Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и

информатики

Кафедра прикладной математики и кибернетики

Отчёт по лабораторной работе №8

Выполнил: студент группы ИП–811 Адов А.С.

Проверил: Малков Е.А.

Новосибирск 2020

**Цель**: познакомиться с принципами синхронизации потоков в части разрешения состязательных ситуаций.

**Задание**:

•программно реализовать алгоритм Петерсона;

•разработать программу для сравнения производительности собственной реализации алгоритма Петерсона и механизма CriticalSection Windows API (лекция 9);

•провести сравнение производительности.

***lab\_9.c***

#include <windows.h>

#include <time.h>

#include <synchapi.h>

#include <stdio.h>

#include <process.h>

#define SEVERAL\_ITERATION 10

#define ERROR\_ALLOCATION\_MEMORY 0b0000001

#define START\_GLOBAL\_VALUE 0

#define QUANTITY\_THREAD 2

#define MAX\_GLOBAL\_VALUE 10000

#define HEAD\_TABLE "Algorithm by Peterson:\n| Expected| Real | Time |\n"

CRITICAL\_SECTION critical\_section;

int global\_value = START\_GLOBAL\_VALUE; //sh==global\_value

int ready\_flag[QUANTITY\_THREAD] = {0};

int other\_thread = 0, turn = 0;

void Enter\_Critical\_Section(int threadId)

{

other\_thread = (1 - threadId); //get number other thread

ready\_flag[threadId] = 1; //up ready flag

turn = threadId; //get number present thread

while(turn == threadId && ready\_flag[other\_thread]); //cycle for wait (while other thread dont change ready flag)

}

void Leave\_Critical\_Section(int threadId)

{

ready\_flag[threadId] = 0; //down ready flag

}

DWORD WINAPI Thread\_For\_First\_Method(void\* param)

{

int c = \*((int\*)param);

for(int i = 0; i < MAX\_GLOBAL\_VALUE; i++)

{

Enter\_Critical\_Section(c);

global\_value++;//increment

Leave\_Critical\_Section(c);

}

return 0;

}

DWORD WINAPI Thread\_For\_Second\_Method(void\* param)

{

for(int i = 0; i < MAX\_GLOBAL\_VALUE; i++)

{

EnterCriticalSection(&critical\_section);//

global\_value++;

LeaveCriticalSection(&critical\_section);

}

return 0;

}

int main()

{

long int value\_time = 0;

int array\_number\_thread[QUANTITY\_THREAD];

HANDLE lpHandles[QUANTITY\_THREAD]; //array handle type for descriptors threads for peterson

DWORD dwThreadId[QUANTITY\_THREAD]; //array dword type for id threads for peterson

HANDLE lpHandlesCS[QUANTITY\_THREAD]; //array handle type for descriptors threads for critical section

DWORD dwThreadIdCS[QUANTITY\_THREAD]; //array dword type for id threads for critical section

InitializeCriticalSection(&critical\_section); //initialization critical section

for(short int i = 0; i < QUANTITY\_THREAD; i++) //fill array number thread by increment

{

array\_number\_thread[i] = i;

}

value\_time = clock(); //get time before start peterson

for(short int i = 0; i < QUANTITY\_THREAD; i++)

{

lpHandles[i] = CreateThread(NULL, 0, Thread\_For\_First\_Method, &array\_number\_thread[i], 0, &dwThreadId[i]); //function for create thread

//first parametr == NULL: for handle couldn't be inherited child process

//second parametr == 0: first size stack(if == 0 then used standart size)

//third parametr == my function: pointer for function which should be executed by the thread

//fourth parametr == &number\_thread[i]: pointer for variable which should be to thread

//fifth parametr == 0: flags for control create thread(if == 0 then start thread immediately after creation)

//sixth parametr == &thread\_id: pointer for variable which get id thread

SetThreadAffinityMask(lpHandles[i], 0x01); //set mask for specified thread

//first parametr == thread\_handle[i]: descriptor for thread

//second parametr == 0x01: allows thread start only in this cores (if == 0x01 then start in first core)

}

WaitForMultipleObjects(QUANTITY\_THREAD, lpHandles, TRUE, INFINITE);

//first parametr == quant\_thread: for number of object handles in the second parametr

//second parametr == thread\_handle: array of object handles

//third parametr == true: the function returns when the state of (if == true then all) objects in the second parameter

//fourth parametr == ifinite: interval time-out (if == infinite then fucntion returns only when objects are signaled)

value\_time = (clock() - value\_time); //get difference time present(after peterson) and time before start

printf(HEAD\_TABLE);

printf("| %d | %d | %ld | \n", (MAX\_GLOBAL\_VALUE \* QUANTITY\_THREAD), global\_value, value\_time);

global\_value = 0; //

value\_time = clock();

for(short int i = 0; i < QUANTITY\_THREAD; i++)

{

lpHandlesCS[i] = CreateThread(NULL, 0, Thread\_For\_Second\_Method, &array\_number\_thread[i], 0, &dwThreadIdCS[i]);

SetThreadAffinityMask(lpHandlesCS[i], 0x01);

}

WaitForMultipleObjects(QUANTITY\_THREAD, lpHandlesCS, TRUE, INFINITE);

value\_time = (clock() - value\_time);

printf("WinAPI:\n");

printf("| %d | %d | %ld | \n", (MAX\_GLOBAL\_VALUE \* QUANTITY\_THREAD), global\_value, value\_time);

for(short int i = 0; i < QUANTITY\_THREAD; i++) //functions for close handle for open object

{

CloseHandle(lpHandles[i]);

CloseHandle(lpHandlesCS[i]);

}

return 0;

}

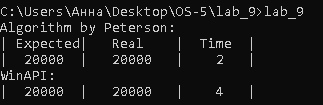


Рис.1 «Результат выполнения программы ***lab\_9.c***»

**Заключение**

Познакомился с принципами синхронизации потоков в части разрешения состязательных ситуаций.