Системное программирование. Процессы и потоки в Windows API

Процессы и потоки в Windows API

- Процесс (задача) – исполняемое компьютером приложение вместе со всеми ресурсами, которые требуются для его исполнения
- Контекст процесса – все ресурсы процесса
- Ресурсы процесса:
  - Адресное пространство – виртуальная память, выделенная процессу для запуска программ
  - Один или несколько потоков
  - Дескрипторы, кучи, стек, сегменты данных и кода и т.п.
- Поток управления (thread) – последовательность выполнения инструкций программы процессором
- Поток – основная единица выполнения в Windows

Управление потоками в Windows API

- Программы могут быть:
  - Однопоточные
  - Многопоточные
- Потоки в приложении:
  - Главный (первичный) поток – функция `main` или `WinMain`
  - Другие потоки создаются явно – функции специального вида (callback, функции обратного вызова)
- Функция для создания потока имеет вид:

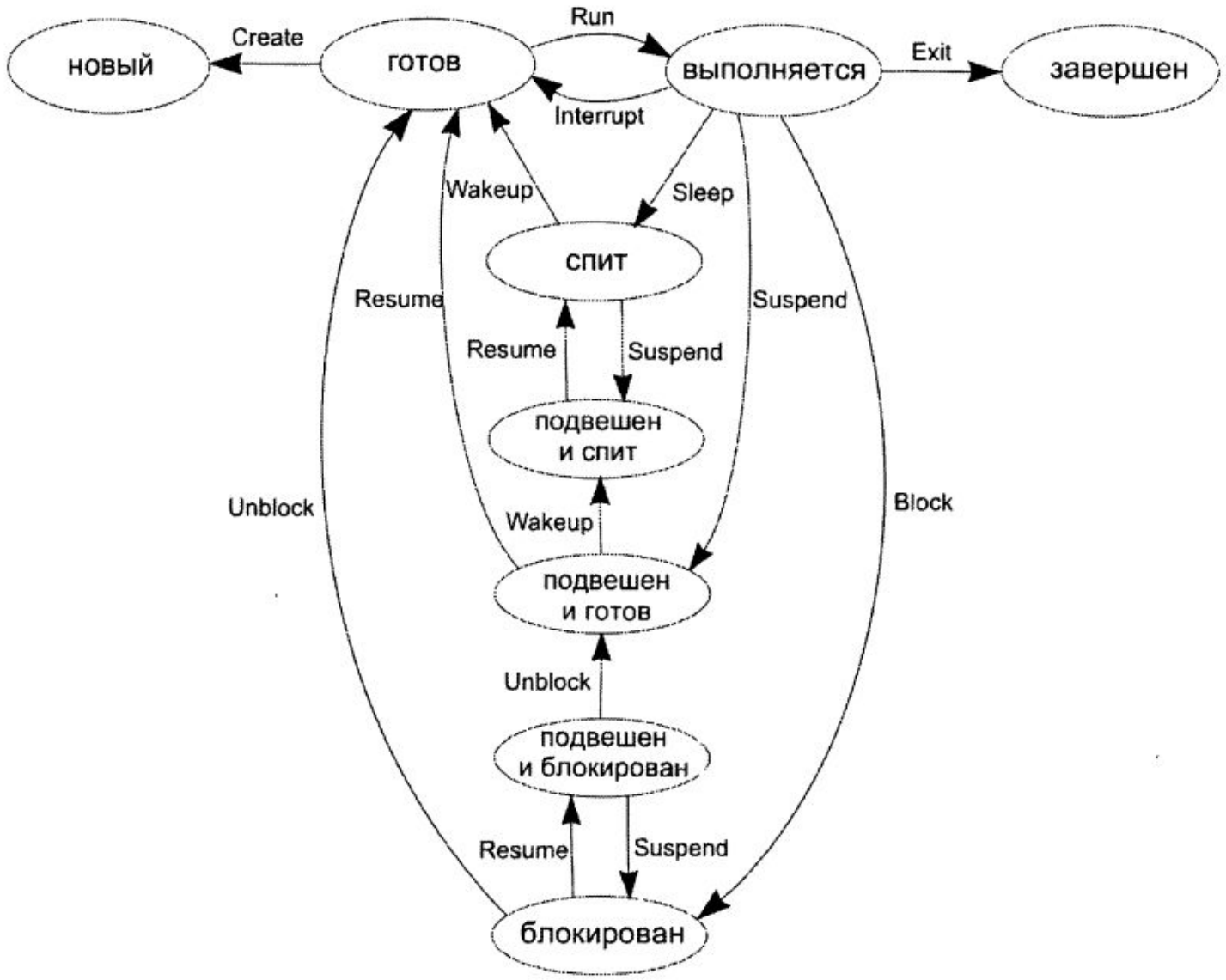
```
DWORD WINAPI <имя>(LPVOID lpParameters)
```

- Требования к функции для создания потоков:
  - *Реентерабельность* – не использует неблокированные глобальные и статические переменные, не возвращает указатели на статические данные, не изменяет свой код
  - *Безопасность для потока* – блокирует доступ к

Управление потоками в Windows API

- Состояния потока:
  - Блокирован – программа не готова, процессор не выделен
  - Готов – программа готова, процессор не выделен
  - Выполняется – программа готова, процессор выделен
  - Новый — поток еще не начавший свою работу
  - Завершен — поток, завершивший свою работу
  - Приостановлен (подвешен) — сам поток или другой поток приостановил его выполнение
  - Спит — поток приостановил свое исполнение на некоторый интервал времени
- Переход между состояниями осуществляется:
  - Самим или другими потоками
  - Операционной системой

Управление потоками в Windows API



Управление потоками в Windows API

- Распределением квантов времени процессора в ОС занимается менеджер потоков:
  - Сохраняет контекст прерываемого потока
  - Восстанавливает контекст выполняемого потока на момент его прерывания
  - Передает управление запускаемому потоку
- Создание потоков:
  - Функция `CreateThread`
  - Макрокоманда `_beginthreadex` из `<process.h>`
- Завершение потока:
  - Функция `ExitThread` – вызывает сам поток
  - Функция `TerminateThread` – из других потоков
  - Возврат значения из функции потока через `return`
  - Макрокоманда `_endthreadex` из `<process.h>`

Управление потоками в Windows API

- Получение кода завершения потока:
  - Функция `GetExitCodeThread`
- Приостановка и возобновление потоков:
  - Функция `SuspendThread` – приостановить поток
  - Функция `ResumeThread` – возобновить выполнение потока
  - Функция `Sleep` – задержать исполнение данного потока
- Получение псевдодескриптора текущего потока:
  - Функция `GetCurrentThread`
- Обработка ошибок
  - Функция `GetLastError` – поддерживается отдельно для каждого потока
  - Функция `SetLastError` – установить код ошибки

Управление потоками в Windows API

- Использование стандартных библиотек Си:
  - Стандартные библиотеки Си не поддерживают многопоточные приложения – не все функции этих библиотек являются реентерабельными
  - Microsoft C предлагает реализацию стандартных библиотек (CRT), поддерживающих многопоточность: LIBCMT.LIB и MSVCRT.LIB
  - Для включения многопоточной версии стандартных библиотек C необходимо (для Visual Studio):
    - В свойствах проекта указать Runtime Library: Multi-threaded или добавить директиву `#define _MT` перед включением библиотек с помощью `#include`
    - Подключить `<process.h>`
    - Вместо функций `CreateThread` и `ExitThread` использовать `_beginthreadex` и `_endthreadex`
    - Допускается использование `return` для завершения потока

Управление потоками в Windows API

- Собственные (защищенные) области памяти для потоков:
  - Данные в стеке: данные, получаемые через параметр при создании потока, и автоматическая память
  - Локальные области хранения потоков (TLS):
    - Функция `TlsAlloc` – получить индекс в TLS
    - Функция `TlsFree` – освободить индекс в TLS
    - Функция `TlsGetValue` – получить значение в TLS
    - Функция `TlsSetValue` – установить значение в TLS
- Управление приоритетом потоков:
  - Функция `GetThreadPriority` – получить приоритет
  - Функция `SetThreadPriority` – установить приоритет
- Функция потока может передаваться при создании потока с приведением типа к `LPTHREAD_START_ROUTINE`

Управление процессами в Windows API

- Создание нового процесса:
  - Функция
```
BOOL CreateProcess(
LPCTSTR lpApplicationName, //имя исполняемого модуля
LPTSTR lpCommandLine, //командная строка
LPSECURITY_ATTRIBUTES lpsaProcess, //защита процесса
LPSECURITY_ATTRIBUTES lpsaThread, //защита потока
BOOL bInheritHandles, //наследование дескрипторов
DWORD dwCreationFlags, //флаги создания процесса
LPVOID lpEnvironment, //параметры среды окружения
LPCTSTR lpCurDir, //текущий каталог
LPSTARTUPINFO lpStartupInfo, //вид основного окна
LPPROCESS_INFORMATION lpProcInfo //дескрипторы и ID
);
```
  - Некоторые возможные значения `dwCreationFlags`:
    - `CREATE_SUSPENDED` – приостановленное состояние
    - `DETACHED_PROCESS` – новый процесс без консоли
    - `CREATE_NEW_CONSOLE` – своя консоль у нового процесса
    - `NORMAL_PRIORITY_CLASS` и др. – приоритет нового процесса

Управление процессами в Windows API

- Создание нового процесса:
  - Структура `lpStartupInfo` – информация об окне нового процесса и о дескрипторах стандартных устройств в/в:
    - `cb` – размер структуры; обычно `sizeof(STARTUPINFO)`
    - `lpTitle` – заголовок окна консоли (не для GUI-приложений)
    - `hStdInput` – дескриптор стандартного устройства ввода
    - `hStdOutput` – дескриптор стандартного устройства вывода
    - `hStdError` – дескриптор стандартного устройства ошибок
    - `dwFlags` – флаги, указывающие на реально используемые поля структуры (`STARTF_USESTDHANDLES` для использования дескрипторов стандартных устройств) и др.
    - Получить структуру из текущего процесса: `GetStartupInfo`
  - Структура `lpProcInfo` – информация о дескрипторах и идентификаторах нового процесса и потока:

Управление процессами в Windows API

- Создание нового процесса:
  - Имя исполняемого файла процесса указывается через:
    - `lpApplicationName` – полный путь к приложению
    - `lpCommandLine` – через командную строку (могут указываться параметры при запуске)
  - Если процесс запускается через командную строку (относительный путь), то исполняемый файл ищется в:
    1. Каталоге модуля текущего процесса
    2. Текущем каталоге
    3. Системном каталоге Windows
    4. Каталоге Windows
    5. Каталогах, перечисленных в переменной окружения PATH
  - Управление блоком параметров окружения (параметр `lpEnvironment`) – функции:
    - `GetEnvironmentVariable` – получить переменные

Управление процессами в Windows API

- Наследуемые дескрипторы объектов:
  - Процессы, запускаемый из другого – *процесс-потомок (дочерний)* для исходного родительского процесса
  - Процессы-потомки могут наследовать дескрипторы объектов родительского процесса, кроме:
    - Дескрипторов виртуальной памяти
    - Дескрипторов DLL
  - Для наследования дескрипторов необходимо:
    - Сделать нужные дескрипторы наследуемыми
    - Установить `bInheritHandles = TRUE` при создании процесса
    - Передать наследуемые дескрипторы дочернему процессу
  - Сделать дескриптор наследуемым можно:
    - Через атрибуты безопасности при создании объекта
    - С помощью функции `SetHandleInformation`

Управление процессами в Windows API

- Наследуемые дескрипторы объектов:
  - Дублирование дескрипторов используется когда необходимо изменить свойства дескриптора, управляющие доступом к объекту (помимо свойства наследования)
  - Передача дескриптора наследуемому процессу:
    - Через параметр командной строки
    - Через дескрипторы стандартных устройств ввода/вывода
    - Через средства межпроцессного взаимодействия (IPC)
- Функции для работы с процессами:
  - Завершение процессов:
    - `ExitProcess` – завершение текущего процесса
    - `TerminateProcess` – завершение процесса из другого процесса
    - `GetExitCodeProcess` – получение кода завершения

Управление процессами в Windows API

- Функции для работы с процессами:
  - Получение информации о себе:
    - `GetCurrentProcess` – псевдодескриптор процесса
    - `GetCurrentProcessId` – идентификатор процесса
    - Получение нормального дескриптора текущего процесса – либо через дублирование псевдодескриптора, либо с помощью функции `OpenProcess`
  - Получение количества открытых дескрипторов текущего процесса:
    - Функция `GetProcessHandleCount`
  - Изменение приоритетов процесса и его потоков:
    - `GetPriorityClass` – получение приоритета процесса
    - `SetPriorityClass` – установка приоритета процесса
    - `SetProcessPriorityBoost` – установка режима динамического изменения базового приоритета потоков
    - `GetProcessPriorityBoost` – получение режима (да/нет)

Блокировка потоков в Windows API

- Функции блокировки выполнения потоков:
  - Перечисленные ниже функции ожидают перехода объекта синхронизации в сигнальное состояние
  - Для процессов и потоков сигнальное состояние наступает после завершения процесса/потока
  - Текущий поток (где вызывается такая функция) блокируется (приостанавливается) до тех пор, пока:
    - Объект синхронизации не перейдет в сигнальное состояние
    - Не истечет время ожидания (таймаут)
  - `DWORD WaitForSingleObject(`
    `HANDLE hObject, //дескриптор объекта синхронизации`
    `DWORD dwMilliseconds);`
  - Некоторые возможные результаты:
    - `WAIT_OBJECT_0` – объект перешел в сигнальное состояние
    - `WAIT_TIMEOUT` – наступил таймаут (время ожидания истекло)

Блокировка потоков в Windows API

- Функции блокировки выполнения потоков:
  - `DWORD WaitForMultipleObjects(`
    `DWORD nCount, //количество объектов синхронизации`
    `CONST HANDLE *lpHandles, //массив дескрипторов`
    `BOOL fWaitAll, //режим ожидания`
    `DWORD dwMilliseconds //таймаут (интервал ожидания)`
    `);`
  - Некоторые возможные результаты:
    - От `WAIT_OBJECT_0` до `(WAIT_OBJECT_0 + nCount - 1)` – интерпретация зависит от значения `fWaitAll`
    - `WAIT_TIMEOUT` – наступил таймаут (время ожидания истекло)
    - `WAIT_FAILED` – ошибка
  - Возможные значение для параметра `dwMilliseconds`:
    - Таймаут в миллисекундах
    - `INFINITE` – бесконечное ожидание

Управление потоками в Windows API

- ***Задание 6***:
  - Самостоятельно изучить функции Windows API:
    - `CreateThread`
    - `ExitThread`
  - Написать программу для параллельного поиска букв «a» и «b» в строке (подсчитать их количество):
    1. Ввести строку (до создания второго потока)
    2. Создать второй поток для поиска букв «a»
    3. В главном потоке выполнить поиск букв «b»
    4. Подождать завершение второго потока
    5. Закрыть дескриптор второго потока
  - Строку объявить как `TCHAR * volatile str` (для Си) или описать в `main` и передать второму потоку как параметр
  - Включить многопоточную реализацию CRT (для Си)
  - Можно использовать любую среду разработки и любой тип приложения (GUI или консольное)

Управление потоками в Windows API

- ***Задание 7***:
  - Самостоятельно изучить функции Windows API:
    - `SuspendThread`
    - `ResumeThread`
    - `Sleep`
    - `TerminateThread`
  - Написать программу для управления вторым потоком из главного потока:
    1. Создать второй поток, который через каждую секунду должен выводить следующее натуральное число
    2. Завершается второй поток либо из главного, либо когда выведутся 20 чисел. При завершении вывести сообщение
    3. В главном потоке вывести меню (в цикле) для управления вторым потоком (приостановить, возобновить, завершить)
  - Включить многопоточную реализацию CRT (для Си)
  - Можно использовать любую среду разработки и любой тип приложения (GUI или консольное)

Управление процессами в Windows API

- ***Задание 8***:
  - Самостоятельно изучить функции Windows API:
    - `GetCurrentProcess`
    - `DuplicateHandle`
    - `TerminateProcess`
    - `GetExitCodeThread`
  - Написать программу для завершения потока первого процесса во втором процессе:
    1. Создать новый поток (выводит числа через 1 сек. на консоль)
    2. Продублировать дескриптор нового потока с режимом доступа `THREAD_TERMINATE` и с возможностью наследования
    3. Запустить второй процесс со своей консолью, передать ему дублированный дескриптор через командную строку
    4. Второй процесс выводит меню, где предлагает завершить поток первого процесса или выйти из программы
    5. Дождаться завершения второго процесса в первом, завершить второй поток (если нужно), закрыть

Блокировка потоков в Windows API

- ***Задание 9***:
  - Написать программу для ожидания завершения любого из двух потоков:
    1. Создать два новых потока. Первый поток выводит 5 положительных чисел, второй – 5 отрицательных. После вывода каждого числа каждый поток засыпает на случайное количество миллисекунд (не более 2000).
    2. В главном потоке дождаться завершения любого из двух новых потоков
    3. После завершения ожидания вывести номер завершившегося потока и завершить второй поток, если он еще не завершился
    4. Закрыть дескрипторы потоков
  - Можно использовать любую среду разработки и любой тип приложения (GUI или консольное)

Спасибо за внимание.