Выполнила: Белоусова Е., ИП-911

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

Описание работы программы

Перед использованием общих переменных (то есть перед входом в свою критическую область) каждый процесс вызывает функцию EnterCriticalRegion(), передавая ей в качестве аргумента свой собственный номер процесса. Этот вызов заставляет процесс ждать, если потребуется, безопасного входа в критическую область. После завершения работы с общими переменными процесс, чтобы показать это и разрешить вход другому процессу, если ему это требуется, вызывает функцию LeaveCriticalRegion().

Для работы алгоритма на n потоках создадим массив размера n типа bool, в котором будем отмечать желание потоков войти в критическую зону, и создадим переменную, в которой будет содержаться номер потока, который в критической зоне. При входе в критическую зону поток отметит свое желание флагом. Далее попробует найти процесс, который также заинтересован во входе в КЗ. Если такие находятся, то текущий процесс будет ожидать конца выполнения, иначе же вернет управление. Когда поток выходит из критической зоны, он убирает флаг входа в критическую зону.

```
Peterson Result: 2000
Peterson Time: 1.570542

WinAPI Result: 2000
WinAPI Time (cs): 1.582113
```

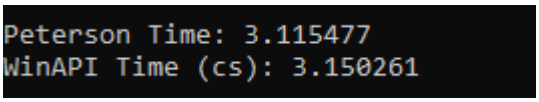
```
Peterson Result: 2000
Peterson Time: 1.569395

WinAPI Result: 2000
WinAPI Time (cs): 1.582062
```

Для того, чтобы алгоритм Петерсона работал корректно, запустим программу на одном ядре (start /affinity 1 1.exe). (Из-за сложности конструкции конвейерных процессоров инструкции могут выполняться в другом порядке. Кроме того, если потоки работают на разных ядрах, что не гарантирует немедленную согласованность кеша, потоки могут использовать разные значения памяти.)

Можно увидеть, что алгоритм Петерсона чаще всего работает быстрее.

Среднее время по результатам 10 экспериментов:



Листинг

```
#include <windows.h>

#include <thread>

#include <stdio.h>

#include <chrono>

#include <process.h>

#define N 20

#define MAX 100

//cl /MT /D "_X86_" /EHsc 1.cpp

//cmd.exe /k "C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\VC\Auxiliary\Build\vcvars32.bat" `& powershell

using namespace std;


CRITICAL_SECTION cs;

int Count = 0;

bool readyFlags[N];

int turn;


/*

int turn[N - 1];

void EnterCriticalRegion(int tID)

{

	for (int level = 0; level < N - 1; ++level)

	{

		readyFlags[tID] = level + 1;
```

```
                turn[level] = tID;

                while(true)
                {
                        int found = false;
                        for (int thread = 0; !found && thread != N; ++thread)
                        {
                                if (thread == tID) {continue;}
                                found = readyFlags[thread] > level;
                        }
                        if (!found) break;
                        if (turn[level] != tID) break;
                }
        }
}

void LeaveCriticalRegion(int tID)
{
        readyFlags[tID] = false;
}
*/

void EnterCriticalRegion(int tID)
{
        readyFlags[tID] = true;
        int i = 0;
        for (i = 0; i < N; i++)
        {
                if(readyFlags[i] && i != tID)
                {
                        turn = i;
                        break;
                }
        }
        while(turn == i && readyFlags[i]);
```

```
        //printf("%d enter\n", tID);
}

void LeaveCriticalRegion(int tID)
{
        //printf("%d leave\n", tID);
        readyFlags[tID] = false;
}


void Counter(int count, int tID)
{
  for(int i=0; i < count; i++)
  {
                EnterCriticalRegion(tID);
    Count++;
                LeaveCriticalRegion(tID);
                Sleep(10);
  }
}

int sCounter = 0;
void Sum(void* pParams)
{
        int *c = (int *)pParams;
        for(int i=0; i < MAX; i++)
  {
                EnterCriticalSection(&cs);
    sCounter++;
                LeaveCriticalSection(&cs);
                Sleep(10);
  }
}

int main()
{
```

```
        std::thread th[N];
  for(int i = 0; i < N; i++)
  {
    th[i] = std::thread(Counter, MAX, i);
  }


        auto start = chrono::system_clock::now();

  for(int i = 0; i < N; i++)
  {
    th[i].join();
  }

        chrono::duration<double> duration = chrono::system_clock::now() - start;

        printf("Peterson Result: %d\n", Count);
        printf("Peterson Time: %lf\n\n", duration.count());

        InitializeCriticalSection(&cs);
        HANDLE hThreads_[N];

        start = chrono::system_clock::now();
        for(DWORD i = 0; i < N; i++)
    hThreads_[i] = (HANDLE)_beginthread(Sum, 0, (void *) (MAX));

        WaitForMultipleObjects(N, hThreads_, TRUE, INFINITE);
        duration = chrono::system_clock::now() - start;
        printf("WinAPI Result: %d\n", sCounter);
        printf("WinAPI Time (cs): %lf\n\n", duration.count());
        DeleteCriticalSection(&cs);
        system("pause");
  return 0;
}
```