3 Событийное управление. Структура оконного приложения, цикл обработки сообщений

3.1 Подходы к управлению и жизненный цикл процесса

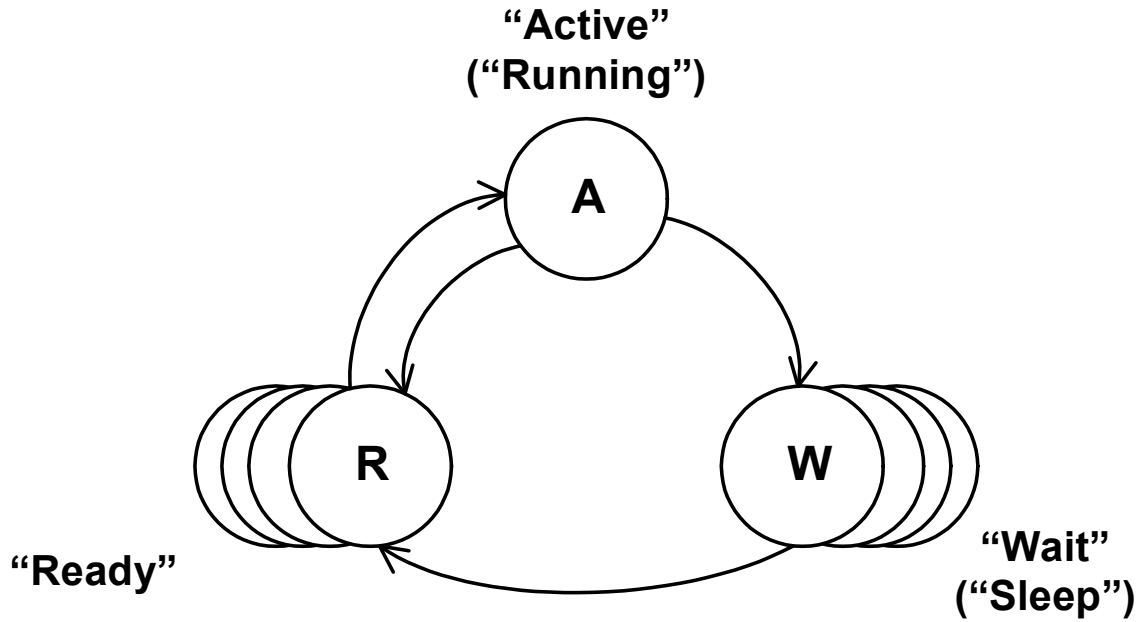
Основные ***состояния*** процесса (потока) в многозадачной системе:

– ***P*** – ***Passive***: пассивное состояние (начальное, до старта процесса, и финальное – «зомби»)

– ***R*** – ***Ready***: готовность к выполнению (очередь на выполнение)

– ***A*** – ***Active***: активное (выполнение шагов алгоритма)

– ***W*** – ***Wait***: ожидание, «сон» (до события, времени, получения ресурса)



Цикл основных состояний процесса

Практически полезная работа выполняется в состоянии Active, в остальных программа так или иначе не функциональна.
Но состояние Wait позволяет освободить ресурсы системы для выполнения других программ. Состояние Ready – вынужденная плата за нехватку исполнительных устройств в системе.

Событие – изменение состояния системы и/или отдельных процессов в ней, в том числе в связи с аппаратными сигналами, в том числе приходящими извне системы, которые должны влиять на выполнение процессов и, следовательно, требуют соответствующей обработки.

Примеры событий:

– получение ресурсов (завершение запроса к системе)

– завершение операций ввода-вывода

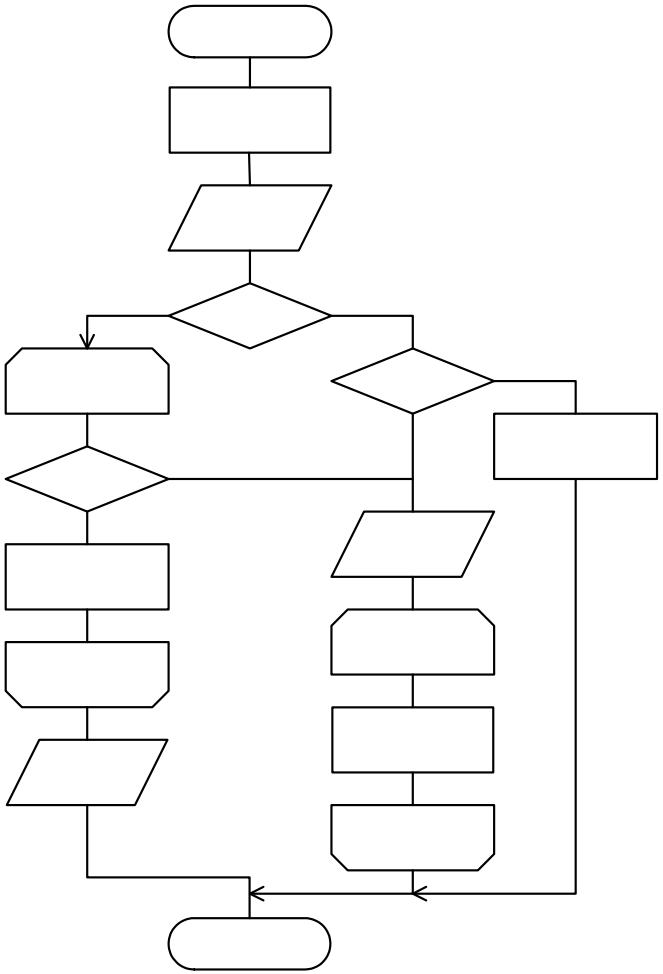
– наступление заданного момента времени (таймеры)

– исключения при выполнении программного кода

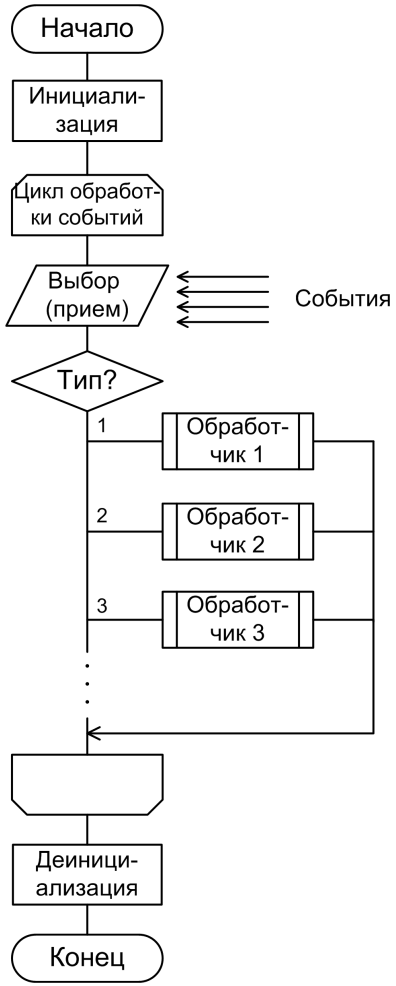
– аппаратные сигналы, и т.д.

События и необходимость реагирования на них (обработки) приводят в выходу тех или иных процессов из состояния ожидания (их «пробуждению»).

Подходы к управлению (структуре программы):



Обычный (алгоритм)



Событийно-ориентированный

Фактически – единый алгоритм распадается на множество отдельных подпрограмм-обработчиков, отвечающих каждый за одно или несколько событий, активизируемых по этим событиям и относительно слабо зависящих от других обработчиков.

Упрощается построение программы с многочисленными сложными ветвлениями, которые вызваны сочетаниями множества условий.

Усложняется взаимодействие внутри программы, приходится решать проблемы синхронизации, но зато и их решение легче поддается формализации.

Подход, основанный на событийном управлении, оказался эффективным, в частности, для интерактивных программ (взаимодействие с пользователем) и специализированных управляющих систем (взаимодействие с реальными физическими объектами и процессами).

События реализуются посредством различных программных объектов (надстроек):

– обработчики прерываний (interrupt) и исключений (exception)

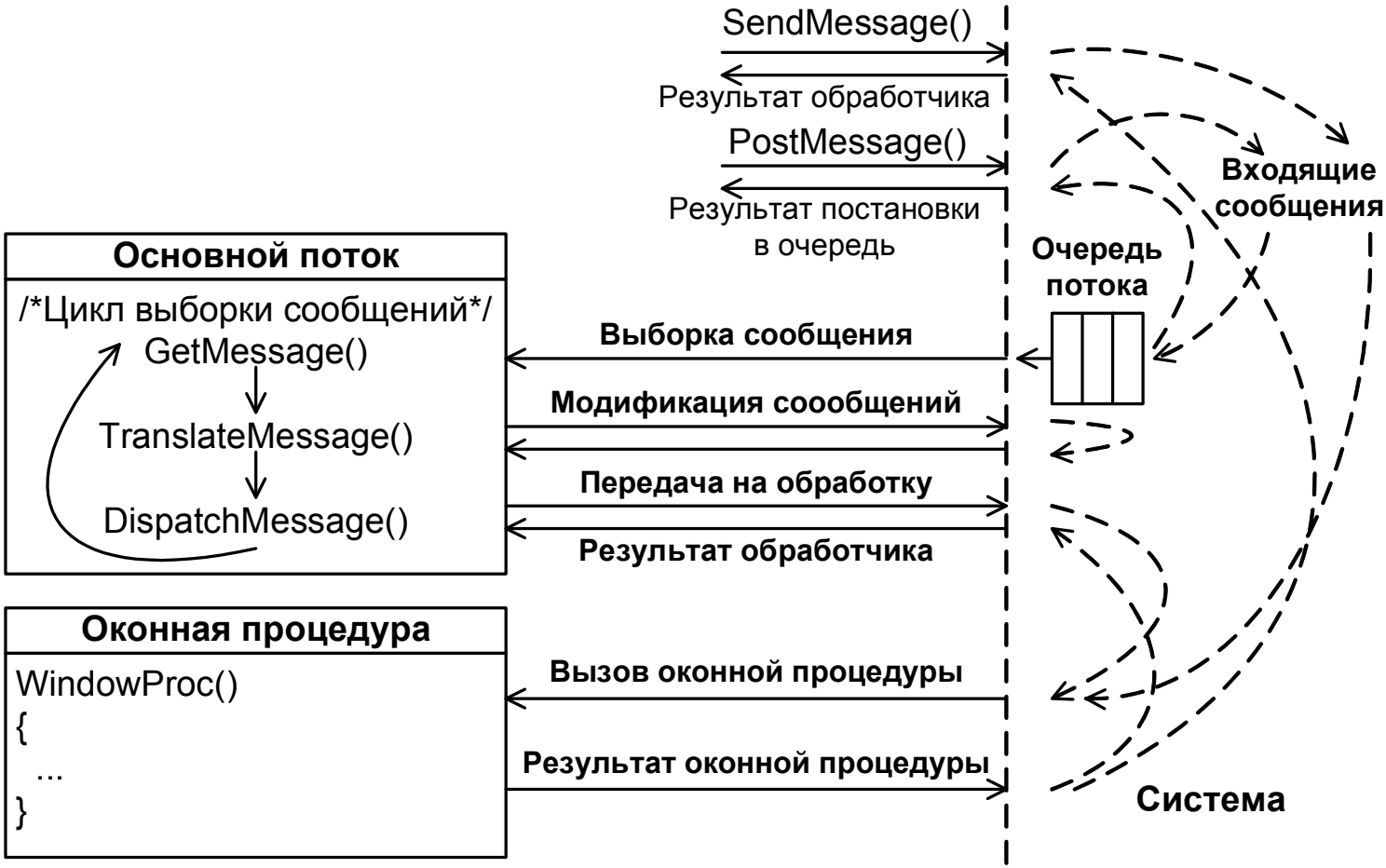
– сигналы (Unix)

– сообщения WinMessage (Windows)

– объекты синхронизации (ISO) и функции ожидания

и т.д.

3.2 Сообщения (Windows) и основной цикл оконного приложения



Обработка оконных сообщений (Windows)

Роль **`GetMessage()`** – выборка очередного сообщения из очереди, ожидание в случае отсутствия сообщений (функция **`PeekMessage()`** – проверка наличия сообщения и, при наличии, выборка, без ожидания)

Роль **`DispatchMessage()`** – перенаправление сообщения в соответствующий обработчик, выбираемый системой в зависимости от окна-получателя; обращение к функции-обработчику опосредовано системой как «обратный вызов» (callback)

Роль **`TranslateMessage()`** – явно управляемая дополнительная обработка и преобразование сообщений, генерация новых сообщений на основании поступивших. Характерный пример – дополнительная обработка ввода с клавиатуры, включая управляющие комбинации, переключение режимов, локализация ввода и т.д.

Пример – ввод с клавиатуры:

Аппаратные события (поступление scan-кодов от клавиатуры)

↓

«Первичные» сообщения («виртуальные» коды клавиш: нажатие и отпускание)

↓

Выборка приложением (**`GetMessage()`**)

↓

После трансляции (**`TranslateMessage()`**) – «вторичные» сообщения (ввод символов)

↓

Выборка приложением «вторичных» сообщений

↓

Перенаправление сообщений на обработку (**`DispatchMessage()`**)

↓

«Прикладные» обработчики сообщений в приложении

Сообщения идентифицируются их кодом (типом). Коды сообщений описываются символическими константами, имена которых группируются по префиксам. Большинство префиксов соответствует сообщения определенной категории оконных элементов управления.

Например:

Префикс	Расшифровка	Группа сообщений
WM_**	Window Message	«Общие» (не вошедшие в другие группы) сообщения
BM_**	Button Message	Сообщения-«команды» к элементу управления Button («кнопка»)
BN_**	Button Notify	Извещения от с элемента управления Button
CDM_**	Common Dialog box Message	Сообщения диалогового окна

Префикс	Расшифровка	Группа сообщений
LVM_**	List-View control Message	Сообщения элемента управления List View
PBM_**	Progress-Bar Mesage	Сообщения элемента управления Progress Bar
STM_**	Static control Message	Сообщения элемента управления Static
TCM_**	Tab Control Message	Сообщения элемента управления Tab Control («окно с закладками»)
TVM_**	Tree-View Message	Сообщения элемента управления Tree View
UDM_**	Up-Down Control Message	Сообщения элемента управления Up-Down (кнопка «вверх-вниз»)

и т.д.

Примеры сообщений:

Сообщение	Код	Назначение
WM_CLOSE	**0x0010**	Требование закрыть окно, извещение о попытке закрытия окна
WM_DESTROY	**0x0002**	Извещение о завершившемся «разрушении» (удалении) окна
WM_QUIT	**0x0012**	Прекращение работы цикла выборки сообщений, завершение приложения
WM_ACTIVATE	**0x0006**	Извещение об «активации» или «деактивации» окна
WM_PAINT	**0x000F**	Извещение об окончании перерисовки фона окна, необходимость возобновить изображение на фоне
WM_NCPAINT	**0x0085**	То же для «не-клиентской» области окна

Сообщение	Код	Назначение
WM_TIMER	**0x0113**	Извещение о событии таймера
WM_COMMAND	**0x0111**	«Команда» интерфейса пользователя – событие элемента управления, меню или «горячей клавиши»
WM_SYSCOMMAND	**0x0112**	«Команда» интерфейса пользователя – «системных» меню и кнопок
WM_MOVE	**0x0003**	Извещение о перемещении окна
WM_SIZE	**0x0005**	Извещение об изменении размеров окна
WM_WINDOWPOS-CHANGED	**0x0047**	Извещение об изменении позиции и/или размера и/или порядка в списке
WM_KEYDOWN **WM_KEYUP**	**0x0100** **0x0101**	Извещение о нажатии/отпускании клавиши клавиатуры

Сообщение	Код	Назначение
WM_CHAR	**0x0102**	Событие ввода – введенный с клавиатуры символ
WM_MOUSEMOVE	**0x0200**	Извещение о перемещении курсора «мыши»
WM_LBUTTONDOWN **WM_LBUTTONUP**	**0x0201** **0x0202**	Извещение о нажатии/отпускании левой кнопки «мыши»
WM_GETTEXT	**0x000D**	Запрос на получение текста (надписи, заголовка) от окна
WM_SETTEXT	**0x000C**	Запрос на изменение текста (надписи, заголовка) окна
BN_CLICKED		Извещение о нажатии элемента управления Button («кнопка»)
BM_CLICK		Запрос к элементу управления Button – эмуляция нажатия

Сообщение	Код	Назначение
BM_GETSTATE		Запрос к элементу управления Button – получение состояния
BM_SETSTATE		Запрос к элементу управления Button – изменить состояния (с пере-рисовкой)

и т.д.

Функции API и структуры данных

Структура сообщения struct MSG:

HWND *hwnd* – окно – адресат сообщения

UINT *message* – код (тип) сообщения

WPARAM *wParam*, **LPARAM *lParam*** – два дополнительных параметров (интерпретация зависит от типа сообщения)

DWORD *time* – время появления (отсылки) сообщения (в «тиках» системного таймера)

POINT *pt* – экранная позиция, с которой связано появление сообщения

DWORD *lPrivate* – служебные данные (?)

Функции для работы с сообщениями

Получение и проверка наличия сообщения в очереди:

```
BOOL GetMessage(
 LPMSG lpMsg,
 HWND hWnd,
 UINT wMsgFilterMin, UINT wMsgFilterMax
);
BOOL PeekMessage(
 LPMSG lpMsg,
 HWND hWnd,
 UINT wMsgFilterMin, UINT wMsgFilterMax,
 UINT wRemoveMsg
);
```

Постановка сообщения в очередь:

```
BOOL PostMessage(
 HWND   hWnd,
 UINT   Msg,
 WPARAM wParam, LPARAM lParam
);

BOOL PostThreadMessage(
 DWORD  idThread,
 UINT   Msg,
 WPARAM wParam, LPARAM lParam
);
```

Прямая передача сообщения в обработчик:

```
LRESULT SendMessage(
 HWND hWnd, UINT Msg,
 WPARAM wParam, LPARAM lParam
);

BOOL SendNotifyMessage(
 HWND hWnd, UINT Msg,
 WPARAM wParam, LPARAM lParam
);

BOOL SendMessageCallback(
 HWND hWnd, UINT Msg,
 WPARAM wParam, LPARAM lParam,
 SENDASYNCPROC lpResultCallBack,
 ULONG_PTR dwData
);
```

```
LRESULT SendMessageTimeout(
 HWND hWnd, UINT Msg,
 WPARAM wParam, LPARAM lParam,
 UINT fuFlags,
 UINT uTimeout,
 PDWORD_PTR lpdwResult
);
```

Оконная процедура (функция – обработчик сообщений)

```
LRESULT CALLBACK MyWndProc(
 HWND hWnd, UINT uMsg,
 WPARAM wParam, LPARAM lParam
)
{
 switch (hWnd) {
   ...
   ... switch (uMsg) {
         ...
       }
   ...
 }
 return result;
}
```

3.3 Механизм WinHook

Предоставляемый системой механизм перехвата сообщений и воздействия на их обработку.

Hook – системный объект, связываемый с потоком.

Идентификация – Handle **`HHOOK`**

Результат срабатывания Hook – вызов процедуры-обработчика (тип **`HOOKPROC`**):

```
LRESULT CALLBACK MyHookProc(int code, WPARAM
wParam, LPARAM lParam)
```

Параметры ***`wParam`*** и ***`lParam`*** – определяются типом Hook (типом перехваченного события) и не равны (не обязательно равны) одноименным параметрам перехваченного сообщения

Параметр code – обычно показывает, должен ли данный Hook выполнять обработку.

Перехватывать с помощью Hook можно:
– сообщения текущего процесса (обработчик в коде процесса)
– сообщения другого процесса (обработчик в отдельном модуле DLL)
– все сообщения системы

Типы перехватываемых с помощью Hook событий (примеры):
`WH_CALLWNDPROC` и **`WH_CALLWNDPROCRET`** – вызов оконной процедуры и ее завершение
`WH_GETMESSAGE` – обращения к сообщениям в очереди
`WH_FOREGROUNDIDLE` – переход текущего активного (foreground) потока приложения в состояние простоя (idle)
`WH_KEYBOARD` и **`WH_KEYBOARD_LL`** – сообщения клавиатуры («обычные» и «low-level»)
`WH_MOUSE` и **`WH_MOUSE_LL`** – сообщения клавиатуры («обычные» и «low-level»)
и т.д.

Функции API для работы с WinHook

Установка (включение) Hook:

```
HHOOK SetWindowsHookEx(
 int idHookType, // тип Hook
 HOOKPROC lpfnHookProc, // указатель процедуры Hook
 HINSTANCE hModule, // описатель модуля (DLL),
содержащего процедуру Hook
 DWORD dwThreadId //идентификатор потока, с которым
связан Hook
);
```

Деинсталляция Hook (отключение от цепочки обработчиков и удаление системного объекта)

```
BOOL UnhookWindowsHookEx(
 HHOOK hCurrentHook
)
```

Явный вызов следующего обработчика в цепочке

```
LRESULT CallNextHookEx(
 HHOOK hCurrentHook,
 int nHookCode,
 WPARAM wParam,
 LPARAM lParam
)
```

3.4 Оконный интерфейс Windows. Окна Windows

Аспекты окон как объектов в Win:
– элемент интерфейса пользователя (GUI)
– участник внутри- и межпрограммного интерфейса – адресат сообщений WinMessage

Иерархия окон

Отношение «***владения***» (***ownership***) – окно-***владелец*** (***owner*** window)

Иерархия:
Desktop – Top-level windows – прочие подчиненные окна

Отношение «***родительское – дочернее***» («***parent – child***»)

Например, все ***элементы управления*** («***controls***») – child-окна по отношению к окну-форме, на которой они размещены.

Каждое child-окно получает идентификатор (ID_Child), уникальный в рамках конкретного приложения, но никак не связанный с идентификаторами child-окон других приложений. Идентификатор служит для передачи parent-окну сообщений о событиях child-окон.

Сообщение **`WM_COMAND`**.

Альтернативные взгляды на сочетаемость «владения» и «родства»:
– окно-родитель является также и владельцем
– для конкретного подчиненного окна вышестоящее окно может быть или только родителем, или только владельцем, но не и тем и другим одновременно

(А что по этому поводу говорит первоисточник, т.е. MS Docs?)

Объекты, сущности и функции API для работы с окнами (основные)

Объект – ***окно*** (***Window object***)

Идентификация – дескриптор (handle) **`HWND`**.

Класс окна, оконный класс (***Window class***) – сущность (не системный объект!), описывающая ряд «родовых» характеристик окон, имеющих сходство во внешнем виде и поведении.

Типичный пример использования классов – стандартные элементы управления, имеющие схожее поведение и внешний вид, и выполняющих в разных приложениях схожие функции.

Идентификация класса:
– числовой идентификатор – тип ATOM
– имя – символьная строка

(«***Atom***» – фактически именованный уникальный числовой идентификатор. «Атомы» хранятся с системных таблицах и используются для идентификации некоторых сущностей, в т.ч. оконных классов.)

Обычно используются имена классов, например «**`BUTTON`**» для стандартных кнопок.

Виды оконных классов:

– ***системные*** (***system***) – глобальные, зарегистрированы системой, описаны в системных библиотеках (DLL) и доступны всем приложениям

– ***прикладные глобальные*** (***application global***) – зарегистрированы отдельными приложениями, но сделаны доступными для остальных, описаны в соответствующих «прикладных» библиотеках DLL, доступных в файловой системе

– ***прикладные локальные*** (***application local***) – зарегистрированы отдельными приложениями для собственного использования, описаны, как правило, в самом приложении

Описание класса – структуры **WNDCLASS** и **WNDCLASSEX**.

В числе прочего, структура содержит указатель на оконную процедуру, которая будет обрабатывать сообщения для всех окон этого класса.

Регистрация класса:

```
ATOM RegisterClass( const WNDCLASS *pwc);
ATOM RegisterClassEx( const WNDCLASSEX *pwc);
```

Прекращение действия класса

```
BOOL UnregisterClass( LPCSTR lpClassName, HINSTANCE
hInstance);
```

Создание окон – задаются характеристики, уникальные для конкретного экземпляра окна:

```
void CreateWindow(
    lpClassName, lpWindowName,
    dwStyle, x, y, nWidth, nHeight,
    hWndParent,
    hMenu,
    hInstance,
    lpParam
);
```

```
HWND CreateWindowEx(
  DWORD dwExStyle, LPCSTR lpClassName,
  LPCSTR lpWindowName,
  DWORD dwStyle, int x, int y, int nWidth, int
nHeight,
  HWND hWndParent,
  HMENU hMenu,
  HINSTANCE hInstance,
  LPVOID lpParam
);
```