Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа искусственного интеллекта**

**Лабораторная работа 3**

**Синхронизация доступа к одноэлементному буферу**

по дисциплине «Параллельное программирование»

Выполнил

студент гр.3530203/80102 Л.Д. Челищева

<*подпись*>

Руководитель К.А. Туральчук

<*подпись*>

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

# Общие условия

Система содержит 4 физических и 4 же логических ядра.

N – Количество сообщений у одного писателя.

M – Длина сообщения.

W – Число писателей.

R – Число читателей.

В таблицах – время в мс.

# Работа без средств синхронизации

Зависимость от числа потоков (20 сообщений всего по 100 символов):

|  |  |  |  |
| --- | --- | --- | --- |
| Читателей/писателей/сообщений на одного писателя | 2/2/10 | 5/5/4 | 10/10/2 |
| Время | 41,63712 | 545,62125 | 3377,20751 |
| Пропущено | 4,9 | 4,3 | 1,5 |
| Дублировано | 0 | 0 | 0 |

Зависимость от числа сообщений у писателя (читателей 5 и писателей 5, длина сообщения 100 символов):

|  |  |  |  |
| --- | --- | --- | --- |
| N/всего | 10/50 | 100/500 | 1000/5000 |
| Время | 478,63976 | 499,23213 | 451,36066 |
| Пропущено | 6,7 | 117,8 | 1374,4 |
| Дублировано | 0 | 0 | 0 |

Зависимость от длины сообщения у писателя (читателей 5 и писателей 5, число сообщений 100 на писателя, 500 всего):

|  |  |  |  |
| --- | --- | --- | --- |
| M | 10 | 100 | 1000 |
| Время | 556,73428 | 479,89271 | 506,92622 |
| Пропущено | 86,4 | 99,1 | 99 |
| Дублировано | 0 | 0 | 0 |

Длина сообщения оказала наименьшее влияние и в дальнейшем не будет изменяться при тестах:

# lock с одной проверкой

Зависимость времени от числа потоков и количества сообщений (длина сообщения 10).

|  |  |  |  |
| --- | --- | --- | --- |
| W/R N | 100 | 10 000 | 1 000 000 |
| 2/2 | 51,40285 | 85,77855 | 350,907 |
| 4/4 | 358,61715 | 398,2281 | 1075,91725 |

# lock с двойной проверкой (вариант v2=3)

Зависимость времени от числа потоков и количества сообщений (длина сообщения 10).

|  |  |  |  |
| --- | --- | --- | --- |
| W/R N | 100 | 10 000 | 1 000 000 |
| 2/2 | 40,1234 | 25,4847 | 450,2378 |
| 4/4 | 278,04445 | 224,0216 | 1252,20315 |

# AutoResetEvent

Зависимость времени от числа потоков и количества сообщений (длина сообщения 10).

|  |  |  |  |
| --- | --- | --- | --- |
| W/R N | 100 | 10 000 | 100 000 |
| 2/2 | 50,67005 | 203,90425 | 1685,44475 |
| 4/4 | 118,11305 | 436,87785 | 3464,7745 |

# Interlocked

Зависимость времени от числа потоков и количества сообщений (длина сообщения 10).

|  |  |  |  |
| --- | --- | --- | --- |
| W/R N | 100 | 10 000 | 1 000 000 |
| 2/2 | 453,1 | 632,49865 | 1435,00785 |
| 4/4 | 1538,0186 | 829,50395 | 3539,67695 |

# Semaphore (вариант v1=0)

Зависимость времени от числа потоков и количества сообщений (длина сообщения 10).

|  |  |  |  |
| --- | --- | --- | --- |
| W/R N | 100 | 10 000 | 100 000 |
| 2/2 | 60,66475 | 200,4509 | 1730,5995 |
| 4/4 | 96,5283 | 365,65135 | 3542,50755 |

# Результаты

На основе результатов измерений можно заметить очевидное преимущество секции lock. При этом на небольших значениях lock с двойной проверкой имеет преимущество перед одинарной, однако при увеличении числа сообщений это преимущество исчезает.

AutoResetEvent и Semaphore показали себя неэффективными в данной задаче.

Критическая секция, реализованная через Interlocked выглядит неэффективной при небольших значениях, однако очевидно мало зависит от количества этих значений и вероятно имеет повышенную эффективность при большем числе значений.

# Полный код программы

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Threading;

namespace lab3

{

class Program

{

static int N = 10; //Количество сообщений

static int M = 100; //Длина сообщения

static int W = 3; //Число писателей

static int R = 2; //Число читателей

static string[,] WriteMessages = new string[W, N]; //Массив сообщений писателей

static List<string>[] ReadMessages = new List<string>[R]; //Списки сообщений читателей

static bool finish; //Маркер конца

static bool bEmpty; //Маркер пустого буфера

static string buffer; //Буфер

static AutoResetEvent aFull;

static AutoResetEvent aEmpty;

static Semaphore sFull;

static Semaphore sEmpty;

static int fFull;

static int fEmpty;

static void Main(string[] args)

{

char C1 = 'L';

char C2 = 'C';

int V1 = C1 % 4;

int V2 = C2 % 4;

Console.WriteLine("Вариант: {0}, {1}", V1, V2);

Stopwatch stopWatch = new Stopwatch();

double tsum = 0;

double dubsum = 0;

double lostsum = 0;

int[] result;

Thread[] WriteThreads;

Thread[] ReadThreads;

CreateMessages();

for (int t = 0; t < 11; t++)

{

WriteThreads = new Thread[W];

ReadThreads = new Thread[R];

finish = false;

//bEmpty = true;

//aFull = new AutoResetEvent(false);

//aEmpty = new AutoResetEvent(true);

//sFull = new Semaphore(0, 1);

//sEmpty = new Semaphore(1, 1);

fFull = 0;

fEmpty = 1;

stopWatch.Start();

for (int i = 0; i < W; i++)

WriteThreads[i] = new Thread(Write3);

for (int i = 0; i < R; i++)

ReadThreads[i] = new Thread(Read3);

for (int i = 0; i < W; i++)

WriteThreads[i].Start(i);

for (int i = 0; i < R; i++)

ReadThreads[i].Start(i);

for (int i = 0; i < W; i++)

WriteThreads[i].Join();

finish = true;

//aFull.Set();

//sFull.Release();

for (int i = 0; i < R; i++)

ReadThreads[i].Join();

stopWatch.Stop();

result = CheckMessages();

if (t > 0) //Пропустить первый тест

{

lostsum += result[0];

dubsum += result[1];

tsum += stopWatch.Elapsed.TotalMilliseconds;

}

stopWatch.Reset();

ReadMessages = new List<string>[R];

}

Console.WriteLine("10 повторов: {0} писателей, {1} сообщений длиной {2} символов на каждого, {3} читателей:\nВремя выполнения {4} мс, Потеряно сообщений: {5}, Дублировано сообщений: {6}", W, N, M, R, tsum/10, lostsum/10, dubsum/10);

Console.ReadKey();

}

public static void CreateMessages()

{

Random random = new Random();

string word;

for (int i = 0; i < W; i++)

{

for (int j = 0; j < N; j++)

{

word = "";

for (int k = 0; k < M; k++)

word += (char)random.Next(65, 90);

WriteMessages[i,j] = word;

}

}

}

public static int[] CheckMessages()

{

int[] check = new int[2]; //0 - пропущено, 1 - дублировано.

bool check1 = false;

int check2 = 0;

for (int i = 0; i < W; i++)

{

for (int j = 0; j < N; j++)

{

for (int k = 0; k < R; k++)

{

if(ReadMessages[k].Contains(WriteMessages[i, j]))

{

check1 = true;

check2++;

}

}

if (!check1)

check[0]++;

if (check2 > 1)

check[1]++;

check1 = false;

check2 = 0;

}

}

return check;

}

//Обычный, lock, lock с двойной проверкой

public static void Read1(object o)

{

int num = (int)o;

ReadMessages[num] = new List<string>();

while (!finish)

{

// if (!bEmpty)

// {

// lock ("Read")

// {

if (!bEmpty)

{

ReadMessages[num].Add(buffer);

bEmpty = true;

}

// }

// }

}

}

public static void Write1(object o)

{

int num = (int)o;

int i = 0;

while (i < N)

{

// if (bEmpty)

// {

// lock ("Write")

// {

if (bEmpty)

{

buffer = WriteMessages[num, i];

i++;

bEmpty = false;

}

// }

// }

}

}

//AutoResetEvent и Semaphore

public static void Read2(object o)

{

int num = (int)o;

ReadMessages[num] = new List<string>();

while (!finish)

{

//aFull.WaitOne();

sFull.WaitOne();

if (finish)

{

//aFull.Set();

sFull.Release();

break;

}

ReadMessages[num].Add(buffer);

//aEmpty.Set();

sEmpty.Release();

}

}

public static void Write2(object o)

{

int num = (int)o;

int i = 0;

while (i < N)

{

//aEmpty.WaitOne();

sEmpty.WaitOne();

buffer = WriteMessages[num, i];

i++;

//aFull.Set();

sFull.Release();

}

}

//Interlocked.CompareExchange (что, с чем сравнить, на что заменить) возвр что

public static void Read3(object o)

{

int num = (int)o;

ReadMessages[num] = new List<string>();

while (!finish)

{

if (Interlocked.CompareExchange(ref fFull, 0, 1) == 1)

{

ReadMessages[num].Add(buffer);

fEmpty = 1;

}

}

}

public static void Write3(object o)

{

int num = (int)o;

int i = 0;

while (i < N)

{

if (Interlocked.CompareExchange(ref fEmpty, 0, 1) == 1)

{

buffer = WriteMessages[num, i];

i++;

fFull = 1;

}

}

}

}

}

# Выводы

В ходе лабораторной работы были рассмотрены методы синхронизации доступа, в частности было уделено внимание синхронизации доступа к одноэлементному буферу через: критическую секцию lock, AutoResetEvent, Semaphore, атомарные операторы Interlocked.

Так же в ходе работы была рассмотрена и проиллюстрирована проблема гонки данных.