Лабораторная работа 3
Работа с процессами и потоками в Windows

Цель работы: овладение навыками работы с процессами и потоками в Windows с помощью WIN32API.

1 Процессы

Процессом обычно называют экземпляр выполняемой программы. Хотя на первый взгляд кажется, что программа и процесс понятия практически одинаковые, они фундаментально отличаются друг от друга. Программа представляет собой статический набор команд, а процесс – это набор ресурсов и данных, использующихся при выполнении программы. Процесс в Windows состоит из следующих компонентов:

1) Структура данных, содержащая всю информацию о процессе, в том числе список открытых дескрипторов различных системных ресурсов, уникальный идентификатор процесса, различную статистическую информацию и т.д.;
2) Адресное пространство – диапазон адресов виртуальной памяти, которым может пользоваться процесс;
3) Исполняемая программа и данные, проецируемые на виртуальное адресное пространство процесса.

2 Потоки

Процессы инертны. Отвечают же за исполнение кода, содержащегося в адресном пространстве процесса, потоки. Поток (*thread*) – сущность внутри процесса, получающая процессорное время для выполнения. В каждом процессе есть минимум один поток. Этот первичный поток создается системой автоматически при создании процесса. Далее этот поток может породить другие потоки, те в свою очередь новые и т.д. Таким образом, один процесс

может владеть несколькими потоками, и тогда они одновременно исполняют код в адресном пространстве процесса. Каждый поток имеет:

1) Уникальный идентификатор потока;

2) Содержимое набора регистров процессора, отражающих состояние процессора;

3) Два стека, один из которых используется потоком при выполнении в режиме ядра, а другой – в пользовательском режиме;

4) Закрытую область памяти, называемую локальной памятью потока (*thread local storage*, *TLS*) и используемую подсистемами, run-time библиотеками и DLL.

Планирование потоков

Чтобы все потоки работали, операционная система отводит каждому из них определенное процессорное время. Тем самым создается иллюзия одновременного выполнения потоков (разумеется, для многопроцессорных компьютеров возможен истинный параллелизм). В Windows реализована система вытесняющего планирования на основе приоритетов, в которой всегда выполняется поток с наибольшим приоритетом, готовый к выполнению. Выбранный для выполнения поток работает в течение некоторого периода, называемого квантом. Квант определяет, сколько времени будет выполняться поток, пока операционная система не прервет его. По окончании кванта операционная система проверяет, готов ли к выполнению другой поток с таким же (или большим) уровнем приоритета. Если таких потоков не оказалось, текущему потоку выделяется еще один квант. Однако поток может не полностью использовать свой квант. Как только другой поток с более высоким приоритетом готов к выполнению, текущий поток вытесняется, даже если его квант еще не истек.

Квант не измеряется, в каких бы то ни было единицах времени, а выражается целым числом. Для каждого потока хранится текущее значение его кванта. Когда потоку выделяется квант процессорного времени, это

значит, что его квант устанавливается в начальное значение. Оно зависит от операционной системы.

Всякий раз, когда возникает прерывание от таймера, из кванта потока вычитается 3, и так до тех пор, пока он не достигнет нуля. Частота срабатывания таймера зависит от аппаратной платформы.

В любом случае операционная система должна определить, какой поток выполнять следующим. Выбрав новый поток, операционная система переключает контекст. Эта операция заключается в сохранении параметров выполняемого потока (регистры процессора, указатели на стек ядра и пользовательский стек, указатель на адресное пространство, в котором выполняется поток и др.), и загрузке аналогичных параметров для другого потока, после чего начинается выполнение нового потока.

Планирование в Windows осуществляется на уровне потоков, а не процессов. Это кажется понятным, так как сами процессы не выполняются, а лишь предоставляют ресурсы и контекст для выполнения потоков. Поэтому при планировании потоков, система не обращает внимания на то, какому процессу они принадлежат. Например, если процесс А имеет 10 готовых к выполнению потоков, а процесс Б – два, и все 12 потоков имеют одинаковый приоритет, каждый из потоков получит 1/12 процессорного времени.

Приоритеты

Процессорное время выделяется потокам в соответствии с их уровнем приоритета. Потоку с более низким приоритетом не выделяется время, если на него претендует поток с более высоким уровнем приоритета. Более того, процесс с более низким приоритетом прерывается до истечения кванта времени, если на процессор претендует поток с более высоким уровнем приоритета. Необходимо помнить, что в среде Windows основная «работа» потока состоит в ожидании события и реагировании на него. Это дает шанс на исполнения потокам с низким уровнем приоритета.

В Windows существует 32 уровня приоритета, от 0 до 31. Они группируются так: 31 – 16 уровни реального времени; 15 – 1 динамические уровни; 0 – системный уровень, зарезервированный для потока обнуления страниц (zero-page thread).

1) Уровень приоритета каждого потока состоит из трех составляющих

2) класс приоритета процесса (простаивающий, нормальный, высокий, реального времени);

3) уровень приоритета потока внутри класса приоритета процесса (нижний, ниже нормального, нормальный, выше нормального, высший);

4) динамически установленный уровень приоритета.

Класс приоритета процесса

При создании процесса, ему назначается один из шести классов приоритетов, показанных в таблице 8.

Таблица 8 – Значения констант классов приоритетов в Windows NT.

Класс	Флаг в функции Create Process	Числовой уровень
Realtime реального времени	`REALTIME_PRIORITY_CLASS`	24
High высокий	`HIGH_PRIORITY_CLASS`	13
Above normal[1)] выше нормального	`ABOVE_NORMAL_PRIORITY_CLASS`	10
Normal нормальный	`NORMAL_PRIORITY_CLASS`	8 (7–9)
Below normal[1)] Ниже нормального	`BELOW_NORMAL_PRIORITY_CLASS`	6
Idle простаивающий	`IDLE_PRIORITY_CLASS`	4

Уровень `Idle` назначается процессу, который ничего не должен делать в случае активности других процессов (например, хранитель экрана).

[1] Уровни приоритета Above normal и Below normal появились начиная с версии Windows 2000.

Процессам, запускаемым пользователем, присваивается нормальный уровень. Пользователь может запустить несколько процессов. Тому процессу, с которым пользователь непосредственно работает (а это может быть только один процесс) уровень приоритета поднимается на две единицы (7+2 = 9). Это делает общение с прикладной программой более “комфортабельным”. Высокий класс приоритета назначается некоторым системным процессам, которые простаивают до возникновения определенных событий и, поэтому, не мешают остальным процессам. Только в особых случаях процесс может относиться к классу Realtime.

Уровень приоритета потока внутри класса приоритетов

Таблица 9 – Уровни приоритетов потока

Идентификатор	**Уровень приоритета потока**
THREAD_PRIORITY_LOWEST	Самый низший. На 2 ниже уровня класса
THREAD_PRIORITY_BELOW_NORMAL	Ниже нормального. На 1 ниже уровня класса
THREAD_PRIORITY_NORMAL	Нормальный. Равен уровню класса
THREAD_PRIORITY_ABOVE_NORMAL	Выше нормального. На 1 выше уровня класса
THREAD_PRIORITY_HIGHEST	Наивысший. На 2 выше уровня класса
THREAD_PRIORITY_IDLE	Пассивный. Равен 1 для процессов класса IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, и HIGH_PRIORITY_CLASS и равен 16 для REALTIME_PRIORITY_CLASS
THREAD_PRIORITY_TIME_CRITICAL	Критичный по времени. Равен 15 для процессов класса IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS, и HIGH_PRIORITY_CLASS, и равен 31 для REALTIME_PRIORITY_CLASS

Динамическое изменение уровня приоритета потока

Класс приоритета процесса и уровень приоритета потока внутри класса определяют базовый уровень приоритета потока. Этот уровень может динамически изменяться системой, а именно, повышаться на 2 единицы в ответ на поступление сообщений в очередь потока с последующим понижением до базового уровня по истечении определенного промежутка времени. Это правило действует только для потоков до 15 уровня.

Таким образом, все 32 уровня приоритета можно свести в таблицу 10.

Таблица 10 – Уровни приоритета

Уровень	**Класс приоритета процесса**	**Класс приоритета процесса**					
		Idle	**Below Normal**	**Normal**	**Above Normal**	**High**	**Realtime**
31							Time critical
30							
29							
28							
27							
26							Highest
25							Above normal
24	Realtime						Normal
23							Below normal
22							Lowest
21							
20							
19							
18							
17							
16							Idle
15		Time critical	Time critical	Time critical	Time critical	Time critical	
15						Highest	
14						Above normal	
13	High					Normal	
12					Highest	Below normal	
11					Above normal	Lowest	
10	Above Normal			Highest	Normal		
9				Above normal	Below normal		
8	Normal		Highest	Normal	Lowest		

Таблица 10 – Уровни приоритета

7			Above normal	Below normal			
6	Below Normal	Highest	Normal	Lowest			
5		Above normal	Below normal				
4	Idle	Normal	Lowest				
3		Below normal					
2		Lowest					
1		Idle	Idle	Idle	Idle	Idle	
0							

Привязка к процессорам

Если операционная система выполняется на машине, где установлено более одного процессора, то по умолчанию, поток выполняется на любом доступном процессоре. Однако в некоторых случаях, набор процессоров, на которых поток может работать, может быть ограничен. Это явление называется привязкой к процессорам (processor affinity). Можно изменить привязку к процессорам на программном уровне, через Win32-функции планирования.

Память

Каждому процессу в Win32 доступно линейное 4-гигабайтное (2^{32} = 4 294 967 296) виртуальное адресное пространство. Обычно верхняя половина этого пространства резервируется за операционной системой, а вторая половина доступна процессу.

Виртуальное адресное пространство процесса доступно всем потокам этого процесса. Иными словами, все потоки одного процесса выполняются в едином адресном пространстве.

С другой стороны, механизм виртуальной памяти позволяет изолировать процессы друг от друга. Потоки одного процесса не могут ссылаться на адресное пространство другого процесса.

Виртуальная память может вовсе не соответствовать структуре физической памяти. Диспетчер памяти транслирует виртуальные адреса на физические, по которым реально хранятся данные. Поскольку далеко не всякий компьютер в состоянии выделить по 4 Гбайт физической памяти на каждый процесс, используется механизм подкачки (swapping). Когда оперативной памяти не хватает, операционная система перемещает часть содержимого памяти на диск, в файл (swap file или page file), освобождая, таким образом, физическую память для других процессов. Когда поток обращается к странице виртуальной памяти, записанной на диск, диспетчер виртуальной памяти загружает эту информацию с диска обратно в память.

3 Работа с процессами и потоками Win32API[2]

Windows API спроектирован для использования в языке Си для написания прикладных программ, предназначенных для работы под управлением операционной системы MS Windows. Работа через Windows API — это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ. Более низкий уровень доступа, необходимый только для драйверов устройств, в текущих версиях Windows предоставляется через Windows Driver Model.

Интерфейс прикладного программирования Windows (API) позволяет разрабатывать настольные и серверные приложения, которые успешно работают во всех версиях Windows, используя при этом преимущества функций и возможностей, уникальных для каждой версии.

Windows API можно использовать во всех настольных приложениях на базе Windows, и те же функции обычно поддерживаются в 32-разрядной и 64-

[2] Ранее эти функции назывались Win32®API. Название Windows API более точно отражает корни связь с 16-битной Windows и поддержку этих функций в 64-битной Windows. 64-битные версии Windows, использующие 64-битную версию Win32, находятся в использовании достаточно давно. Тем не менее, 64-битная версия Win32 все еще известна как Win32, так как она, по сути, использует идентичный интерфейс, и единственное отличие составляют указатели больших размеров.

разрядной версиях Windows. Различия в реализации элементов программирования зависят от возможностей базовой операционной системы. Эти различия отмечены в документации к API-функциям.

Создание процессов

Создание Win32 процесса осуществляется вызовом одной из таких функций, как CreateProcess, CreateProcessAsUser (для Windows NT) и CreateProcessWithLogonW (начиная с Windows 2000) и происходит в несколько этапов:

1) Открывается файл образа (exe), который будет выполняться в процессе. Если исполняемый файл не является Win32-приложением, то ищется образ поддержки (support image) для запуска этой программы. Например, если исполняется файл с расширением .bat, запускается cmd.exe и т.п.

В WinNT для отладки программ реализовано следующее. CreateProcess, найдя исполняемый Win32 файл, ищет в SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Option раздел с именем и расширением запускаемого файла, затем ищет в нем параметр Debugger, и если строка не пуста, запускает то, что в ней написано вместо данной программы.

2) Создается объект Win32 «процесс».

3) Создается первичный поток (стек, контекст и объект «поток»).

4) Подсистема Win32 уведомляется о создании нового процесса и потока.

5) Начинается выполнение первичного потока.

6) В контексте нового процесса и потока инициализируется адресное пространство (например, загружаются требуемые DLL-библиотеки) и начинается выполнение программы.

Завершение процессов

Процесс завершается если:

– Входная функция первичного потока возвратила управление.

– Один из потоков процесса вызвал функцию ExitProcess.

– Поток другого процесса вызвал функцию TerminateProcess.

Когда процесс завершается, все User- и GDI-объекты, созданные процессом, уничтожаются, объекты ядра закрываются (если их не использует другой процесс), адресное пространство процесса уничтожается.

Функции для работы с процессами

В Win32API определен ряд функций работы с процессами, которые показаны в Таблице 11.

Таблица 11 – Функции для работы с процессами

Название функции	**Выполняемое действие**
CreateProcess	Создает процесс
ExitProcess	Завершает процесс и все его потоки
GetCurrentProcess	Возвращает описатель (handle) текущего процесса
GetCurrentProcessId	Возвращает идентификатор текущего процесса
GetEnvironmentStrings	Возвращает строку переменных среды
GetEnvironmentVariable	Возвращает значение указанной переменной среды
GetProcessVersion	Сообщает версию Windows, для которой предназначен процесс
GetPriorityClass	Возвращает класс приоритета процесса
GetProcessTimes	Возвращает временные характеристики указанного процесса
GetStartupInfo	Возвращает параметры процесса, полученные им при создании
OpenProcess	Возвращает описатель (handle) указанного процесса
SetPriorityClass	Устанавливает класс приоритета процесса
TerminateProcess	Завершает указанный процесс

В данной работе подробно рассматриваются только некоторые функции, информацию по остальным функциям можно найти в справочном руководстве к Win32 API. Также подробное описание функций приведено в Win32 Programmer’s Reference[3].

[3] Официальный сайт корпорации Microsoft. — 2019 [Электронный ресурс]. — URL: https://docs.microsoft.com/en-us/windows/win32/api/ (дата обращения: 04.11.2019).

Функция CreateProcess()

Функция `CreateProcess()` создает новый процесс и его первичный поток. Новый процесс исполняет указанный исполняемый файл. Функция имеет следующий формат:

```
BOOL CreateProcess(LPCTSTR lpApplicationName,
LPTSTR lpCommandLine,
LPSECURITY_ATTRIBUTES lpProcessAttributes,
LPSECURITY_ATTRIBUTES lpThreadAttributes,
BOOL bInheritHandles,
DWORD dwCreationFlags,
LPVOID lpEnvironment,
LPCTSTR lpCurrentDirectory,
LPSTARTUPINFO lpStartupInfo,
LPPROCESS_INFORMATION lpProcessInformation
);
```

Параметр `lpApplicationName` – указатель на строку, содержащую имя исполняемой программы. Имя может быть абсолютным. Если оно не абсолютное, то поиск файла производится в текущем каталоге. Параметру может быть присвоено значение `NULL`. В этом случае в качестве имени файла выступает первая выделенная пробелами лексема из строки `lpCommandLine`.

Параметр `lpCommandLine` – указатель командной строки. Если параметр `lpApplicationName` имеет значение `NULL`, то имя исполняемого файла выделяется из `lpCommandLine`, а поиск исполняемого файла производится в соответствии с правилами, действующими в операционной системе.

Параметр `lpProcessAttributes` – указатель на структуру, описывающую параметры защиты процесса. Если параметру присвоено значение `NULL`, то устанавливаются атрибуты «по умолчанию».

Параметр `lpThreadAttributes` – указатель на структуру, описывающую параметры защиты первичного потока. Если параметру присвоено значение `NULL`, то устанавливаются атрибуты «по умолчанию».

Параметр `bInheritHandles` определяет, будет ли порожденный процесс наследовать описатели (handles) объектов родительского процесса. Например, если родительский процесс `A` уже до этого порождал процесс `B`, то он получил

описатель процесса B и может им манипулировать. Если теперь он порождает процесс C с параметром bInheritHandles равным TRUE, то и процесс C сможет работать с описателем процесса B.

Параметр dwCreationFlags определяет некоторые дополнительные условия создания процесса и его класс приоритета.

Параметр lpEnvironment – указатель на блок переменных среды порожденного процесса. Если этот параметр равен NULL, то порожденный процесс наследует среду родителя. Иначе он должен указывать на завершающийся нулем блок строк, каждая из которых завершается нулем;

Параметр lpCurrentDirectory – указатель на строку, содержащую полное имя текущего каталога порожденного процесса. Если этот параметр равен NULL, то порожденный процесс наследует каталог родителя;

Параметр lpStartupInfo – указатель на структуру STARTUPINFO, которая определяет параметры главного окна порожденного процесса;

Параметр lpProcessInformation – указатель на структуру PROCESSINFO, которая будет заполнена информацией о порожденном процессе после возврата из функции.

Создание потоков

Первичный поток создается автоматически при создании процесса. Остальные потоки создаются функциями CreateThread и CreateRemoteThread (только в Windows NT/2000/XP).

Завершение потоков

Поток завершается если:

– функция потока возвращает управление;

– поток самоуничтожается, вызвав ExitThread;

– другой поток данного или стороннего процесса вызывает TerminateThread;

– завершается процесс, содержащий данный поток.

Функции для работы с потоками

В Win32 API определен ряд функций работы с потоками, некоторые из которых перечислены в Таблице 12.

Таблица 12 – Функции для работы с потоками

Название функции	**Выполняемое действие**
CreateThread	Создает поток
ExitThread	Завершает поток
GetCurrentThread	Возвращает описатель (handle) текущего потока
GetCurrentThreadId	Возвращает идентификатор текущего потока
GetThreadPriority	Сообщает приоритет указанного потока
GetThreadTimes	Возвращает временные характеристики указанного потока
ResumeThread	Уменьшает счетчик задержаний потока (или запускает его)
Sleep	Задерживает исполнение потока на указанное количество миллисекунд
SuspendThread	Приостанавливает исполнение указанного потока
TerminateThread	Завершает указанный поток

Информацию об оставшихся функциях можно посмотреть в справочном руководстве к Win32 API.

Функция CreateThread()

Функция `CreateThread()` создает новый поток в адресном пространстве процесса. Функция возвращает описатель порожденного потока. Функция имеет следующий формат:

```
HANDLE CreateThread(     LPSECURITY_ATTRIBUTES
lpThreadAttributes,
DWORD dwStackSize,
LPTHREAD_START_ROUTINE lpStartAddress,
LPVOID lpParameter,
DWORD dwCreationFlags,
LPDWORD lpThreadId
);
```

Параметр `lpThreadAttributes` – указатель на структуру, описывающую параметры защиты потока. Если параметру присвоено значение `NULL`, то устанавливаются атрибуты «по умолчанию».

Параметр `dwStackSize` устанавливает размер стека, который отводится потоку. Если параметр равен нулю, то устанавливается стек, равный стеку первичного потока.

Параметр `lpStartAddress` – адрес функции, которую будет исполнять поток. Функция имеет один 32-битный аргумент и возвращает 32 битное значение.

`lpParameter` – параметр, передаваемый в функцию, которую будет исполнять поток.

Параметр `dwCreationFlags` – дополнительный флаг, который управляет созданием потока. Если этот параметр равен `CREATE_SUSPENDED`, то поток после порождения не запускается на исполнение до вызова функции `ResumeThread`.

Параметр `lpThreadId` – указатель на 32-битную переменную, которой будет присвоено значение уникального идентификатора потока.

4 Практическое задание

Работа с процессами.

а) Напишите программу, запускающую новый процесс с помощью функции CreateProcess().

1) Откройте среду разработки Visual Studio. Выберите пункт меню File→New. В появившемся окне выберите Win32 Console Application. Далее следуйте инструкциям системы.
2) Чтобы увидеть исходный код кликните по закладке FileView, выберите имя_проекта files, кликните Source files, дважды кликните

имя_проекта.cpp. Теперь можно увидеть шаблон простого приложения на C. Можете приступать к написанию программы.

3) Создайте структуру `pi` типа `PROCESS_INFORMATION`, в которую при запуске процесса будут помещены указатели процесса и первичного потока, а также их идентификаторы. Например: `PROCESS_INFORMATION pi;`

4) Создайте структуру `si` типа `STARTUPINFO` (например: `STARTUPINFO si)`, обнулите её поля с помощью функции `ZeroMemory(&si, sizeof(si))`, полю `cb` присвойте значение равное размеру структуры `si`.

5) Теперь, с помощью функции `CreateProcess()` запустите новый процесс в виде приложения Windows Paint, для этого в качестве параметров для функции `CreateProcess()` передайте следующие:

```
lpApplicationName   NULL
lpCommandLine "Абсолютный Путь к приложению paint.exe",
lpProcessAttributes NULL,
lpThreadAttributes  NULL,
bInheritHandles     FALSE,
dwCreationFlags     0,
lpEnvironment  NULL,
lpCurrentDirectory  NULL,
lpStartupInfo  &si,
lpProcessInformation     &pi.
```

В случае неудачи, функция `CreateProcess()` возвращает `FALSE`, в случае удачного завершения – `TRUE`. Сделайте обработку на случай неудачного запуска процесса:

```
if(!CreateProcess(…){
     printf(“Process Creation Error\n”);
     return 0;
}
```

6) Произведите отладку приложения.

7) Запустите приложение. Наблюдайте процесс запуска приложения Windows Paint.

8) Сохраните проект.

9) Добавьте возможность завершения процесса в предыдущую программу. В функции `main()` перед вызовом оператора `return` добавьте функцию, закрывающую процесс:

```
if(!TerminateProcess(pi.hProcess,0)){
    printf("Process ShutDown Error\n");
    return 0;
}
```

10) Произведите отладку и запуск приложения. Наблюдайте процесс запуска и останова приложения Windows Paint.

11) Сохраните проект (File→SaveWorkspace).

б) Получите информацию о запущенном процессе

1) В среде Visual Studio продолжайте работать с ранее созданным проектом.

2) Перед вызовом функции `TerminateProcess()` добавьте вывод на консоль приложения информацию о запущенном процессе: Идентификатор процесса: `pi.dwProcessId`, в десятичном виде. Дескриптор процесса: `pi.hProcess`, в шестнадцатиричном виде. Идентификатор первичного потока: `pi.dwThreadId` в десятичном виде. Дескриптор первичного потока: `pi.hThread`, шестнадцатиричном виде. Класс приоритета процесса: `GetPriorityClass(…)`, передайте в качестве параметра дескриптор процесса.

3) Произведите отладку и запуск приложения. Наблюдайте процесс запуска и останова приложения Windows Paint.

4) Попробуйте менять значение параметра `dwCreationFlags` функции `CreateProcess()`, записывая туда константы соответствующие различным классам приоритета процессов..

5) Произведите отладку и запуск приложения. Наблюдайте изменение приоритета приложения Windows Paint.

6) Сохраните проект.

в) Запустим два процесса одновременно

1) В среде Visual Studio продолжайте работать с ранее созданным проектом.
2) Рядом с объявлением структуры `pi`, объявите структуру `pi1` типа `PROCESS_INFORMATION`, в которую при запуске процесса будут помещены указатели второго процесса и его первичного потока, а так же их идентификаторы.
3) Объявите еще одну структуру `si1` типа `STARTUPINFO` и обнулите её поля с помощью функции `ZeroMemory(&si1, sizeof(si1))`. Полю `cb` присвойте значение равное размеру структуры `si1`.
4) Теперь, с помощью функции `CreateProcess()` запустите второй процесс в виде приложения cmd. Для этого в качестве параметров для функции `CreateProcess()` передайте:

```
lpCommandLine  "cmd",
dwCreationFlags     CREATE_NEW_CONSOLE,
lpStartupInfo  &si1,
lpProcessInformation     &pi1.
```

   Остальные параметры аналогичны запуску предыдущего процесса.
5) Перед вызовом функции `TerminateProcess()` добавьте вывод на консоль приложения информации о втором процессе, аналогично тому как делали это при получении информации о первом процессе.
6) Напишите обработчик закрытия второго процесса.
7) Произведите отладку и запуск приложения. Наблюдайте процесс запуска и останова приложений Windows Paint и Cmd.
8) С помощью функции `SetPriorityClass()` поменяйте класс приоритета первого процесса на `HIGH_PRIORITY_CLASS`, а второго, на `IDLE_PRIORITY_CLASS`.
9) Произведите отладку и запуск приложения. Наблюдайте процесс запуска и останова приложений Windows Paint и Cmd.
10) Сохраните проект.

Работа с потоками

г) Напишите программу, запускающую новый поток с помощью функции `CreateThread()`

1) Создайте еще один проект в среде разработки Visual Studio выберите пункт Win32 Console Application.
2) Создайте функцию, которую будет исполнять поток, например: `DWORD WINAPI ThreadA(void * arg)`. Пусть она в бесконечном цикле выводит на консоль букву «A» с задержкой 100 мс (воспользуйтесь функцией `Sleep(100))`.
3) Объявите переменную идентификатор потока, а также дескриптор потока.

```
DWORD idThreadA;
HANDLE hThreadA;
```

4) Создайте поток, с помощью функции `CreateThread()` параметры:

```
lpThreadAttributes   NULL,
dwStackSize          0,
lpStartAddress       имя функции ThreadA,
lpParameter          NULL,
dwCreationFlags      0,
lpThreadId           адрес переменной идентификатора потока.
```

5) Произведите отладку и запуск приложения. Наблюдайте процесс вывода на консоль символа «A».
6) Сохраните проект.

д) Добавим в программу еще один поток

1) Продолжайте работать со вторым проектом.
2) Создайте функцию наподобие `ThreadA` пусть это будет `ThreadB`, которая будет выводить на консоль букву «B» с той же задержкой.
3) Аналогично первому потоку объявите переменные – идентификатор потока и дескриптор потока.

4) Создайте второй поток с помощью функции `CreateThread()`, аналогично предыдущему.
5) Произведите отладку и запуск приложения. Наблюдайте процесс вывода на консоль символов «A» и «B».
6) Добавьте в программу функции завершения потоков. Для этого в конец функции `main()` добавьте два вызова функции `TerminateThread()`, передав ей первом случае в качестве параметра дескриптор первого потока и нулевой код завершения, и дескриптор второго потока и нулевой код завершения во втором случае. Можете разделить функции завершения потоков паузой, для удобства.
7) Произведите отладку и запуск приложения. Наблюдайте процесс вывода на консоль символов «A» и «B» и поочередного завершения потоков.
8) Сохраните проект.

е) Изменим приоритеты потоков

1) Продолжайте работать со вторым проектом.
2) Перед вызовами функций завершения потоков с помощью функции `SetThreadPriority()` установите приоритет потока `ThreadA` равным `THREAD_PRIORITY_TIME_CRITICAL`, а потока `ThreadB` – `THREAD_PRIORITY_NORMAL`.
3) Произведите отладку и запуск приложения. Наблюдайте процесс вывода на консоль символов «A» и «B».
4) Замените приоритет потока `ThreadA` параметр на `THREAD_PRIORITY_LOWEST`, у второго потока этот параметр оставьте неизменным.
5) Произведите отладку и запуск приложения. Наблюдайте процесс вывода на консоль символов «A» и «B».

6) Замените приоритет потока `ThreadA` параметр на `THREAD_PRIORITY_IDLE`, у второго потока этот параметр сделайте равным `THREAD_PRIORITY_TIME_CRITICAL`.
7) Произведите отладку и запуск приложения. Наблюдайте процесс вывода на консоль символов «A» и «B».
8) Сохраните проект.

ж) Завершение потока с помощью функции `ExitThread()`

1) Продолжайте работать со вторым проектом.
2) В функции второго потока, в цикле установите счетчик циклов. Поставьте условие, что при достижении счетчиком числа 100 выполнялась бы функция `ExitThread()`, с нулевым кодом завершения. Не забудьте закомментировать вызов функции `TerminateThread()` для этого потока.
3) Произведите отладку и запуск приложения. Наблюдайте процесс вывода на консоль символов «A» и «B» и поочередного завершения потоков.
4) Сохраните проект.

з) Индивидуальное задание

По указанию преподавателя выполните одно из заданий, приведенных в таблице. Создайте индивидуальный проект для этого приложения.

Таблица 13 – Варианты индивидуального задания

№	Задание
1	Создать приложение, запускающее пять дочерних потоков. Каждый поток выполняет вывод сообщения об уровне своего приоритета, а затем выводит цифры своего порядкового номера. Повысить приоритет одного из потоков на 2. Понизить приоритет одного из потоков на 1. Принудительно завершить поток, имеющий наименьший приоритет.
2	Создать приложение, запускающее два дочерних потока. Каждый поток выполняет вывод сообщения об уровне своего приоритета, а затем выводит цифры своего порядкового номера. Понизить приоритет одного из потоков на 2 и вывести значение приоритетов

	потоков. Приостановить на некоторое время поток, имеющий наименьший приоритет.
3	Создать приложение, запускающее три дочерних потока. Каждый поток выполняет вывод сообщения о том, что он создан, а затем выводит цифры своего порядкового номера. Для первого потока приоритет не изменять, для второго повысить на 1, для третьего повысить на 2. Вывести значение приоритетов потоков в порядке его убывания.
4	Создать приложение, запускающее четыре дочерних потока. Каждый поток выполняет вывод сообщения о том, что он создан, а затем выводит цифры своего порядкового номера. Изменить приоритеты потоков так, чтобы их приоритеты увеличивались по порядку их создания (первый поток – наименьший приоритет, четвертый – наибольший). Вывести сведения о приоритетах потоков и поочередно завершить их работу.
5	Создать приложение, запускающее три дочерних потока. Каждый поток выполняет вывод сообщения о том, что он создан, а затем выводит цифры своего порядкового номера. Для первого потока приоритет повысить на 2, для второго понизить на 1, для третьего не изменять. Вывести значение приоритетов потоков в порядке его возрастания.
6	Создать приложение, запускающее три дочерних потоков. Каждый поток выполняет вывод сообщения о том, что он создан, а затем выводит цифры своего порядкового номера. Изменить приоритеты потоков так, чтобы их приоритеты уменьшались по порядку их создания (первый поток – наибольший приоритет, третий – наименьший). Вывести сведения о приоритетах потоков и поочередно завершить их работу.

5 Контрольные вопросы

1) Какие четыре события приводят к созданию процессов?
2) Нарисуйте простейшую модель процесса? Какими состояниями процесса можно дополнить эту модель?
3) Какие обстоятельства приводят к завершению процессов?
4) Дайте определения следующим понятиям: планировщик, таблица процессов, вектор прерываний.
5) Опишите классическую модель потоков.
6) Как организованы потоки в пространстве пользователя и ядре?

7) Из чего состоят процесс и поток в среде Windows?
8) Что такое планирование потоков?
9) Уровни приоритета процессов и потоков в среде Windows.
10) Как происходит создание и завершение процесса в среде Windows?
11) Как происходит создание и завершение потока в Windows?