ЛЕКЦІЯ 8

Потоки в операційній системі Windows.

Загальні поняття пов’язані з потоками.

Засоби синхронізацій потоків.

Локальна пам’ять потоків.

Засоби роботи з потоками в Visual C++ та функції Windows API.

Операційні системи

доц. Сінельнікова Т.Ф.

ЗАГАЛЬНІ ВІДОМОСТІ ПРО ПОТОКИ

При створенні процесу в системі з'являється новий програмний потік, що належить цьому процесу. На початку будь-який новостворений процес має лише один потік. Цей потік може створювати нові потоки, а ці нові потоки, в свою чергу, можуть створювати інші нові потоки. Процес продовжує своє існування до тих пір, поки в його володінні знаходиться принаймні один програмний потік.

Реалізація багатопоточності грунтується на наданні кожному потоку певного часу для роботи, тобто кожен потік отримує в своє розпорядження процесор на деякий час. Такий підхід призводить до того, що багатопотоковий додаток може працювати повільніше, ніж однопоточний, але у той же час використання потоків може бути мало не єдиним рішенням у багатьох завданнях. У першу чергу потоки знаходять застосування при розробці комунікаційних програм і програм для різних обчислень.

ЗАГАЛЬНІ ВІДОМОСТІ ПРО ПОТОКИ

Будь-який потік визначає послідовність виконання коду в процесі і складається з двох компонентів:

- об'єкт ядра, через який операційна система управляє потоком, і в якому міститься інформація про потік.
- стек потоку, який містить параметри всіх функцій і локальні змінні, необхідні потоку для виконання коду.

ПРИНЦИПИ ФУНКЦІОНУВАННЯ ПОТОКІВ

1) Будь-який потік належить якомусь процесу, який нічого не виконує, а є контейнером (адресним простором) для потоків.

2) Потоки завжди створюються в контексті будь-якого процесу, і все їхнє життя проходить в його межах (адресному просторі). Два і більше потоку, створені в контексті одного процесу розділяють один адресний простір.

3) Потоки можуть виконувати один і той же код і маніпулювати одними і тими ж даними, а також спільно використовувати дескриптори об'єктів ядра, оскільки таблиця описувачів належить процесу, а не потоку.

4) Потоку потрібен об'єкт ядра і стек, обсяг статичних відомостей про потік невеликий і займає небагато пам'яті, на відміну від процесу, який вимагає значних системних ресурсів (в адресний віртуальний простір процесу завантажуються DLL і EXE файли).

ЛОКАЛЬНА ПАМ’ЯТЬ ПОТОКІВ

Деяка область пам'яті, яка відноситься до конкретного потоку, але не є локальною змінної називається **локальною пам'яттю потоку** (Thread Local Storage, TLS).

Робота локальної пам'яті потоків може бути розглянута на такому прикладі. Припустимо, що є таблиця з двома колонками, таким чином, кожен запис має два поля: у першому полі міститься ідентифікатор ID-потоку, а в другому - 32-бітове число. Кожен раз при зверненні потоку до функції в цій функції визначається ID потока, що звернувся за допомогою функції GetCurrentThreadID і здійснюється його пошук в таблиці. Якщо запис з таким ідентифікатором є, то функція використовує цей запис, в іншому випадку в таблицю вноситься новий запис, що містить ID поточного потоку. Таким чином, виходить таблиця, що належить поточному потоку, у другому полі цього запису можна зберігати лічильник звернення даним потоком до функції.

ЛОКАЛЬНА ПАМ’ЯТЬ ПОТОКІВ

Для кожного з процесів створюється набір внутрішніх таблиць. Windows може створити для кожного процесу до 64 таких таблиць, таким чином можна використовувати 64 різні змінні TLS. При зверненні до TlsAlloc Windows виділяє одну таблицю TLS. Після цього можна використовувати функції TlsGetValue і TlsSetValue для того, щоб встановити або прочитати значення з таблиці. При цьому операційна система звертається саме до того запису в таблиці, який відповідає поточному потоку. Якщо таблиця більше не потрібна, то можна звернутися до функції TlsFree для того, щоб звільнити її. У кожному запису таблиці можна зберігати будь-яке 32-бітове число, однак, найчастіше таблиця TLS використовується для зберігання покажчиків на клас або структуру, що містить всі необхідні для потоку змінні.

СИНХРОНІЗАЦІЯ ПОТОКІВ

З реалізацією багатопоточності тісно пов'язане поняття **синхронізація** - управління спільним доступом різних потоків до одних і тих же ресурсів (дані, процесор, екранні форми, модеми, принтери і т.д.). Цей процес дуже складний і в ОС Windows існують різні способи синхронізації, які будуть розглянуті нижче. Всі потоки в системі зазвичай мають доступ до її ресурсів - файлів, блоків пам'яті, послідовних портів і т.д. Будь-який потік, запитуючи потрібний ресурс, повинен отримати монопольний доступ до ресурсу, що і є одним із завдань синхронізації потоків.

СИНХРОНІЗАЦІЯ ПОТОКІВ

Таким чином, основними завданнями синхронізації потоків є:

- монопольний доступ до спільно розділяємого ресурсу, інакше потоки можуть зруйнувати його;
- повідомлення деяких потоків про завершення будь-яких операцій в інших потоках.

Якщо не використовувати ніяких методів синхронізації, то можна зіткнутися з ситуацією гонок (race condition) і тупиків. Обидві ці ситуації приведуть до зупинки роботи багатопотокового застосування.

СИТУАЦІЇ ГОНОК ТА ТУПИКІВ

Ситуація гонок виникає, коли два або більше потоку намагаються отримати доступ до загального ресурсу і змінити його стан. Припустимо Потік 1 отримав доступ до ресурсу і змінив його згідно сценарію своєї роботи. Після цього був запущений Потік 2 і він також модифікував цей же ресурс до завершення Потоку 1. Потік 1 вважає, що ресурс залишився без змін, проте він уже змінений. В залежності від того, коли саме був змінений ресурс і який це був ресурс, результати можуть бути різними: або взагалі нічого не відбудеться, або додаток перестане працювати, або якісь дані будуть втрачені.

СИТУАЦІЇ ГОНОК ТА ТУПИКІВ

Тупики мають місце тоді, коли потік чекає ресурс, який в даний момент належить іншому потоку. Припустимо Потік 1 отримує доступ до об'єкта А і, для того щоб продовжувати роботу, чекає можливості отримання доступу до об'єкта Б. У той же час Потік 2 отримує доступ до об'єкту Б і чекає можливості отримати доступ до об'єкта А. Дана ситуація призводить до блокування обох потоків. Тепер ні Потік 1, ні Потік 2 не будуть виконуватися. Виникнення ситуацій гонок і тупиків можна уникнути, якщо використовувати методи синхронізації, що розглядаються нижче.

СПОСОБИ СИНХРОНІЗАЦІЇ ПОТОКІВ ЗАСОБАМИ ОС WINDOWS

Атомарний доступ (atomic access) - монопольне захоплення ресурсів потоком, що звертається до них, - це є основа основ синхронізації потоків. Interlocked функції - найпростіший спосіб синхронізації потоків за рахунок атомарного доступу до змінної, ці функції гарантують монопольний доступ до змінної типу LONG незалежно від того, який саме код генерується компілятором і скільки процесорів має комп'ютер. Важливою особливістю є те, що це найшвидший спосіб синхронізації потоків з усіх існуючих. Виклик однієї такої функції вимагає не більше 50 тактів процесора, а при використанні синхронізуючих об'єктів ядра потрібен перехід в режим ядра, що забирає не менше 1000 тактів процесора і вихід з режиму ядра.

СПОСОБИ СИНХРОНІЗАЦІЇ ПОТОКІВ ЗАСОБАМИ ОС WINDOWS

Спін-блокування (spin-lock). Це спосіб використання функції InterlockedExchange для монопольного доступу до ресурсу. Для вирішення цього завдання вводиться змінна, яка використовується як прапор, що показує стан ресурсу, (вільний / зайнятий). Якщо ресурс зайнятий, то цикл буде працювати і завантажувати процесор до тих пір, поки ресурс не звільниться. Спін-блокування зазвичай використовується, коли захищається ресурс зайнятий недовго, а також для реалізації критичних секцій (critical section). Даний метод розрахований на однаковий пріоритет потоків, які працюють з ресурсом, і динамічна зміна пріоритетів потоків повинно бути відключена. При використанні спін-блокування не можна допустити потрапляння прапора зайнятості ресурсу і даних, що захищаються в одну кеш-лінію, це може викликати колосальне зниження продуктивності на багатопроцесорних машинах.

СПОСОБИ СИНХРОНІЗАЦІЇ ПОТОКІВ У РЕЖИМІ КОРИСТУВАЧА ЗАСОБАМИ ОС WINDOWS

Кеш-лінії процесора (CPU cache lines).

Критичні секції (critical section) - це частини вихідного коду, які не можуть виконуватися двома потоками в один і той же час. Використовуючи критичну секцію, можна впорядкувати виконання конкретних частин вихідного коду. Критичні секції можуть використовуватися тільки в межах одного процесу, одного додатку. Система може витіснити потік і передати процесорний час іншому потоку, але до захищеного ресурсу жоден потік доступу не отримає, поки даний потік не вийде з критичної секції. Критичні секції реалізовані на основі Interlocked-функцій і виконуються дуже швидко, але їх недолік полягає в тому, що вони дають можливості синхронізувати потоки тільки в рамках одного процесу. Для кожного ресурсу, використовується окремо виділений примірник структури CRITICAL_SECTION, який зазвичай є глобальним, і з яким оперують спеціальні функції входу і виходу в / з критичної секції. Перед використанням структури вона ініціалізується функцією InitializeCriticalSection.

ПРИКЛАД: ВИКОРИСТОВУВАННЯ КРИТИЧНИХ СЕКЦІЙ

```
int     iMyRes1;
char    cMyRes2;
CRITICAL_SECTION csResource1;
CRITICAL_SECTION csResource2;
DWORD WINAPI ThreadFunc(PVOID pvParam) {
    EnterCriticalSection(&csResource1);
// доступ к ресурсу iMyRes1
    LeaveCriticalSection(&csResource1);
 // -------------выполняется код потока----------------
EnterCriticalSection(&csResource2);
// доступ к ресурсу cMyRes2
    LeaveCriticalSection(&csResource2);
// -------------выполняется код потока----------------
    EnterCriticalSection(&csResource1);
    EnterCriticalSection(&csResource2);
// одновременный доступ к нескольким ресурсам (iMyRes1 и
cResource2)     LeaveCriticalSection(&csResource2);
    LeaveCriticalSection(&csResource1);
}
```

СИНХРОНІЗАЦІЯ ПОТОКІВ ОБ’ЄКТАМИ ЯДРА

Механізми синхронізації в режимі користувача (розглянуті вище) забезпечують високу швидкодію, але ці механізми мають багато обмежень, і в більшості програм їх недостатньо. Механізми синхронізації об'єктами ядра надають більшу функціональність, єдиний їхній недолік - це менше швидкодія.

До об'єктів ядра, які можна використовувати для синхронізації відносяться:

- **Мьютекси;**
- **Семафори;**
- **Події;**

ВИНИКНЕННЯ СИТУАЦІЇ ВЗАЄМНОГО БЛОКУВАННЯ (DEAD LOCK) ВНАСЛІДОК НЕДОТРИМАННЯ ПОРЯДКУ ВХОДЖЕННЯ В КРИТИЧНІ СЕКЦІЇ

```
DWORD WINAPI ThreadFunc1(PVOID pvParam)
{
    EnterCriticalSection(&csResource1);
    // доступ до ресурсу iMyRes1
    EnterCriticalSection(&csResource2);
    // доступ до ресурсу cMyRes2
    // одночасний доступ до декількох ресурсів (iMyRes1 и
cMyRes2)
    LeaveCriticalSection(&csResource2);
    LeaveCriticalSection(&csResource1);
}

DWORD WINAPI ThreadFunc2(PVOID pvParam)
{
EnterCriticalSection(&csResource2);
// доступ до ресурсу cMyRes2
EnterCriticalSection(&csResource1);
// доступ до ресурсу iMyRes1
// одночасний доступ до декількох ресурсів (iMyRes1 и cMyRes2)
LeaveCriticalSection(&csResource2);
LeaveCriticalSection(&csResource1);
}
```

МЬЮТЕКСИ(MUTEX)

Мьютекс (mutex) - це глобальні об'єкти, за допомогою яких можливо організувати послідовний доступ до ресурсів. Спочатку встановлюється мьютекс, потім відбувається деяка дія і після цього звільняється мьютекс. Якщо мьютекс встановлено, будь-який інший потік (або процес), який спробує встановити той же мьютекс, буде зупинено до того моменту, коли мьютекс буде звільнений попереднім потоком (або процесом). Мьютекс може спільно використовуватися різними додатками.

Мьютекс застосовуються для синхронізації доступу до ресурсу декількох потоків з різних процесів, - при цьому можна задавати максимальний час очікування доступу до ресурсу. Взагалі, мьютекс поводяться подібно критичним секціях, і якщо є можливість замінити їх критичними секціями, то рекомендується це зробити, тому що останні мають значно більш високо швидкодію. Створити мьютекс можна функцією **CreateMutex**, а відкрити його функцією - **OpenMutex**.

МЬЮТЕКСИ(MUTEX)

При роботі з мьютексами необхідно слідувати наступним правилам:

- Якщо його ідентифікатор потоку дорівнює 0 (потік не може мати такий ідентифікатор), мьютекс не захоплен жодним з потоків і знаходиться у вільному стані.
- Якщо його ідентифікатор не дорівнює 0, мьютекс захоплений одним з потоків і знаходиться в зайнятому стані.
- Захоплення мьютекса здійснюється Wait-функцією, при цьому значення ідентифікатору потоку приймає значення потоку, що його захопив.
- Звільнити мьютекс можна функцією **ReleaseMutex**.
- На відміну від інших об'єктів ядра, мьютекс можна захопити кілька разів одним і тим же потоком, при цьому буде збільшуватися лічильник рекурсії.
- Функція **ReleaseMutex** - це функція звільнення мьютекса.

СЕМАФОРИ

Семафори (semaphor) - аналогічні мьютексам, але при їх роботі підраховується число звернень. Семафор - синхронізуючий об'єкт ядра, що містить лічильник числа користувачів і два 32-розрядних цілих числа зі знаком: лічильник поточного числа ресурсів від 0 і до значення максимального числа ресурсів (контрольованого семафором). За допомогою семафора можна дозволити доступ до ресурсу, наприклад, не більше ніж трьом потокам одночасно. Мьютекс це той же семафор з максимальним значенням лічильника, рівним 1. Створити семафор можна функцією **CreateSemaphore**, а відкрити його - функцією **OpenSemaphore**.

СЕМАФОРИ

При роботі з семафорами необхідно дотримуватись таких правил:

- Якщо лічильник числа ресурсів дорівнює нулю - семафор зайнятий.
- Якщо лічильник числа ресурсів більше, ніж нуль - семафор вільний.
- Лічильник числа ресурсів не може бути менше нуля або більше, ніж максимальне значення лічильника ресурсів.
- Кожне успішне очікування семафора Wait-функцією зменшує значення лічильника числа ресурсів на 1, тобто потік захоплює ресурс.
- Функція **ReleaseSemaphore** збільшує значення лічильника числа користувачів на 1, може викликатися потоком, що закінчили працювати з ресурсом, або при надходженні клієнтського запиту.
- Потоки, що очікують доступ через семафор, отримують його за принципом роботи стека FIFO (First Input First Output) - перший потік, який став на чергу очікування отримає доступ першим.

СЕМАФОРИ

Семафор може використовуватися в двох випадках.

1)Коли необхідно надати доступ до ресурсу обмеженому числу потоків або точніше, коли на ресурс претендує більша кількість потоків, ніж може собі це дозволити ресурс.

2)Коли організується буфер зберігання клієнтських запитів, і коли клієнтські запити обробляються серверними потоками, В обох випадках це називають чергою. Збільшити значення лічильника поточного числа ресурсів можна функцією **ReleaseSemaphore**.

ПОДІЇ(EVENT)

Події (event) - можуть застосовуватися як засіб синхронізації потоку з системними подіями, такими як користувальницькі операції з файлами. Метод WaitFor класу TThread (Delphi, C + + Builder) використовує подія. Події можуть також використовуватися для того, щоб активізувати одночасно кілька потоків.

ПОДІЇ(EVENT)

Подія - найпростіший різновид синхронізуючого об'єкта ядра, використовується для повідомлення про закінчення будь-якої операції. Подія містить лічильник числа користувачів (як і всі об'єкти ядра) і дві булеві змінні: одна повідомляє тип даного об'єкта ядра «подія» (з авто-скиданням або без), інша - його стан (вільний або зайнятий). Об'єкт ядра «подія» буває двох типів:

- зі скиданням вручну (manual-reset events) - виклик Wait-функції ніяк не впливає на стан об'єкт ядра «подія», що дозволяє відновити відразу кілька чекаючих потоків і надати їм доступ «тільки для читання».
- з авто-скиданням (auto-reset events) - при успішному очікуванні Wait-функція автоматично переводить об'єкт ядра «подія» у зайнятий стан, що дозволяє відновити тільки один чекаючий потік, він отримає доступ, як для читання, так і для запису.

ПОДІЇ(EVENT)

Створити об'єкт ядра «подія» можна за допомогою функції **CreateEvent**. У числі параметрів цієї функції є два булевих параметра. Один параметр bManualReset визначає тип об'єкта ядра «подія». Якщо він дорівнює TRUE, то об'єкт ядра «подія» - зі скиданням вручну, якщо FALSE - з авто-скиданням. Параметр bInitialState визначає початковий стан об'єкта ядра «подія». Якщо він дорівнює TRUE, то об'єкт ядра «подія» має «вільне» стан, а якщо він дорівнює FALSE - «зайняте» стан. За допомогою функції OpenEvent можна відкрити об'єкт ядра «подія». Для управління станом об'єкта ядра «подія» можна використовувати такі функції: **SetEvent** - переводить об'єкт ядра «подія» у вільний стан, **ResetEvent** - переводить об'єкт ядра «подія» у зайняте стан.

ПРИКЛАД: ВИКОРИСТАННЯ ОБ'ЄКТА ЯДРА "ПОДІЯ" З АВТО-СКИДАННЯМ

```
HANDLE hMyEvent; // глобальный дескриптор події
int WINAPI WinMain(...)
{     hMyEvent = CreateEvent( NULL, FALSE, FALSE, NULL);
      HANDLE hMyThread[2];
      DWORD dwThreadId;
      hMyThread[0] = _beginthreadex( NULL, ColorCorrector,   NULL, 0, &dwThreadId);
      hMyThread[1] = _beginthreadex( NULL, ContrastCorrector, NULL, 0, &dwThreadId);

      LoadImage(...);     // тепер один з потоків отримає доступ до ресурсу
      (зображенню)
      SetEvent( hEvent );
...}
//*********************************************************************

DWORD WINAPI ColorCorrection(LPVOID lpParam) {
   WaitForSingleObject( hMyEvent, INFINITE );
// подія автоматично переводиться в зайнятий стан
// отриман доступ до зображення і виконуємо корекцію кольору зображення
SetEvent( hEvent ); // звільняємо об'єкт ядра «подія»
return(0);
}
//*********************************************************************

DWORD WINAPI ContrastCorrector(LPVOID lpParam){
    WaitForSingleObject( hMyEvent, INFINITE );
// подія автоматично переводиться в зайнятий стан
// отриман доступ до зображення і виконуємо корекцію контраста зображення
    SetEvent( hEvent );
// звільняємо об'єкт ядра «подія»
   return(0);}
```

ФУНКЦІЇ ОЧІКУВАННЯ ОБ'ЄКТІВ ЯДРА(WAIT-ФУНКЦІЇ)

Функції цього типу дозволяють потоку в будь-який момент призупинитися і чекати звільнення якого-небудь об'єкта ядра. Найпростішою і найбільш використовуваної функцією є **WaitForSingleObject**. Вона має два параметри - покажчик на об'єкт ядра і інтервал часу. Потік, викликавши цю функцію, чекає звільнення об'єкта ядра, дескриптор якого передано в першому параметрі, об'єкт ядра повинен підтримувати стани <зайнятий / вільний>. Другий параметр функції вказує, скільки часу потік (що викликає цю функцію) повинен чекати об'єкт ядра, якщо передати значення INFINITE (0xFFFFFFFF), то потік буде чекати необмежений час.

Функція повертає такі значення:

- WAIT_OBJECT_0 - об'єкт ядра став вільним
- WAIT_TIMEOUT - час очікування вийшов
- WAIT_FAILED - помилка функції (невірний дескриптор), GetLastError
- WAIT_ABANDONED - зазначений об'єкт є об'єктом ядра <мьютекс>, який не звільнений потоком, якому він належить і цей потік не завершений. Володіння об'єктом <мьютекс> передано потоку, що визиває.

ФУНКЦІЇ ОЧІКУВАННЯ ОБ’ЄКТІВ ЯДРА(WAIT-ФУНКЦІЇ)

Функція WaitForSingleObject може бути використана для роботи з такими об'єктами:

- Зміна повідомлення (Change notification),
- Консольний введення (Console input),
- Подія (Event), мьютекс (Mutex),
- Процес (Process),
- Семафор (Semaphore),
- Потік (Thread),
- Таймер (Timer).

ФУНКЦІЇ ОЧІКУВАННЯ ОБ'ЄКТІВ ЯДРА (WAIT-ФУНКЦІЇ)

Для очікування відразу декількох об'єктів ядра або якогось одного з декількох, використовується функція **WaitForMultipleObjects**. Вона має такі параметри:

- Параметр nCount визначає кількість об'єктів ядра в масиві lpHandles - це значення може перебувати в межах від 1 до MAXIMUM_WAIT_OBJECTS (визначено як 64);
- Параметр lpHandles - покажчик на масив покажчиків очікуваних об'єктів;
- Параметр fWaitAll визначає - чекати звільнення всіх об'єктів ядра відразу (TRUE) або якогось одного (FALSE);
- Параметр dwMilliseconds визначає час очікування об'єктів в мілісекунди.

Значення, що повертаються аналогічні функції WaitForSingleObject за винятком WAIT_OBJECT_0, тут це значення говорить тільки про нульовий об'єкті ядра з масиву покажчиків. Щоб дізнатися який саме об'єкт ядра із зазначеного масиву звільнився, необхідно додати номер об'єкта в масиві до значення WAIT_OBJECT_0. Нумерація в масиві покажчиків починається з нуля.

ПРИКЛАД: ЗАСТОСУВАННЯ ФУНКЦІЇ WAITFORMULTIPLEOBJECT

```
WAIT_OBJECT_0 + 0;
WAIT_OBJECT_0 + 1;
WAIT_OBJECT_0 + 2;
HANDLE hProc[3];
hProc[0] = hMyProc1;
hProc[1] = hMyProc2;
hProc[2] = hMyProc3;
DWORD dwResult = WaitForMultipleObjects( 3, hProc, FALSE, 5000);
switch(dwResult) {
    case WAIT_FAILED: // неправильный вызов функции (неверный
дескриптор?)
        break;
    case WAIT_TIMEOUT: // время вышло, ни один из объектов не
освободился за 5000 мс
        break;
    case (WAIT_OBJECT_0 + 0):
// завершился процесс, идентифицируемый hProc[0] или hMyProc1
        break;
    case (WAIT_OBJECT_0 + 1):
// завершился процесс, идентифицируемый hProc[1] или hMyProc2
        break;
    case (WAIT_OBJECT_0 + 2):
// завершился процесс, идентифицируемый hProc[2] или hMyProc3
       break;
}
```