  ms = generateMatrix(size);
  mz = generateMatrix(size);

  vector result = func3(ms, mz, w, x);

  if (size < 8) {
    std::stringstream ss;
    ss << "task 3 result: ";
    ss << '[';
    for (int i = 0; i < result.size(); ++i) {
      ss << result[i] << " ";
    }
    ss << ']';

    std::cout << ss.str() << std::endl;
  }

  std::cout << "task 3 in thread " << GetCurrentThreadId() << " finished" << std::endl;
}
```
vector.cpp
```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#include "vector.h"

vector generateVector(int size, int filler) {
  vector result = vector(size, filler);
  return result;
}

vector operator*(matrix left, vector right) {
  assert(left.size() > 0);
  assert(right.size() > 0);
  assert(right.size() == left[0].size());

  vector result = generateVector(left.size(), 0);
  for (int i = 0; i < left.size(); ++i) {
    for (int j = 0; j < right.size(); ++j) {
      result[i] += left[i][j] * right[j];
    }
  }
  return result;
}

vector operator*(vector left, matrix right) {
  assert(left.size() > 0);
  assert(right.size() > 0);
  assert(left.size() == right[0].size());

  vector result = generateVector(left.size(), 0);
  for (int i = 0; i < right[0].size(); ++i) {
    for (int j = 0; j < right.size(); ++j) {
      result[i] += right[i][j] * left[j];
    }
  }
  return result;
}

vector operator+(vector left, vector right) {
  assert(left.size() > 0);
  assert(left.size() == right.size());

  vector result = vector(left);
  for (int i = 0; i < result.size(); ++i) {
    result[i] += right[i];
  }

  return result;
}

vector operator-(vector left, vector right) {
    assert(left.size() > 0);
    assert(left.size() == right.size());

    vector result = vector(left);
    for (int i = 0; i < result.size(); ++i) {
      result[i] -= right[i];
    }

    return result;
}
```
cmdopts.h
```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#ifndef LAB_CMD_OPTS
#define LAB_CMD_OPTS

#include <algorithm>
#include <string>

char *getCmdOption(char **begin, char **end, const std::string &option);
bool cmdOptionExists(char **begin, char **end, const std::string &option);

#endif
```

functions.h
```
/**
 * Parallel programming
 * Lab 4
 *
 * Functions:
 * F1: C = A - B * (MA * MD)
 * F2: o = Min(MK * MM)
 * F3: T = (MS * MZ) * (W + X)
 *
 * @since 2015-10-18
 * @author Olexandr Kovalchuk
 * @group IP-32
 */

#ifndef LAB_FUNCTIONS_H
#define LAB_FUNCTIONS_H

#include "matrix.h"
#include "vector.h"

vector func1(vector a, vector b, matrix ma, matrix md);
int func2(matrix mk, matrix mn);
vector func3(matrix ms, matrix mz, vector w, vector x);

#endif // LAB_FUNCTIONS_H

```
matrix.h
```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#ifndef LAB_MATRIX_H
#define LAB_MATRIX_H

#include <vector>
#include <assert.h>

typedef std::vector<std::vector<int>> matrix;

matrix generateMatrix(int size, int filler = 1);
matrix generateMatrix(int rows, int columns, int filler = 1);

matrix operator*(matrix left, matrix right);
int min(matrix mtrx);

#endif // LAB_MATRIX_H
```
tasks.h
```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
```

```
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#ifndef LAB_TASKS_H
#define LAB_TASKS_H

#include <iostream>
#include "functions.h"
#include <windows.h>
#include <sstream>

void task1(LPVOID lpSize);
void task2(LPVOID lpSize);
void task3(LPVOID lpSize);

#endif
```

vector.h

```
/**
* Parallel programming
* Lab 4
*
* Functions:
* F1: C = A - B * (MA * MD)
* F2: o = Min(MK * MM)
* F3: T = (MS * MZ) * (W + X)
*
* @since 2015-10-18
* @author Olexandr Kovalchuk
* @group IP-32
*/

#ifndef LAB_VECTOR_H
#define LAB_VECTOR_H

#include <vector>
#include "matrix.h"
#include <assert.h>
typedef std::vector<int> vector;

vector generateVector(int size, int filler = 1);

vector operator*(matrix left, vector right);
vector operator*(vector left, matrix right);
vector operator+(vector left, vector right);
vector operator-(vector left, vector right);

#endif // LAB_VECTOR_H

```