Изображение выглядит как цепь

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Системы реального времени | | |
| Курс | III | | | Группа | 404 |

Отчёт по лабораторной работе № 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 404 |  |  |  | Шишко Даниил Юрьевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

**Оглавление**

[1 Создание процесса CreateProcess с заданием приоритета 3](#_Toc113364063)

[2 Создание потоков \_beginthreadex с заданием относительного с заданием относительного приоритета 3](#_Toc113364064)

[3 Использование wait\* функций для ожидания завершения потоков 3](#_Toc113364065)

[4 Использование механизмов синхронизации с отражением их индивидуальных особенностей: Mutex, Semaphore, Event (сс автосбросом или со сбросом в ручную на выбор), WaitableTimer, CRITICAL\_ SECTION 4](#_Toc113364066)

# Создание процесса CreateProcess с заданием приоритета

1. Функция CreateProcess создает новый процесс и его первичный (главный) поток. Новый процесс запускает заданный исполняемый файл в контексте безопасности вызывающего процесса.

Если вызывающий процесс представляет другого пользователя, новый процесс использует маркер доступа для вызывающего процесса, а не маркер заимствования прав. Чтобы запустить новый процесс в контексте системы безопасности пользователя, обозначенного маркером заимствования прав, используйте функцию CreateProcessAsUser или CreateProcessWithLogonW.

BOOL CreateProcess(

LPCTSTR lpApplicationName , // имя исполняемого модуля

LPTSTR lpCommandLine, // командная строка

LPSECURITY\_ATTRIBUTES lpProcessAttributes , // SD (дескриптор безопасности)

LPSECURITY\_ATTRIBUTES lpThreadAttributes, // SD

BOOL bInheritHandles, // дескриптор параметра наследования

DWORD dwCreationFlags, // флажки создания

LPVOID lpEnvironment, // новый блок конфигурации

LPCTSTR lpCurrentDirectory, // имя текущего каталога

LPSTARTUPINFO lpStartupInfo, // информация предустановки

LPPROCESS\_INFORMATION lpProcessInformation // информация о процессе

);

# Создание потоков \_beginthreadex с заданием относительного с заданием относительного приоритета

1. Функция \_beginthreadex обеспечивает большую подконтрольность создания потока, чем \_beginthread . Функция \_endthreadex также является более гибкой. Например, с помощью \_beginthreadexможно использовать сведения о безопасности, задавать исходное состояние потока (выполняемого или приостановленного) и получить идентификатор только что созданного потока. Можно также использовать дескриптор потока, возвращаемого методом \_beginthreadex , с помощью функций синхронизации API-интерфейса, что невозможно в случае с \_beginthread.

unsigned long \_beginthreadex(

void \*Security, // указатель на дескриптор защиты

unsigned StackSize, // начальный размер стека

unsigned (\*StartAddress)(void\*), // адрес функции задачи

void \*ArgList, // параметры для задачи

unsigned Initflag, // параметры создания задачи

unsigned \*ThrdAddr); // адрес созданной задачи

# Использование wait\* функций для ожидания завершения потоков

1. DWORD WaitForSingleObject(

[in] HANDLE hHandle,

[in] DWORD dwMilliseconds

);

Ожидает, пока указанный объект не перейдет в состояние сигнализации или не истечет интервал ожидания.

DWORD WaitForMultipleObjects(

[in] DWORD nCount,

[in] const HANDLE \*lpHandles,

[in] BOOL bWaitAll,

[in] DWORD dwMilliseconds

);

Ожидает, пока один или все указанные объекты не перейдут в состояние сигнализации или не истечет интервал ожидания.

# Использование механизмов синхронизации с отражением их индивидуальных особенностей: Mutex, Semaphore, Event (сс автосбросом или со сбросом в ручную на выбор), WaitableTimer, CRITICAL\_ SECTION

1. Механизмы синхронизирования

* Mutex

Если два или более потоков должны получить доступ к общему ресурсу одновременно, системе требуется механизм синхронизации, чтобы обеспечить использование ресурса только одним потоком. Mutex — это примитив синхронизации, предоставляющий монопольный доступ к общему ресурсу только одному потоку. Если поток получает мьютекс, второй поток, который хочет получить этот мьютекс, приостанавливается до тех пор, пока первый поток не выпустит мьютекс.

* Semaphore

Семафор (semaphore) – примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующей.

Семафоры могут быть двоичными и вычислительными. Вычислительные семафоры могут принимать целочисленные неотрицательные значения и используются для работы с ресурсами, количество которых ограничено, либо участвуют в синхронизации параллельно исполняемых задач. Двоичные семафоры могут принимать только значения 0 и 1 и используются для взаимного исключения одновременного нахождения двух или более процессов в своих критических секциях.

* Event

Событие — это сообщение, посланное объектом, чтобы сообщить о совершении действия.

Создается объект событие функцией CreateEvent():

HANDLE CreateEvent

(

LPSECURITY\_ATTRIBUTES lpEventAttributes, // атрибут защиты

BOOL bManualReset, // тип сброса TRUE - ручной

BOOL bInitialState, // начальное состояние TRUE - сигнальное

LPCTSTR lpName // имя обьекта

);

Если функция успешна, то вернется дескриптор события. В том случае, если с таким именем событие уже создано, то вернется дескриптор уже созданного события, а GetLastError() вернет код ERROR\_ALREADY\_EXISTS. Но функция может вернуть и NULL, если объект события создать не удалось совсем. Функция SetEvent() меняет состояние на сигнальное (есть событие).

BOOL SetEvent

(

HANDLE hEvent // дескриптор события

);

В случае успеха вернет ненулевое значение. А функция ResetEvent() меняет состояние на невыделенное (нет события):

BOOL ResetEvent

(

HANDLE hEvent // дескриптор события

);

* WaitableTimer

Таймер ожидания переходит в сигнальное состояние по завершении заданного интервала времени. Для его создания используется функция CreateWaitableTimer:

function CreateWaitableTimer(

lpTimerAttributes: PSecurityAttributes; // Адрес структуры

// TSecurityAttributes

bManualReset: BOOL; // Задает, будет ли таймер переходить в

// сигнальное состояние по завершении функции

// ожидания

lpTimerName: PChar // Имя объекта

): THandle; stdcall;

Когда параметр bManualReset равен TRUE, то таймер после срабатывания функции ожидания остается в сигнальном состоянии до явного вызова SetWaitableTimer, если FALSE-таймер автоматически переходит в несигнальное состояние.

Если lpTimerName совпадает с именем уже существующего в системе таймера, то функция возвращает его идентификатор, позволяя использовать объект для синхронизации между процессами. Имя таймера не должно совпадать с именем уже существующих объектов типов event, semaphore, mutex, job или file-mapping.

Идентификатор уже существующего таймера можно получить функцией:

function OpenWaitableTimer(

dwDesiredAccess: DWORD; // Задает права доступа к объекту

bInheritHandle: BOOL; // Задает, может ли объект наследоваться

// дочерними процессами

lpTimerName: PChar // Имя объекта

): THandle; stdcall;

* CRITICAL\_SECTION