Отчет по выполнению  
Лабораторная работа №5  
«Синхронизация потоков»

Выполнил:  
студент гр. МОиАИС 184-1  
Ибраев Ерлан Иржанович

Цель работа: изучить различные виды синхронизации, параллельных вычислений и проанализировать их эффективность.

Несколько потоков работают с общим одноэлементным буфером.

Потоки делятся на «писателей», осуществляющих запись сообщений в буфер, и «читателей», осуществляющих извлечение сообщений из буфера.

Работу с буфером может осуществлять только один поток. Если буфер свободен, то только один писатель может осуществлять запись в буфер. Если буфер занят, то только один читатель может осуществлять чтение из буфера. После чтения буфер освобождается и доступен для записи.

В качестве буфера используется глобальная переменная, например, типа string. Работа приложения заканчивается после того, как все сообщения писателей через общий буфер будут обра-ботаны читателями

1. Реализуйте взаимодействие потоков-читателей и потоков-писателей с общим буфером без каких-либо средств синхронизации. Проиллюстрируйте проблему совместного доступа. Почему возникает проблема доступа?

2. Реализуйте доступ "читателей" и "писателей" к буферу с применением следующих средств синхронизации:

1. блокировки (lock);
2. сигнальные сообщения (ManualResetEvent, AutoResetEvent, ManualResetEventSlim);
3. семафоры (Semaphore, SemaphoreSlim).

3. Исследуйте производительность средств синхронизации при разном числе сообщений, разном объеме сообщений, разном числе потоков. Сделайте выводы об эффективности применения средств синхронизации.

При работе алгоритма может возникнуть могут возникнуть ряд проблем. Отсутствие средств синхронизации при обращении к буферу приводит к появлению гонки данных – несколько читателей могут прочитать одно и то же сообщение, прежде чем успеют обновить статус буфера; несколько писателей могут одновременно осуществить запись в бузаер, стирая предыдущее сообщение; В данной задаче следствием гонки данных является потеря одних сообщений и дублирование других;

|  |  |  |
| --- | --- | --- |
| Типы синхронизации | N = 100000 writers = 4 readers = 4 | N = 100000 writers = 4 readers = 4 |
| Без синхронизации | 110,7043 | 164,5612 |
| Lock | 141,6207 | 283,1927 |
| ManualResetEvent | 564.486 | 911.5635 |
| AutoResetEvent | 6387.4901 | 13512.6941 |
| ManualResetEventSlim | 97.707 | 160.5694 |
| Semaphore | 57511.4048 | 34743.413 |
| SemaphoreSlim | 644.0791 | 3529.564 |

Вывод: на основе таблицы можно предположить, что ManualReserSlim самая эффективная синхронизация. В случае вызова метода Set () при наличии нескольких потоков, ждущих событий в сигнальном состоянии, ожидание всех этих потоков прекращается. В случае, если поток просто вызывает метод WaitOne (), а событие уже находится в сигнальном состоянии, ожидаемый поток может сразу же продолжить работу.

Код программы:

class Program

{

static bool bEmpty = true;

static bool finish = false;

static object locker = new object();

static ManualResetEvent MREFull;

static ManualResetEvent MREEmpty;

static AutoResetEvent AREFull;

static AutoResetEvent AREEmpty;

static ManualResetEventSlim MRESFull;

static ManualResetEventSlim MRESEmpty;

static Semaphore SEmpty;

static SemaphoreSlim SSEmpty;

static SemaphoreSlim ssEmpty;

static DateTime dt1, dt2;

static int read = 4;

static int write = 4;

static int mes = 100000;

static string buffer;

static Thread[] writers = new Thread[write];

static Thread[] readers = new Thread[read];

static List<string[]> resWrit = new List<string[]>();

static List<List<string>> resRead = new List<List<string>>();

static void Reader()

{

List<string> MyMessagesRead = new List<string>();

while (!finish)

if (!bEmpty)

{

bEmpty = true;

MyMessagesRead.Add(buffer);

}

}

static void ReaderLock()

{

List<string> MyMessagesRead = new List<string>();

while (!finish)

if (!bEmpty)

lock (locker)

if (!bEmpty)

{

bEmpty = true;

MyMessagesRead.Add(buffer);

}

}

static void ReaderManualResetEvent(object state)

{

var RMREFull = ((object[])state)[0] as ManualResetEvent;

var RMREEmpty = ((object[])state)[1] as ManualResetEvent;

List<string> MyMessagesRead = new List<string>();

while (!finish)

{

RMREFull.WaitOne();

if (finish)

{

RMREFull.Set();

break;

}

MyMessagesRead.Add(buffer);

RMREEmpty.Set();

}

}

static void ReaderAutoResetEvent(object state)

{

var RAREFull = ((object[])state)[0] as AutoResetEvent;

var RAREEmpty = ((object[])state)[1] as AutoResetEvent;

List<string> MyMessagesRead = new List<string>();

while (!finish)

{

RAREFull.WaitOne();

if (finish)

{

RAREFull.Set();

break;

}

MyMessagesRead.Add(buffer);

RAREEmpty.Set();

}

}

static void ReaderManualResetEventSlim(object state)

{

var RMRESFull = ((object[])state)[0] as ManualResetEventSlim;

var RMRESEmpty = ((object[])state)[1] as ManualResetEventSlim;

List<string> MyMessagesRead = new List<string>();

while (!finish)

{

RMRESFull.Wait();

if (finish)

{

RMRESFull.Set();

break;

}

MyMessagesRead.Add(buffer);

RMRESEmpty.Set();

}

}

static void ReaderSemaphore(Semaphore state)

{

var SRead = state as Semaphore;

List<string> MyMessagesRead = new List<string>();

while (!finish)

if (!bEmpty)

{

SRead.WaitOne();

if (!bEmpty)

{

bEmpty = true;

MyMessagesRead.Add(buffer);

}

SRead.Release();

}

}

static void ReaderSemaphoreSlim(object state)

{

var SSRead = state as SemaphoreSlim;

List<string> MyMessagesRead = new List<string>();

while (!finish)

if (!bEmpty)

{

SSRead.Wait();

if (!bEmpty)

{

bEmpty = true;

MyMessagesRead.Add(buffer);

}

SSRead.Release();

}

}

static void Writer()

{

string[] MyMessagesWri = new string[mes];

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

MyMessagesWri[j] = j.ToString();

int i = 0;

while (i < mes)

if (bEmpty)

{

buffer = MyMessagesWri[i++];

bEmpty = false;

}

}

static void WriterLock()

{

string[] MyMessagesWri = new string[mes];

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

MyMessagesWri[j] = j.ToString();

int i = 0;

while (i < mes)

lock (locker)

{

if (bEmpty)

{

buffer = MyMessagesWri[i++];

bEmpty = false;

}

}

}

static void WriterManualResetEvent(object state)

{

var WMREFull = ((object[])state)[0] as ManualResetEvent;

var WMREEmpty = ((object[])state)[1] as ManualResetEvent;

string[] MyMessagesWri = new string[mes];

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

MyMessagesWri[j] = j.ToString();

int i = 0;

while (i < mes)

{

WMREEmpty.WaitOne();

buffer = MyMessagesWri[i++];

WMREFull.Set();

}

}

static void WriterAutoResetEvent(object state)

{

var AREFull = ((object[])state)[0] as AutoResetEvent;

var AREEmpty = ((object[])state)[1] as AutoResetEvent;

string[] MyMessagesWri = new string[mes];

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

MyMessagesWri[j] = j.ToString();

int i = 0;

while (i < mes)

{

AREEmpty.WaitOne();

buffer = MyMessagesWri[i++];

AREFull.Set();

}

}

static void WriterManualResetEventSlim(object state)

{

var WMRESFull = ((object[])state)[0] as ManualResetEventSlim;

var WMRESEmpty = ((object[])state)[1] as ManualResetEventSlim;

string[] MyMessagesWri = new string[mes];

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

MyMessagesWri[j] = j.ToString();

int i = 0;

while (i < mes)

{

WMRESEmpty.Wait();

buffer = MyMessagesWri[i++];

WMRESFull.Set();

}

}

static void WriterSemaphore(object state)

{

var SWrite = state as Semaphore;

string[] MyMessagesWri = new string[mes];

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

MyMessagesWri[j] = j.ToString();

int i = 0;

while (i < mes)

if (bEmpty)

{

SWrite.WaitOne();

if (bEmpty)

{

bEmpty = false;

buffer = MyMessagesWri[i++];

}

SWrite.Release();

}

}

static void WriterSemaphoreSlim(object state)

{

var SSWrite = state as SemaphoreSlim;

string[] MyMessagesWri = new string[mes];

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

MyMessagesWri[j] = j.ToString();

int i = 0;

while (i < mes)

if (bEmpty)

{

SSWrite.Wait();

if (bEmpty)

{

bEmpty = false;

buffer = MyMessagesWri[i++];

}

SSWrite.Release();

}

}

static void Main()

{

dt1 = DateTime.Now;

ssEmpty = new SemaphoreSlim(1);

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

{

writers[i] = new Thread(WriterSemaphoreSlim);

writers[i].Start(ssEmpty);

}

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

{

readers[i] = new Thread(ReaderSemaphoreSlim);

readers[i].Start(ssEmpty);

}

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

writers[i].Join();

finish = true;

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

readers[i].Join();

dt2 = DateTime.Now;

Console.WriteLine((dt2 - dt1).TotalMilliseconds);

}

}

}

Скриншоты:

Без синхронизации  


Lock  


ManualResetEvent  
  


AutoResetEvent  
  


ManualResetEventSlim  
  


Semaphore  
  


SemaphoreSlim  
  
