



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
імені Семена Кузнеця





Лекція № 5

Змістовний модуль № 1: Системне програмування в Windows

по курсу 'Системне програмування'

Тема лекції: Засоби міжпроцесного взаємодії

Лектор:

*Доцент кафедри Інформаційних систем
кандидат технічних наук, доцент
Голубничий Дмитро Юрійович*

НАВЧАЛЬНІ ПИТАННЯ:

1.

Засоби міжпроцесного обміну даними.

2.

Синхронізація потоків в режимі ядра.

3.

Синхронізація потоків у виконавчій системі.

4.

Синхронізація потоків в режимі користувача.



Вступ



Засоби взаємодії між процесами (InterProcess Communication - IPC)

Засоби міжпроцесної синхронізації

семафори

критичні секції

Виключають семафори
(м'ютекси)

події

очікують таймери

блокують змінні

Сокети (sockets)

Технології OLE / ActiveX

Засоби міжпроцесного обміну даними

Буфер обміну

атоми

канали

поштові слоти

Передача повідомлень
між процесами

хуки

колективна пам'ять

Бібліотеки динамічного
компонування (DLL)

Протокол динамічного
обміну даними (DDE)

Рис. 1. Класифікація засобів взаємодії між процесами

СИНХРОНІЗАЦІЯ - узгодження швидкостей потоків шляхом припинення потоку до настання деякої події і подальшій його активізації при настанні цієї події

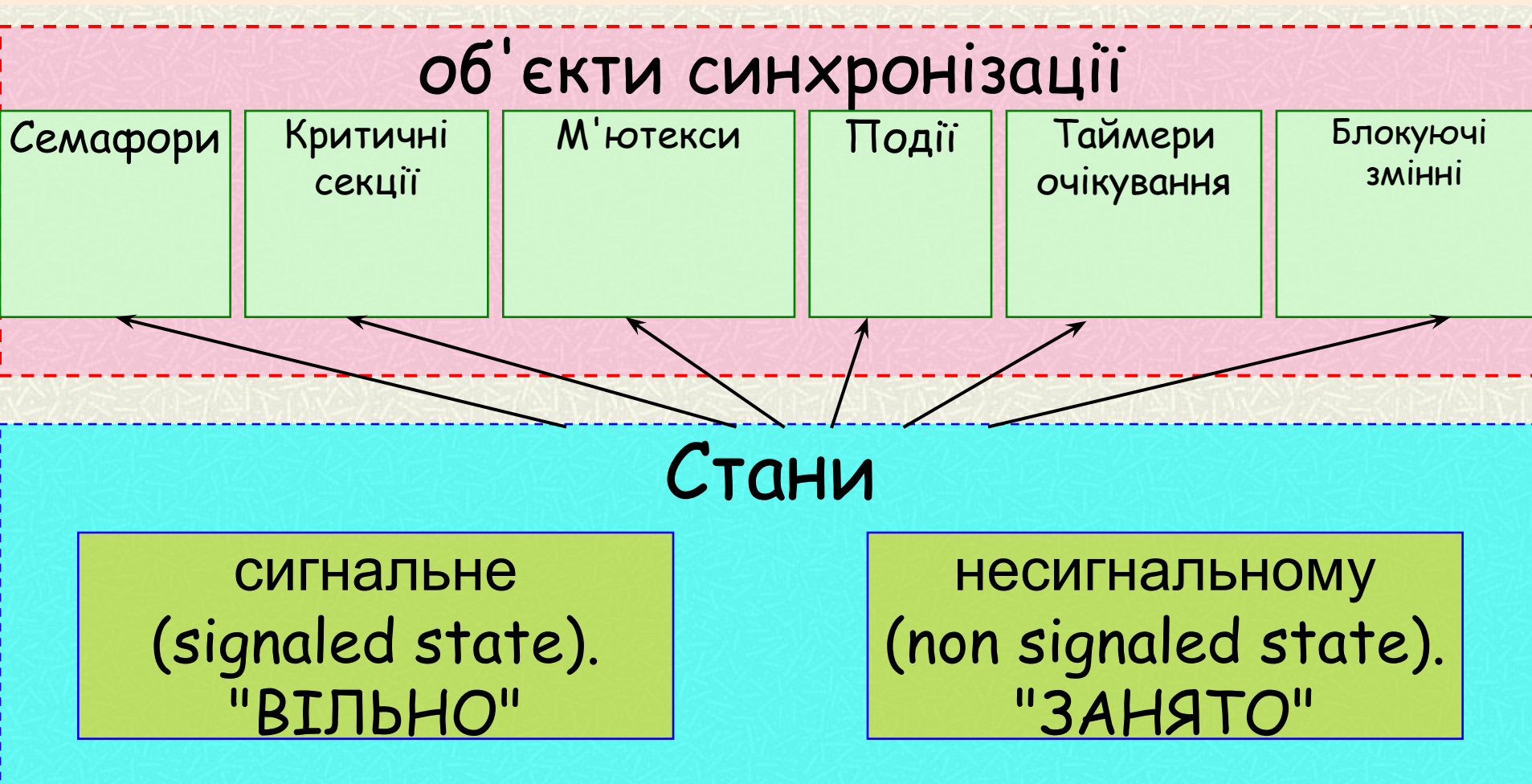
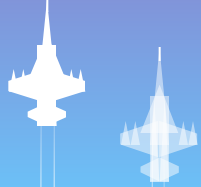


Рис. 2. Види станів об'єктів синхронізації



1.1. Буфер обміну



Буфер обміну (clipboard) Windows забезпечує простий обмін даними між додатками



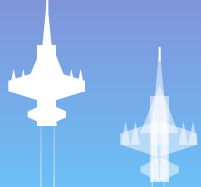
Рис. 1. Взаємодія між власником і клієнтом буфера обміну

Приклади визначених форматів буфера обміну

<i>Формат буфера обміну</i>	<i>Тип дескриптора</i>	<i>опис даних</i>
CF_BITMAP	HBITMAP	Дані являють собою набір бітів.
CF_DSPTEXT	HANDLE	Приватний для додатка текст.
CF_DSPBITMAP	HBITMAP	Растрове зображення, яке є приватним для додатка.
CF_GDIOBJFIRST	HGDIOBJ	Описані додатком формати наскрізного буфера обміну (throughclipboard), представлені об'єктами GDI
CF_OEMTEXT	HANDLE	Об'єкт пам'яті, який містить завершується нулем рядок символів набору OEM.
CF_OWNERDISPLAY	NULL	Вказує, що власник буфера обміну буде відповідати за відображення даних, а також оновлює вікна перегляді буфер обміну. Вікно перегляду буфера обміну відправляє власнику повідомлення WM_ASKCBFORMATNAME , WM_PAINTCLIPBOARD , WM_HSCROLLCLIPBOARD , WM_SIZECLIPBOARD і WM_VSCROLLCLIPBOARD .
Від CF_PRIVATEFIRST до CF_PRIVATELAST		Цей діапазон позначає приватні формати буфера обміну. Windows не керує цими форматами. Власник буфера обміну повинен управляти ресурсів »через повідомлення WM_DESTROYCLBOARD .
CF_RBIFF	HANDLE	Складна підтримка звукових даних.
CF_TEXT	HANDLE	Об'єкт пам'яті, що містить рядок символів, завершується нулем.
CF_TIFF	HANDLE	Формат дескриптора файлу зображення.
CF_UNICODETEXT	HANDLE	Об'єкт пам'яті, що містить завершується нулем рядок у форматі багатобайтові глобального коду символів Unicode.
CF_WAVE	HANDLE	Стандартна підтримка Wave-файлів.

Формат CF_OWNERDISPLAY

Повідомлення	Значення
WM_ASKCBFORMATNAME	Відправляється, коли вікно перегляду буфера обміну запитує ім'я формату. Власник буфера обміну повинен скопіювати байти wParam в буфер, на який вказує параметр lParam .
WM_PAINTCLIPBOARD	Відправляється, коли клієнтська область вікна буфера обміну вимагає оновлення. параметр wParam є дескриптором вікна перегляду буфера обміну. параметр lParam є покажчиком на PAINTSTRUCT .
WM_SIZECLIPBOARD	Відправляється при зміні розміру клієнтської області вікна перегляду буфера обміну. параметр wParam є дескриптором вікна перегляду буфера обміну. параметр lParam - покажчик на структуру RECT .
WM_HSCROLLCLIPBOARD і WM_VSCROLLCLIPBOARD	Відправляється при прокручуванні клієнтської області вікна перегляду буфера обміну. параметр wParam є дескриптором вікна перегляду буфера обміну. Молодше слово параметра lParam позначає тип запиту смуги прокрутки (подібно параметру wParam в повідомленні WM_HSCROLL або WM_VSCROLL). Старше слово параметра lParam вказує позицію бігунка тоді і тільки тоді, коли смуга прокрутки запитує SB_THUMBPOSITION .



1.2. Атому



Атом (*atom*) являє собою унікальне 16-розрядне значення, яке пов'язане зі строковою константою. Значення рядка, яке представляє атом, відоме під назвою **імені атома (*atom name*)**.

1. Збереження рядка в локальній таблиці атомів
ATOM AddAtom (LPCTSTR *lpzStringToStore*)
2. Зменшення лічильника посилань на атом
ATOM DeleteAtom (ATOM *nAtom*)
3. Пошук атома
ATOM FindAtom (LPCTSTR *lpzString*)
4. Повернення імені атома з таблиці атомів
UINT GetAtomName (ATOM *nAtom*, LPTSTR *lpBuffer*, int *nSize*)
5. Встановлення кількості елементів верхнього рівня в локальній таблиці атомів
BOOL InitAtomTable (DWORD *nSize*)

Приклади роботи з атомами

```
LPCTSTR szAtom = "Atom";
LRESULT CALLBACK WndProc (HWND hWnd, UINT
uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_CREATE:
// Збільшити до 73 число елементів у верхній частині
таблиці атомів.
        InitAtomTable(73);
        break;
        case WM_PAINT:
        {
            // Показати результати пошуку атома.
            static PAINTSTRUCT ps;
            static char szWorkArea [33];
            static char szBuffer [128];
            static ATOM aAnAtom;
            aAnAtom = INVALID_ATOM;
            BeginPaint (hWnd, & ps);
            if (aAnAtom = FindAtom(SzAtom))
            {
                GetAtomName(AAnAtom, szWorkArea, 32);
                wsprintf (szBuffer, "атом НЕ знайдений. ",
SzWorkArea);
            }
            else
                lstrcpy( szBuffer, "Атом може бути доданий." );
```

```
TextOut( ps.hdc, 0, 0, szBuffer, lstrlen( szBuffer ));
        EndPaint (hWnd, & ps);
    }
    break;
case WM_COMMAND:
    switch (LOWORD (wParam))
    {
        case IDM_ADD:
            // ввести атом.
            AddAtom(SzAtom);
            InvalidateRect (hWnd, NULL, TRUE);
            break;
        case IDM_DELETE:
            // Знайти і видалити атом.
            if ( FindAtom(SzAtom))
                DeleteAtom(FindAtom (szAtom));
            InvalidateRect (hWnd, NULL, TRUE);
            break;
        ...
```

Застосування цілих атомів

```
char szStoredString [6];
WORD wValue = 100;
ATOM aValue = AddAtom(MAKEINTATOM(wValue));
// значення атома одно 100.
// рядок буде містити число "#100".
GetAtomName( aValue, szStoredString, 6 );
```



1.3. Канали передачі даних



Канали (pipe)

Організувати передачу даних між локальними процесами

дозволяють

Організувати передачу даних між процесами, запущеними на різних робочих станціях в мережі

стандарт UNC

Стосовно до каналів (в загальному вигляді)

\\ІмяСервера\pipe\Ім'я_Канала

Стосовно до каналів (для одного комп'ютера)

\\.\Pipe\Ім'я_Канала

ФУНКЦІЇ ДЛЯ РОБОТИ З КАНАЛАМИ

№ п/п	Функція	Призначення
1.	CreatePipe ()	Створення анонімного каналу
2.	CreateNamedPipe ()	Створення іменованого каналу
3.	ConnectNamedPipe ()	Установка з'єднання з каналом з боку сервера
4.	CreateFile ()	Установка з'єднання з каналом з боку клієнта
5.	DisconnectNamedPipe ()	Відключення серверного процесу від клієнтського процесу
6.	CloseHandle ()	Закриття хендлом каналу
7.	WriteFile ()	Запис даних у відкритий канал
8.	ReadFile ()	Читання даних з каналу
9.	PeekNamedPipe ()	Читання даних з каналу без видалення
10.	CallNamedPipe ()	Прийом / передача даних від каналу
11.	TransactNamedPipe ()	Прийом / передача даних від каналу
12.	WaitNamedPipe ()	Очікування процесом доступу до каналу для з'єднання
13.	SetNamedPipeHandleState ()	Змінити режими роботи для вже створеного каналу.
14.	GetNamedPipeHandleState ()	Визначити стан каналу.
15.	GetNamedPipeInfo ()	Отримати інформацію про іменований канал

Режими роботи каналу

HANDLE CreateNamedPipe (LPCTSTR lpName, DWORD dwOpenMode, DWORD dwPipeMode, DWORD nMaxInstances, DWORD nOutBufferSize, DWORD nInBufferSize, DWORD nDefaultTimeout, LPSECURITY_ATTRIBUTES lpSecurityAttributes)

DWORD dwOpenMode

Способи передачі даних через канал

Передача потоку байт (PIPE_TYPE_BYTE)

Передача повідомлень (PIPE_TYPE_MESSAGE)

Режим відкриття каналу

прапори захисту

Тільки для читання PIPE_ACCESS_INBOUND

Тільки для запису PIPE_ACCESS_OUTBOUND

Для читання і запису PIPE_ACCESS_DUPLEX

DWORD dwPipeMode

Способи використання каналу

Читання в режимі послідовної передачі байт (PIPE_READMODE_BYTE)

Читання в режимі передачі окремих повідомлень (PIPE_READMODE_MESSAGE)

PIPE_WAIT - Блокуючий режим

PIPE_NOWAIT - Неблокуючий режим

Приклад 1 Створити іменований канал з ім'ям \$MyPipe\$

Рішення

```
// Ідентифікатор каналу Pipe
HANDLE hNamedPipe;
// Ім'я створюваного каналу Pipe
LPSTR lpszPipeName = "\\.\pipe\ $ MyPipe $";
// Створюємо канал Pipe, що має ім'я lpszPipeName
hNamedPipe = CreateNamedPipe( lpszPipeName,
PIPE_ACCESS_DUPLEX, //режим відкриття каналу
PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT, //режим роботи
PIPE_UNLIMITED_INSTANCES, // максимальна кількість реалізацій каналу
512, // розмір вихідного буфера в байтах
512, // розмір вхідного буфера в байтах
100, // час очікування в мілісекундах
NULL); // покажчик на атрибути захисту
// Якщо виникла помилка, виводимо її код і зваершаем роботу додатка
if (hNamedPipe == INVALID_HANDLE_VALUE)
{
    fprintf (stdout, "CreateNamedPipe: Error%ld \n",
        GetLastError ());
    getch ();
    return 0;
}
```

З'єднання з каналом з боку сервера

BOOL ConnectNamedPipe (

HANDLE *hNamedPipe*, // хендл іменованого каналу

LPOVERLAPPED *lpOverlapped* // покажчик на структуру *OVERLAPPED*

асинхронний режим

синхронний режим

структура **OVERLAPPED**

NULL

```
typedef struct _OVERLAPPED {
```

```
DWORD Internal; //стан системи
```

```
DWORD InternalHigh; // число прочитаних або записаних байтів
```

```
DWORD Offset; // молодші розряди зміщення у файлі
```

```
DWORD OffsetHigh; // старші розряди зміщення в файлі
```

```
HANDLE hEvent; // хендл об'єкта синхронізації
```

```
} OVERLAPPED, * LPOVERLAPPED;
```

Відключення серверного процесу від клієнтського процесу

BOOL DisconnectNamedPipe (HANDLE *hNamedPipe*)

З'єднання з каналом з боку клієнта

HANDLE CreateFile (

LPCTSTR IpFileName, // покажчик на рядок імені файлу (каналу)

DWORD dwDesiredAccess, //режим доступу

DWORD dwShareMode, // режим спільного використання файлу

LPSECURITY_ATTRIBUTES IpSecurityAttributes, // атрибути захисту

DWORD dwCreationDistribution, // параметри створення

DWORD dwFlagsAndAttributes, // атрибути файлу

HANDLE hTemplateFile // хендл файлу з атрибутами

)

DWORD dwShareMode



Константа	опис
NULL	Спільне використання файлу заборонено
FILE_SHARE_READ	Інші процеси можуть відкривати файл для читання
FILE_SHARE_WRITE	Інші процеси можуть відкривати файл на запис

Закриття іменованого каналу

CloseHandle
hNamedPipe).

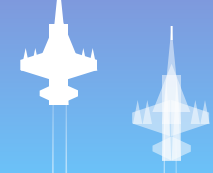
(HANDLE

Приклад 2

Фрагмент клієнтського додатка, який відкриває канал з ім'ям \$MyPipe\$. Канал відкривається як для запису, так і для читання

Рішення

```
char szPipeName [256];  
HANDLE hNamedPipe;  
strcpy (szPipeName, "\\.\ pipe \ $ MyPipe  
$");  
hNamedPipe = CreateFile (  
    szPipeName, GENERIC_READ | GENERIC_WRITE,  
    0, NULL, OPEN_EXISTING, 0, NULL);
```



1.4. Поштові канали передачі даних (MailSlot)



MailSlot - це файл, що знаходиться в пам'яті, доступ до якого здійснюється стандартними файловими функціями Win32. Загальний розмір даних не може бути більше **64К**.

Типи процесів

MailSlot-сервер

Процес-сервер може завести поштову скриньку і дати йому ім'я, глобальне в мережі.

MailSlot-клієнт

Процес-клієнт може за допомогою операцій роботи з файлами відправити дані в поштову скриньку.

Сервер **не може** виконувати над каналом Mailslot операцію запису, так як цей канал призначений тільки для односторонньої передачі даних - від клієнта до сервера.

Функції для роботи з каналами MailSlot

Функція	Призначення
CreateMailslot()	Створення каналу MailSlot
CloseHandle()	Закриття хендлом каналу
GetMailslotInfo()	Визначення поточного стану каналу Mailslot.
SetMailslotInfo()	Зміна часу очікування для каналу Mailslot після його створення

стандарт UNC

Стосовно до поштових каналів (всередині мережі)

\\ім'я комп'ютера\Mailslot\[Шлях]Ім'я_Канала

Стосовно до поштових каналів (всіх комп'ютерів домену)

\\.\Mailslot\[Шлях] Ім'я_Канала

Стосовно до поштових каналів (одночасно всіх станціях мережі)

***\Mailslot\[Шлях] Ім'я_Канала**

Приклад 3

Створення каналу Mailslot

```
LPSTR lpszMailslotName = "\\.\mailslot\\$MailslotName$";  
hMailslot = CreateMailslot (lpszMailslotName, 0,  
MAILSLOT_WAIT_FOREVER, NULL);
```

Відкриття каналу Mailslot

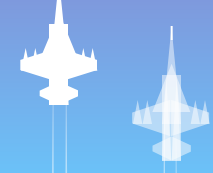
```
LPSTR lpszMailslotName = "\\.\mailslot\\$MailslotName$";  
hMailslot = CreateFile (lpszMailslotName, GENERIC_WRITE,  
FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
```

Запис повідомлень в канал Mailslot

```
HANDLE hMailslot;  
char szBuf[512];  
DWORD cbWritten;  
WriteFile (hMailslot, szBuf, strlen (szBuf) + 1, &  
cbWritten, NULL);
```

Читання повідомлень з каналу Mailslot

```
HANDLE hMailslot;  
char szBuf [512];  
DWORD cbRead;  
ReadFile (hMailslot, szBuf, 512, & cbRead, NULL);
```



1.5. Передача повідомлень між процесами

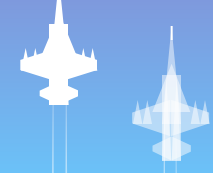


Метод заснований на передачі повідомлення **WM_COPYDATA** за допомогою функції **SendMessage()**

wParam - ідентифікатор вікна, що посилає повідомлення.

lParam - покажчик на попередньо заповнену структуру
COPYDATASTRUCT

```
typedef struct tagCOPYDATASTRUCT
{
    DWORD dwData; //32-розрядні дані
    DWORD cbData; // розмір переданого буфера з даними
    PVOID lpData; // покажчик на буфер з даними
} COPYDATASTRUCT;
```



1.6. Колективна пам'ять



Колективна пам'ять є собою сегмент фізичної пам'яті, відображеної в віртуальному адресному просторі двох або більше процесів.

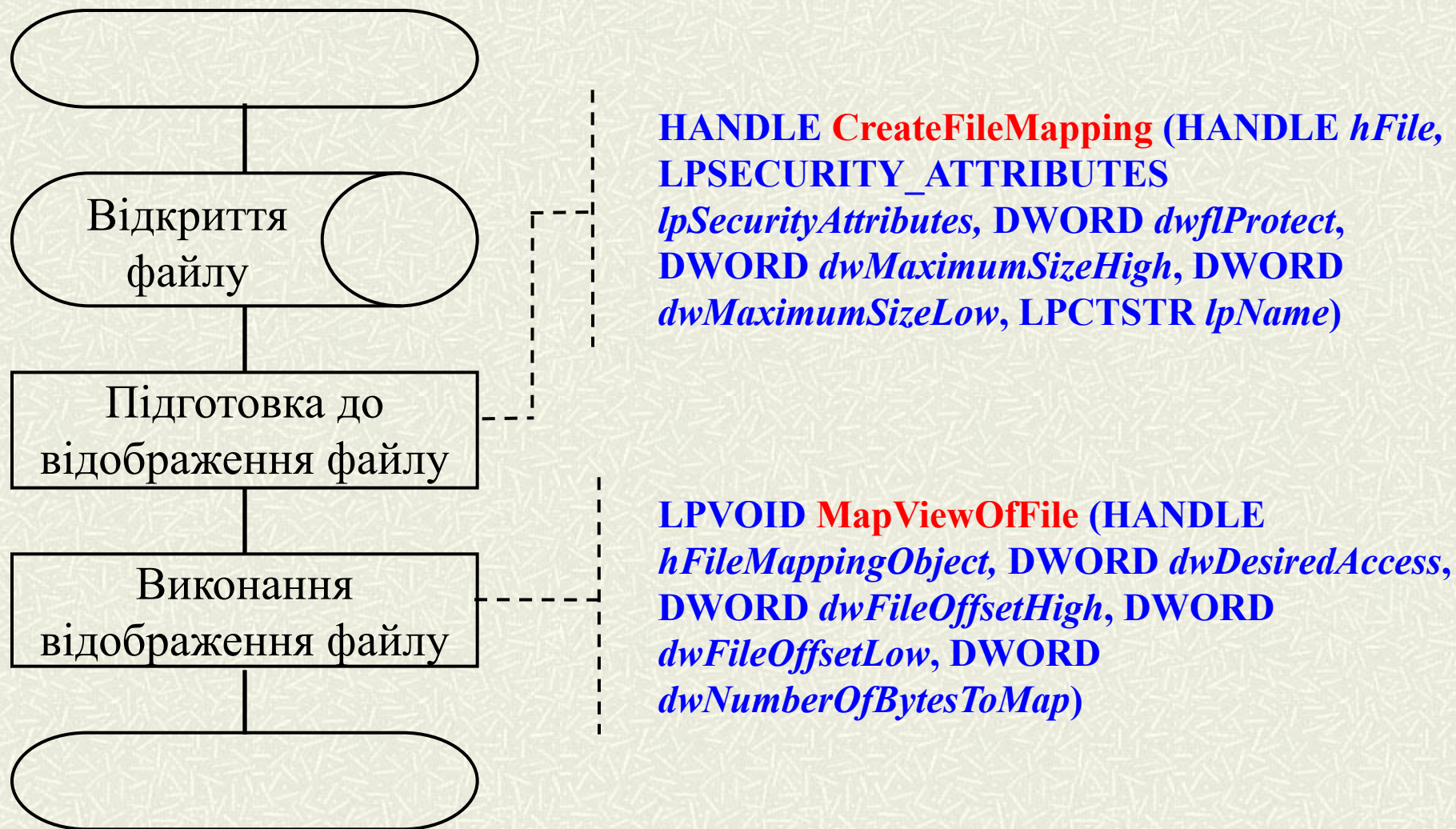


Схема створення відображення файлу



2. Синхронізація потоків в режимі ядра.

2.1. Блокуючі змінні



Атомарний доступ (Atomic access) -

МОНОПОЛЬНИЙ захоплення ресурсу потоком, що до нього звертається

Приклад № 1

```
long x = 0;
DWORD WINAPI ThreadFunc1
(PVOID pvParam)
{
    x ++;
    return (0);
}
```

```
DWORD WINAPI ThreadFunc2
(PVOID pvParam)
{
    x ++;
    return (0);
}
```

**Питання.
Чому
дорівнює
значення
глобальної
змінної x?**

Сімейство *Interlocked*-функцій

1. Обмін однієї змінної на значення типу LONG

```
LONG InterlockedExchange (LPLONG lpTarget, LONG  
lNewVal);
```

2. Додавання однієї змінної з іншого

```
LONG InterlockedExchangeAdd (PLONG pAddend, LONG  
lIncrement);
```

3. Збільшення на 1 змінної

```
LONG InterlockedIncrement (LPLONG lpAddend);
```

4. Зменшення на 1 змінної

```
LONG InterlockedDecrement (LPLONG lpAddend);
```

5. Якщо pIDestination = ICommand, то * pIDestination буде IExchange

```
PVOID InterlockedCompareExchange(PLONG pIDestination,  
LONG IExchange, LONG ICommand);
```

6. аналог функції InterlockedCompareExchange для 64 розрядних

```
PVOID InterlockedCompareExchangePointer (PVOID *  
ppvDestination, PVOID pvExchange, PVOID pvComparand)
```


Сімейство *Interlocked*-функцій

Приклад № 2

```
// визначаємо глобальну змінну
long g_x = 0;

DWORD WINAPI ThreadFunc1 (PVOID
pvParam)
{
InterlockedExchangeAdd (& g_x,
1);
return (0);
}

DWORD WINAPI ThreadFunc2 (PVOID
pvParam)
{
InterlockedExchangeAdd (& g_x,
1);
return (0);
}
```

```
// змінна типу
//LONG, використовувана
//декількома потоками
LONG g_x;
// неправильний спосіб
//збільшення змінної
//типу LONG
g_x ++;
// правильний спосіб
//збільшення змінної
//типу LONG
InterlockedExchangeAdd
(& g_x, 1);
```

Сімейство *Interlocked*-функцій

Приклад № 3

Використання *Interlocked*-функцій при спіні-блокуванні (spinlock)

```
// глобальна змінна, яка використовується як індикатор  
// того, чи зайнятий розділяється ресурс  
BOOL g_fResourceInUse = FALSE;  
...  
void Func1 ()  
{  
// чекаємо доступу до ресурсу  
while (InterlockedExchange (& g_fResourceInUse, TRUE)  
== TRUE)  
Sleep (0);  
...  
// отримуємо ресурс в своє розпорядження  
// доступ до ресурсу більше не потрібен  
InterlockedExchange (& g_fResourceInUse, FALSE);  
}
```



2. Синхронізація потоків в режимі ядра.

2.2. Критичні секції



Критичний розділ - це блок коду, при виконанні якого потік не може бути перерваний.



Критичні секції можна використовувати для синхронізації завдань, створених різними процесами.

```
struct CRITICAL_SECTION
```

```
{
```

```
    void * DebugInfo;
```

```
    LONG LockCount;
```

```
    LONG RecursionCount;
```

```
    HANDLE OwningThread;
```

```
    HANDLE LockSemaphore;
```

```
    DWORD SpinCount;
```

```
};
```

```
win32.h
```

ОПЕРАЦІЇ З КРИТИЧНИМИ РОЗДІЛАМИ

1. Ініціалізація

```
VOID InitializeCriticalSection (LPCRITICAL_SECTION lpCrSect);
```

2. Вхід в критичну секцію

```
VOID EnterCriticalSection (LPCRITICAL_SECTION lpCrSect) ;
```

3. Вихід з критичної секції

```
VOID LeaveCriticalSection (LPCRITICAL_SECTION lpCrSect) ;
```

4. Видалення критичної секції

```
VOID DeleteCriticalSection (LPCRITICAL_SECTION lpCrSect) ;
```

Приклад № 4. Використання критичної секції

```
EnterCriticalSection (& cs);  
hdc = BeginPaint (hWnd, & ps);  
GetClientRect (hWnd, & rc);  
DrawText (hdc, "SDI Window", -1, & rc,  
DT_SINGLELINE | DT_CENTER | DT_VCENTER);  
EndPaint (hWnd, & ps);  
LeaveCriticalSection (& cs);
```

Час

блокування:

***HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\
Session Manager***

параметр *CriticalSectionTimeout* = 2592000 секунд (30 діб)



3. Синхронізація потоків у виконавчій системі

3.1. Таймери очікування



Таймери очікування (waitable timers) - це об'єкти ядра, які самостійно переходять у вільний стан в певний час або через регулярні проміжки часу.

1. Створення таймера

```
HANDLE CreateWaitableTimer(LPSECURITY_ATTRIBUTES  
lpTimerAttrib, BOOL fManualReset, LPCTSTR lpzName);
```

2. Отримання дескриптора таймера

```
HANDLE OpenWaitableTimer(DWORD dwDesiredAccess, BOOL  
blInheritHandle, LPCTSTR lpzName);
```

3. Створення копії дескриптора об'єкта

```
BOOL DuplicateHandle (HANDLE hSourceProcess, HANDLE hSourceObject,  
HANDLE hTargetProcess, LPHANDLE lphTarget, DWORD dwAccessFlags,  
BOOL, DWORD dwOptions);
```

4. Перехід у вільний стан

```
BOOL SetWaitableTimer(HANDLE hTimer, Const LARGE_INTEGER  
*pDueTime, LONG lPeriod, PTIMERAPCROUTINE pfnCompletionRoutine,  
PVOID pvArgToCompletionRoutine, BOOL fResume);
```

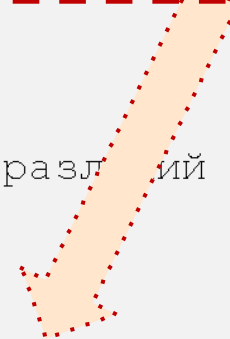
5. Видалення таймера

```
BOOL CancelWaitableTimer(HANDLE hTimer);
```


Приклад № 5. Використання таймерів очікування

```
// объявляем свои локальные переменные
HANDLE hTimer;
SYSTEMTIME st;
FILETIME ftLocal, ftUTC;
LARGE_INTEGER liUTC;
// создаем таймер с автосбросом
hTimer = CreateWaitableTimer(NULL, FALSE, NULL);
// таймер должен сработать в первый раз 1 января 2010 года в 13:00
// по местному времени
st.wYear = 2010;           // год
st.wMonth = 1;            // январь
st.wDayOfWeek = 0;        // игнорируется 0
st.wDay = 1,               // первое число месяца
st.wHour = 13;            // 1 PM
st.wMinute = 0;           // 0 минут
st.wSecond = 0,           // 0 секунд
st.wMilliseconds = 0;     // 0 миллисекунд
SystemTimeToFileTime(&st, &ftLocal);
// преобразуем местное время в UTC-время
LocalFileTimeToFileTime(&ftLocal, &ftUTC);
// преобразуем FILETIME в LARGE_INTEGER из-за разл. в выравнении
данных
liUTC.LowPart = ftUTC.wLowDateTime;
liUTC.HighPart = ftUTC.dwHighDateTime;
// устанавливаем таймер
SetWaitableTimer(hTimer, &liUTC, 6 * 60 * 60 * 1000, NULL, NULL, FALSE);
```

**Година:хвилина:секунда:
мілісекунда**



Приклад № 6. Спрацьовування таймера через 5 сек.

```
// оголошуємо свої локальні змінні
HANDLEF hTimer;
LARGE_INTEGER li;
// створюємо таймер з автоскиданням
hTimer = CreateWaitableTimer (NULL, FALSE, NULL);
// таймер повинен спрацювати через 5 секунд після виклику
SetWaitableTimer;
// задаємо час в інтервалах по 100 нс
const int nTimerUnitsPerSecond = 10000000;
// робимо отримане значення негативним, щоб //
SetWaitableTimer знав:
// нам потрібен відносний, а не абсолютний час
li.QuadPart = - (5 * nTimerUnitsPerSecond);
// встановлюємо таймер (він спрацьовує спочатку через 5
секунд, а потім через кожні 6 годин)
SetWaitableTimer (hTimer, &li, 6 * 60 * 60 * 1000, NULL,
NULL, FALSE);
...
```



4. Синхронізація потоків

в призначеному для користувача режимі

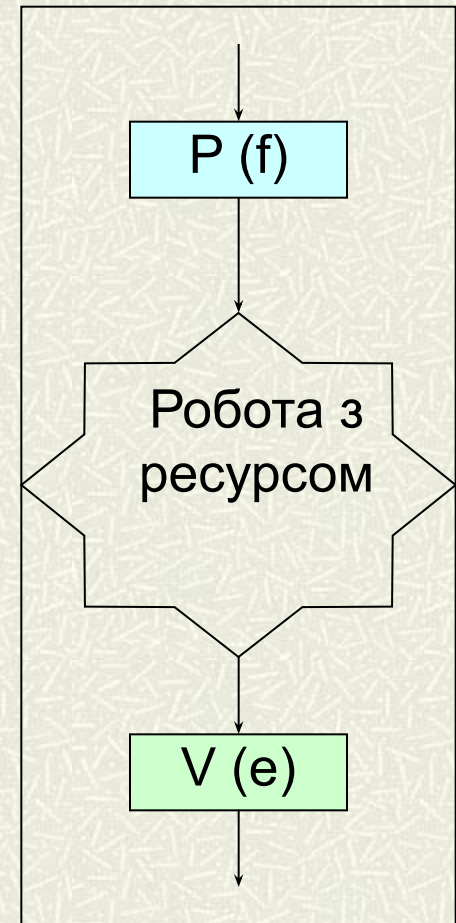
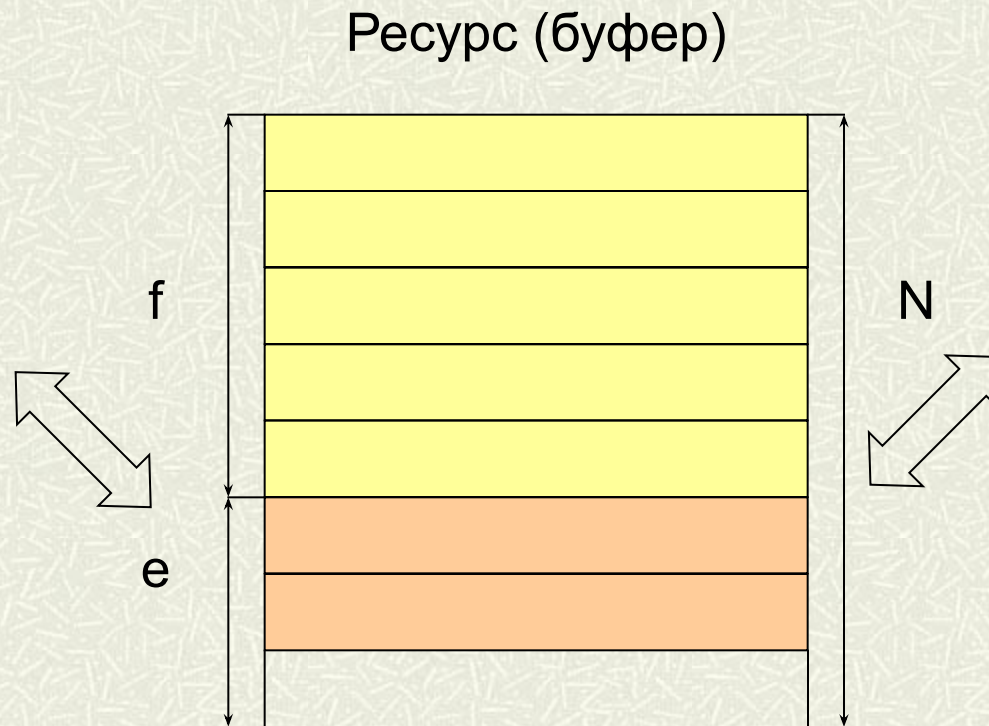
4.1. Семафори



Початкові значення семафорів: $e = N$
 $f = 0$



потік-виробник



потік-споживач

Використання семафорів для синхронізації потоків

Функції для роботи з семафора

1. Створення семафора

HANDLE CreateSemaphore (LPSECURITY_ATTRIBUTES
lpThreadSecurity, LONG *lSemInitialCount*, LONG *lSemMaxCount*,
LPCTSTR *lpzSemName*);

2. Відкриття семафора

HANDLE OpenSemaphore (DWORD *dwAccessFlag*, BOOL
blnherit, LPCTSTR *lpzSemName*);

3. Збільшення значення лічильника семафора

BOOL ReleaseSemaphore (HANDLE *hSemaphore*, LONG
lReleaseCount, LPLONG *lpPreviousCount*);

4. Зменшення значення лічильника семафора (функції очікування)

DWORD WaitForSingleObject (HANDLE *handle*, DWORD *timeout*);

DWORD WaitForMultipleObjects (DWORD *count*, CONST
HANDLE **handles*, BOOL *waitall*, DWORD *timeout*);

```
DWORD WINAPI ThreadProc (LPDWORD lpData)
```

```
{
```

```
    TCHAR szBuffer [256];
```

```
    DWORD dwSemCount = 0;
```

```
    HWND hList = (HWND) lpData;
```

```
    HANDLE hSemaphore = OpenSemaphore (SYNCHRONIZE |  
SEMAPHORE_MODIFY_STATE, FALSE, lpzSemaphore);
```

```
    wsprintf (szBuffer, "Thread% x waiting for semaphore% x", GetCurrentThreadId  
(), hSemaphore);
```

```
    SendMessage (hList, LB_INSERTSTRING, (WPARAM) -1, (LPARAM)  
szBuffer);
```

```
    // Перевірка стану семафора.
```

```
    WaitForSingleObject (hSemaphore, INFINITE);
```

```
    wsprintf (szBuffer, "Thread% x got semaphore", GetCurrentThreadId ());
```

```
    SendMessage (hList, LB_INSERTSTRING, (WPARAM) -1, (LPARAM)  
szBuffer);
```

```
    Sleep (5000);
```

```
    // Видалення семафора.
```

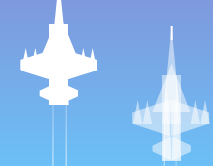
```
    ReleaseSemaphore (hSemaphore, 1, & dwSemCount);
```

```
    wsprintf (szBuffer, "Thread% x is done with semaphore. Its count was% ld.",  
GetCurrentThreadId (), dwSemCount);
```

```
    SendMessage (hList, LB_INSERTSTRING, (WPARAM) -1, (LPARAM)  
szBuffer);
```

приклад № 6.

Використання семафорів

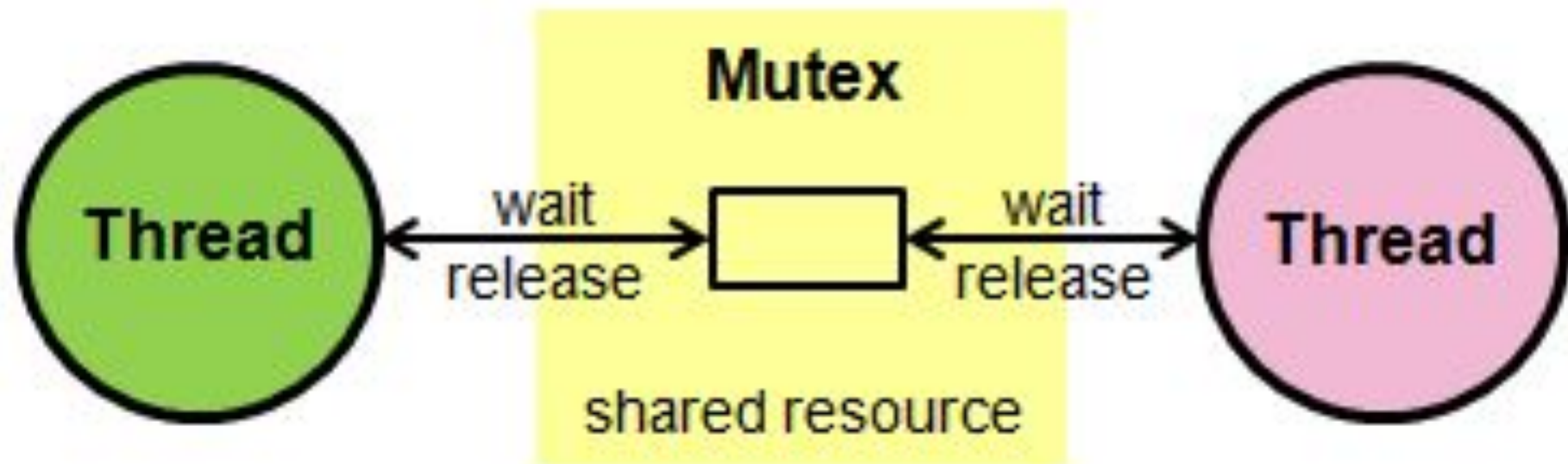


4. Синхронізація потоків в призначеному для користувача режимі

4.2. М'ютекси



М'ютекс (mutex, **mutual exclusion** — взаємне виключення) призначено для захисту певного об'єкта у потоці від доступу інших потоків. М'ютекс є одним із засобів синхронізації роботи потоків або процесів



М'ютекси — це прості двійкові семафори, які можуть перебувати в одному з двох станів - сигнальному або несигнальному (відкритий і закритий відповідно). Коли потік отримує м'ютекс, той переводиться в несигнальний стан.

Організація послідовного доступу до ресурсів з використанням м'ютексів стає нескладною, оскільки в кожен конкретний момент тільки один потік може володіти цим об'єктом. Для того, щоб об'єкт mutex став доступний потокам, що належать різним процесам, при створенні йому необхідно присвоїти ім'я. Потім це ім'я потрібно передати «у спадок» завданням, які повинні його використовувати для взаємодії.

1. Створення об'єкта **Mutex**

HANDLE CreateMutex (LPSECURITY_ATTRIBUTES
lpSecurityAttribs, BOOL *blInitialOwner*, LPCTSTR
lpzMutexName);

2. Звільнення дескриптора об'єкта

Mutex

BOOL CloseHandle (HANDLE *hObject*);

3. Відкриття об'єкту

Mutex

HANDLE OpenMutex (DWORD *dwAccessFlag*, BOOL
blInherit, LPCTSTR *lpzMutexName*);

4. Зменшення значення лічильника семафора

DWORD WaitForSingleObject (HANDLE *handle*, DWORD
timeout);

DWORD WaitForMultipleObjects (DWORD *count*,
CONST HANDLE **handles*, BOOL *waitall*, DWORD
timeout);

5. звільнення об'єкта

Mutex

BOOL ReleaseMutex (HANDLE *hMutex*)



4. Синхронізація потоків

в призначеному для користувача режимі

4.3. Події



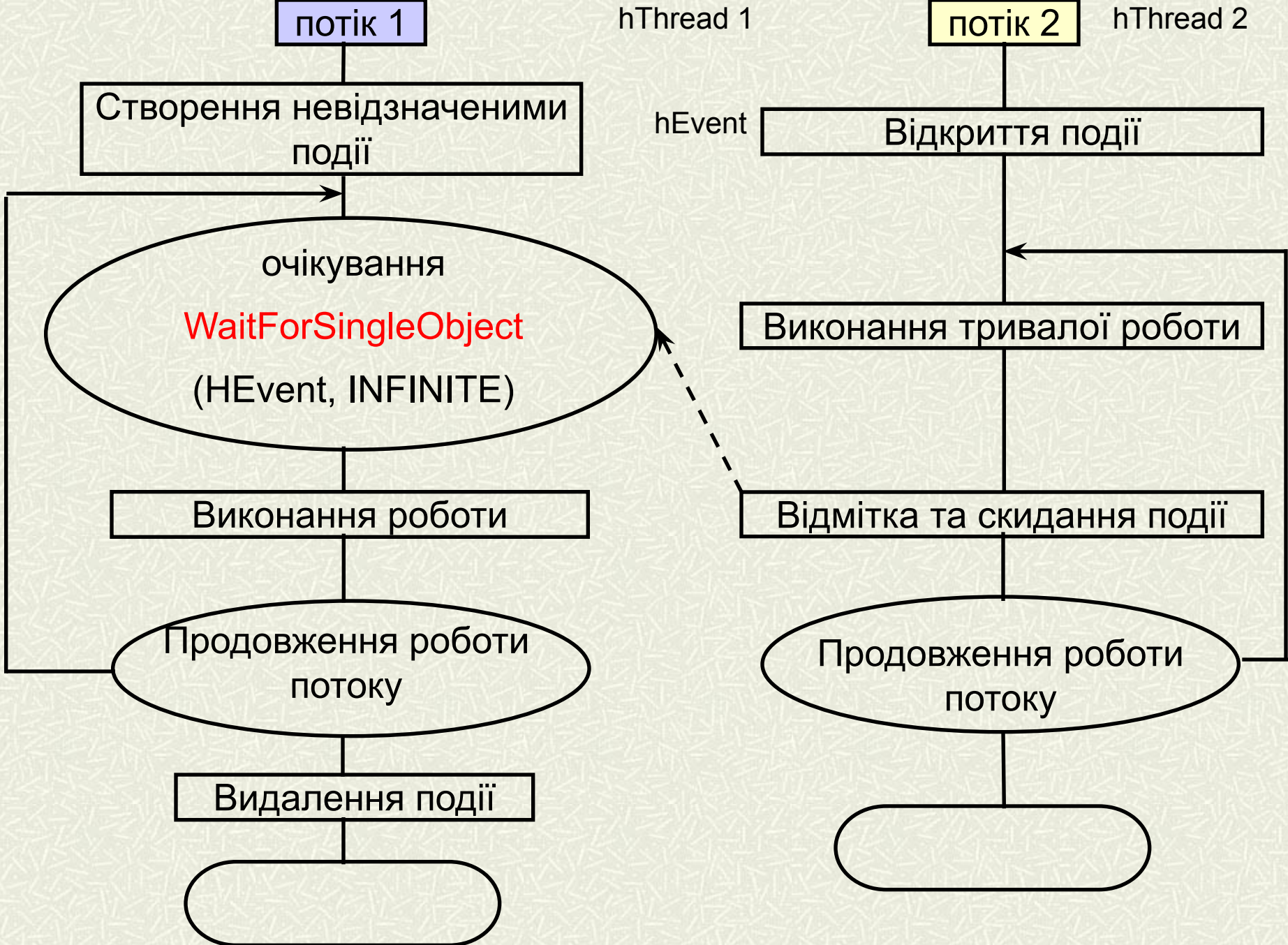


Схема використання подій

1. Створення об'єкта Event

HANDLE CreateEvent (LPSECURITY_ATTRIBUTES
lpEventSecurity, BOOL *bManualReset*, BOOL *bInitialState*,
LPCTSTR *lpzEventName*)

2. Встановлення вільного стану об'єкта

Event BOOL SetEvent (HANDLE
hEvent)

3. Встановлення зайнятого стану об'єкта

Event BOOL ResetEvent (HANDLE
hEvent)

4. Установка / скидання стану об'єкта

Event BOOL PulseEvent (HANDLE
hEvent)



Дякую за увагу!