Особенности каждого конкретного взаимодействия между двумя или более параллельными процессами определяются задачей синхронизации. Количество различных задач синхронизации неограниченно. Однако некоторые из них являются типичными. К ним относятся: *взаимное исключение*, *производители-потребители*, *читатели-писатели*, *обедающие философы* и т.д. Большинство задач в реальных ОС по согласованию параллельных процессов можно решить либо с помощью этих типовых задач, либо с помощью их модификаций.

Рассмотренные способы синхронизации, основанные на глобальных переменных процесса, обладают существенным недостатком – они не подходят для синхронизации потоков различных процессов. В таких случаях ОС должна предоставлять потокам системные объекты синхронизации, которые были бы видны для всех потоков, даже если они принадлежат разным процессам и работают в разных адресных пространствах.

Примерами таких синхронизирующих объектов являются *системные семафоры, мьютексы, события, таймеры* и др. Набор таких объектов определяется конкретной ОС. Чтобы разные процессы могли разделять синхронизирующие объекты, используются различные методы. Некоторые ОС возвращают указатель на объект. Этот указатель может быть доступен всем родственным процессам, наследующим характеристики общего родительского процесса. В других ОС процессы в запросах на создание объектов синхронизации указывают имена, которые должны им быть присвоены. Далее эти имена используются различными процессами для манипуляций объектами синхронизации. В этом случае работа с синхронизирующими объектами подобна работе с файлами. Их можно создавать, открывать, закрывать, уничтожать.

Ко **второму** классу объектов синхронизации относится ожидающий таймер (waitable timer). К **третьему** классу объектов синхронизации относятся объекты, которые переходят в сигнальное состояние по завершении своей работы или при получении некоторого сообщения. Примерами таких объектов синхронизации являются потоки и процессы. Пока эти объекты выполняются, они находятся в несигнальном состоянии. Если выполнение этих объектов заканчивается, то они переходят в сигнальное состояние.

Теперь перейдем к функциям ожидания. **Функции ожидания** в Windows это такие функции, параметрами которых являются объекты синхронизации. Эти функции обычно используются для блокировки потоков, которая выполняется следующим образом. Если дескриптор объекта синхронизации является параметром функции ожидания, а сам объект синхронизации находится в несигнальном состоянии, то поток, вызвавший эту функцию ожидания, блокируется до перехода этого объекта синхронизации в сигнальное состояние. Сейчас мы будем использовать только две функции ожидания WaitForSingleObject иWaitForMultipleObject.

Для ожидания перехода в сигнальное состояние одного объекта синхронизации используется функция WaitForSingleObject, которая имеет следующий прототип:

```
DWORD WaitForSingleObject(
  HANDLE hHandle,  // дескриптор объекта
  DWORD dwMilliseconds  // интервал ожидания в миллисекундах
);
```

Функция WaitForSingleObject в течение интервала времени, равного значению параметра dwMilliseconds, ждет пока объект синхронизации с дескриптором hHandle перейдет в сигнальное состояние. Если значениепараметра dwMilliseconds равно нулю, то функция только проверяет состояние объекта. Если же значение параметра dwMilliseconds равно INFINITE, то функция ждет перехода объекта синхронизации в сигнальное состояние бесконечно долго.

В случае удачного завершения функция WaitForSingleObject возвращает одно из следующих значений:

```
WAIT_OBJECT_0
WAIT_ABANDONED
WAIT_TIMEOUT
```

Значение WAIT_OBJECT_0 означает, что объект синхронизации находился или перешел в сигнальное состояние. Значение WAIT_ABANDONED означает, что объектом

синхронизации является мьютекс, который не был освобожден потоком, завершившим свое исполнение. После завершения потока этот мьютекс освободился системой и перешел в сигнальное состояние. Такой мьютекс иногда называется забытым мьютексом (abandoned mutex). Значение WAIT_TIMEOUT означает, что время ожидания истекло, а объект синхронизации не перешел в сигнальное состояние. В случае неудачи функция WaitForSingleObject возвращает значение WAIT_FAILED.

Событием называется оповещение о некотором выполненном действии. В программировании события используются для оповещения одного потока о том, что другой поток выполнил некоторое действие. Сама же и задача оповещения одного потока о некотором действии, которое совершил другой поток называется задачейусловной синхронизации или иногда задачей оповещения.

В операционных системах Windows события описываются объектами ядра Events. При этом различают два типа событий:

− события с ручным сбросом;

− события с автоматическим сбросом.

Различие между этими типами событий заключается в том, что событие с ручным сбросом можно перевести в несигнальное состояние только посредством вызова функции ResetEvent, а событие с автоматическим сбросом переходит в несигнальное состояние как при помощи функции ResetEvent, так и при помощи функции ожидания. При этом отметим, что если события с автоматическим сбросом ждут несколько потоков, используя функцию WaitForSingleObject, то из состояния ожидания освобождается только один из этих потоков.

Создаются события вызовом функции CreateEvent, которая имеет следующий прототип:

```
HANDLE CreateEvent(
  LPSECURITY_ATTRIBUTES  lpSecurityAttributes, // атрибуты защиты
  BOOL    bManualReset,  // тип события
  BOOL    bInitialState,  // начальное состояние события
    LPCTSTR      lpName      // имя события
);
```

Как и обычно, пока значение параметра lpSecurityAttributes будем устанавливать в NULL. Основную смысловую нагрузку в этой функции несут второй и третий параметры. Если значение параметра bManualReset равно TRUE, то создается событие с ручным сбросом, в противном случае – с автоматическим сбросом. Если значение параметра bInitialState равно TRUE, то начальное состояние события является сигнальным, в противном случае – несигнальным. Парметр lpName задает имя события, которое позволяет обращаться к нему из потоков, выполняющихся в разных процессах. Этот параметр может быть равен NULL, тогда создается безымянное событие. В случае удачного завершения функция CreateEvent возвращает дескриптор события, а в случае неудачи – значение NULL. Если событие с заданным именем уже существует, то функция CreateEvent возвращает дескриптор этого события, а функция GetLastError, вызванная после функции CreateEvent вернет значение ERROR_ALREADY_EXISTS.