Лабораторна робота №3

Тема: Критичні секції.

Мета роботи: навчитися використовувати і програмувати критичні секції.

Порядок виконання.

Критичні секції - це об'єкти, що використовуються для блокування доступу всіх потоків (threads) додатка, крім однієї, до деяких важливих даних в один момент часу. Наприклад, є змінна m_pObject і кілька ниток, що викликають методи об'єкта, на який посилається m_pObject, причому ця змінна може змінювати своє значення час від часу. Іноді там навіть виявляється нуль. Припустимо, є ось такий код:

```
// Потік № 1
void Proc1 ()
(
if (m_pObject)
  m_pObject-> SomeMethod ();
)
// Потік № 2
void Proc2 (IObject * pNewObject)
(
if (m_pObject)
delete m_pObject;
m_pObject = pNewobject;
)
```

Тут ми маємо потенційну небезпеку виклику m_pObject-> SomeMethod () після того, як об'єкт був знищений за допомогою delete m_pObject. Справа в тому, що в системах з витісняючої багатозадачністю виконання будь-якої нитки процесу може перерватися в самий невідповідний для неї момент часу, і почне виконуватися зовсім інша Потік. У даному прикладі невідповідним моментом буде той, у якому Потік № 1 вже перевірила m_pObject, але ще не встигла викликати SomeMethod (). Виконання нитки № 1 урвалася, і почала виконуватися нитку № 2. Причому Потік № 2 встигла викликати деструктор об'єкта. Що ж станеться, коли Потік № 1 отримає трохи процесорного часу і викличе-таки SomeMethod () у вже неіснуючого об'єкта? Напевне, щось жахливе.

Саме тут приходять на допомогу критичні секції. Перепишемо наш приклад.

```
// Потік № 1
void Proc1 ()
(
  :: EnterCriticalSection (& m_lockObject);
if (m_pObject)
```

```
  m_pObject-> SomeMethod ();
  :: LeaveCriticalSection (& m_lockObject);
)
// Потік № 2
void Proc2 (IObject * pNewObject)
(
  :: EnterCriticalSection (& m_lockObject);
if (m_pObject)
delete m_pObject;
m_pObject = pNewobject;
:: LeaveCriticalSection (& m_lockObject);
)
```

Код, поміщений між:: EnterCriticalSection () і :: LeaveCriticalSection () з однієї і тієї ж критичною секцією як параметра, ніколи не буде виконуватися паралельно. Це означає, що якщо Потік № 1 встигла "захопити" критичну секцію m_lockObject, то при спробі нитки № 2 дістати цю ж критичну секцію у своє одноосібне користування, її виконання буде припинено до тих пір, поки Потік № 1 не "відпустить" m_lockObject за допомогою виклику:: LeaveCriticalSection (). І навпаки, якщо Потік № 2 встигла раніше нитки № 1, то та "зачекає", перш ніж почне роботу з m_pObject.

Робота з критичними секціями

Що ж відбувається всередині критичних секцій і як вони влаштовані? Перш за все, слід відзначити, що критичні секції - це не об'єкти ядра операційної системи. Практично вся робота з критичними секціями відбувається в Який створив їх процесі. З цього випливає, що критичні секції можуть бути використані тільки для синхронізації в межах одного процесу. Тепер розглянемо критичні секції ближче.

Структура RTL_CRITICAL_SECTION

```
typedef struct _RTL_CRITICAL_SECTION (
PRTL_CRITICAL_SECTION_DEBUG DebugInfo;// Використовується операційною системою
LONG LockCount;// Счетчик використання цієї критичної секції
LONG RecursionCount;// Счетчик повторного захоплення з нитки-власника
HANDLE OwningThread;// Унікальний ID нитки-власника
HANDLE LockSemaphore;// Об'єкт ядра використовується для очікування
ULONG_PTR SpinCount;// Кількість холостих циклів перед викликом ядра
) RTL_CRITICAL_SECTION, * PRTL_CRITICAL_SECTION;
```

Поле LockCount збільшується на одиницю при кожному виклик:: EnterCriticalSection () і зменшується при кожному виклику :: LeaveCriticalSection (). Це перший (а часто і єдина перевірка) на шляху до "захоплення" критичної секції. Якщо після збільшення в цьому полі знаходиться нуль, це означає, що до цього моменту непарних викликів:: EnterCriticalSection () з інших ниток не було. У цьому випадку можна забрати дані, що охороняються цією критичною секцією у монопольне

користування. Таким чином, якщо критична секція інтенсивно використовується не більш ніж однієї ниткою, :: EnterCriticalSection () практично вироджується в + + LockCount, а :: LeaveCriticalSection () в - LockCount. Це дуже важливо. Це означає, що використання багатьох тисяч критичних секцій в одному процесі не спричинить значної витрати ні системних ресурсів, ні процесорного часу.

У полі RecursionCount зберігається кількість повторних викликів:: EnterCriticalSection () з однієї і тієї ж нитки. Дійсно, якщо спричинити:: EnterCriticalSection () з однієї і тієї ж нитки кілька разів, все виклики будуть успішні. Тобто ось такий код не зупиниться навічно в другому виклик:: EnterCriticalSection (), а відпрацює до кінця.

```
// Потік № 1
void Proc1 ()
(
  :: EnterCriticalSection (& m_lock);
//. ..
Proc2 ()
//. ..
  :: LeaveCriticalSection (& m_lock);
)
// Все ще Потік № 1
void Proc2 ()
(
  :: EnterCriticalSection (& m_lock);
//. ..
:: LeaveCriticalSection (& m_lock);
)
```

Дійсно, критичні секції призначені для захисту даних від доступу з декількох ниток. Багаторазове використання однієї і тієї ж критичної секції з однієї нитки не призведе до помилки. Це цілком нормальне явище. Слідкуйте, щоб кількість викликів:: EnterCriticalSection () і:: LeaveCriticalSection () збігалося, і все буде добре.

Поле OwningThread містить 0 для ніким не зайнятих критичних секцій або унікальний ідентифікатор нитки-власника. Це поле перевіряється, якщо при виклику:: EnterCriticalSection () поле LockCount після збільшення на одиницю виявилося більше нуля. Якщо OwningThread збігається з унікальним ідентифікатором поточної нитки, то RecursionCount просто збільшується на одиницю і:: EnterCriticalSection () повертається негайно. Інакше :: EnterCriticalSection () буде чекати, поки Потік, що володіє критичної секцією, не викличе:: LeaveCriticalSection () необхідну кількість разів.

Команди роботи з критичними секціями:

InitializeCriticalSection – ініціалізація об’єкту типу *CRITICAL_SECTION*, після ініціалізації нашого об’єкту ми в кожному із паралельних потоків перед входом в критичну секцію визиваємо функцію *EnterCriticalSection*, яка виключає одночасний вхід в критичні секції. Після завершення роботи з роздільним ресурсом потік повинен залишити свою критичну секцію, що виконується виловом функції *LeaveCriticalSection.* Після закінчення з об’єктами типу *CRITICAL_SECTION* необхідно звільнити всі системні ресурси, котрі використовувались цим об’єктом. Для цього використовуєтсья функція *DeleteCriticalSection.*

Тепер розглянемо роботу цих функцій. Для цього спочатку виконаємо приклад в якому виконуються несинхронізовані паралельні потоки, а потім синхронізуємо їх роботу, використовуючи критичні секції.

```
#include <windows.h>
#include <iostream.h>
#include <math.h>
#include <conio.h>


DWORD WINAPI thread(LVOID)
 {
   float y;
   int i,j;

   for (j=0; j<10; j++)
    {

      for (i=0; i<10; i++)
       {
         cout<< "i"<<j << ' '<<flush;
         Sleep(7);
       }
      cout << endl;
       }
    return 0;
 }
 int main()
  {
    int i,j;
    float x;
    HANDLE hThread;
    DWORD IDThread;
    hThread=CreateThread(NULL, 0, thread, NULL, 0, &IDThread);

    if (hThread ==NULL)
      return GetLastError();

    for (j=0; j<10; j++)
     {

       for (i=0; i<10; i++)
       {
           cout<< "j" <<j << ' ' << flush;
           Sleep(7);
       }
     cout << endl;
       }
     getch();
    WaitForSingleObject(hThread, INFINITE);

    return 0;
  }
```

В програмі кожний із потоків main і thread виводить рядки однакових чисел. Але через паралельність роботи потоків кожний виведений рядок може містити не рівні між

собою елементи. Наша задача буде полягати в наступному: потрібно так синхронізувати потоки, щоб в кожній стрічці виводились тільки рівні між собою числа.

Нижче наведено програмну реалізації цієї задачі з використанням об’єкту типу *CRITICAL_SECTION:*

Підключення бібліотек:

```
#include <windows.h>
#include <iostream.h>
#include <math.h>
#include <conio.h>
```

Код програми:

```
CRITICAL_SECTION cs;

DWORD WINAPI thread(LVOID)
 {
   float y;
   int i,j;

   for (j=0; j<10; j++)
    {
      EnterCriticalSection (&cs);
      for (i=0; i<10; i++)
       {
         cout<< "i"<<j << ' '<<flush;
         Sleep(7);
       }
      cout << endl;

      LeaveCriticalSection(&cs);
    }
    return 0;
 }
 int main()
  {
    int i,j;
    float x;
    HANDLE hThread;
    DWORD IDThread;

    InitializeCriticalSection(&cs);
    hThread=CreateThread(NULL, 0, thread, NULL, 0, &IDThread);

    if (hThread ==NULL)
      return GetLastError();

    for (j=0; j<10; j++)
     {
       EnterCriticalSection(&cs);
       for (i=0; i<10; i++)
       {
           cout<< "j" <<j << ' ' << flush;
           Sleep(7);
       }
     cout << endl;

     LeaveCriticalSection(&cs);
     }
     getch();
    WaitForSingleObject(hThread, INFINITE);
    DeleteCriticalSection(&cs);

    return 0;
  }
```

Завдання

Задано дійсне число **x=2.7*№** (де № – номер варіанту) і цілі числа **m=10** і **n=5**. Відповідно до номера варіанту потрібно розробити програму для обчислення значень елементів двох матриць та виконати їх сортування по зростанню. Обчислення матриць **A та B** виконати в різних потоках (відповідно до наведеного зразку)

A = [**Ai** = [**aij** = **f[x,i, j], j = 1,n], i = 1,m**] і **B** = [**Bi** = [**bij** = **f[x,i, j], j = 1,n], i = 1,m**]

Матриці обчислюються за формулами:

1) $a_{ij} = \frac{i + j^2}{4{,}5x - j} \sin^2 ij;$ $\quad b_{ij} = \frac{i + j}{x + 3} \cdot tg^3 j / i.$

2) $a_{ij} = \sqrt[3]{\frac{i + j}{0.2x}} \cdot \cos i;$ $\quad b_{ij} = \frac{i^3 + j}{x + j} \cdot \sin(i / j)^2$

3) $a_{ij} = \sin^2 \frac{i^3 + j^{1.3}}{x + j};$ $\quad b_{ij} = \frac{i - x}{i^2 + j} \cos(j / i)^2$

4) $a_{ij} = i^3 \cdot \frac{\sin j}{e^{0.2i - j}};$ $\quad b_{ij} = \frac{e^{i+x}}{i + j} \cdot \cos^2(x \cdot j).$

5) $a_{ij} = \frac{i - j}{i + j^2 / 2};$ $\quad b_{ij} = \frac{i^2 - \cos j}{x + 1{,}5 + ij}.$

6) $a_{ij} = i^2 + j^3 \frac{\cos(i + j)}{e^{-(i+2)}};$ $\quad b_{ij} = \frac{arctg(i \cdot j)}{\lg(i / j + x)}.$

7) $a_{ij} = \frac{e^i}{x} \cdot \frac{i + \sin j}{2{,}5 + i \cdot j};$ $\quad b_{ij} = \frac{i^3 + j - x}{2i + \cos(x - j)}.$

8) $a_{ij} = \frac{x(i - j^{2.3})}{\cos(i + j) - 2{,}5i};$ $\quad b_{ij} = \frac{i \cdot \cos j + x \cdot \sin i}{x + \lg(i + j^2)}.$

9) $a_{ij} = x^3 \frac{\cos(i + j^2)}{\sin^2(i^2 + j)} - j^2;$ $\quad b_{ij} = \frac{e^{\cos i / x}}{\sin x / j} \cdot tg(i + j).$

10) $a_{ij} = \frac{2i + j}{x5{,}5 - 2j} \cos^2 ij;$ $\quad b_{ij} = \frac{i^2 - x}{j^2 + x} \cdot tg(j + 1).$

11) $a_{ij} = \sqrt[3]{\frac{i + 3j}{x + j}} \cdot \sin i;$ $\quad b_{ij} = \frac{i + j}{j + x} \cdot \cos(2j + i)$