**Курсовая работа**

Командный процессор

Вариант №18

Выполнил: студент  
группы ИВТ 41-08  
Павлов П.В.

Проверил: преподаватель Симаков А.Л.

**Задание**

Реализовать командный процессор — оболочку для запуска программ, вывода списка активных заданий, завершения заданий. Программы, запущенные при помощи оболочки должны выполняться вне зависимости от запущенности оболочки. Необходима возможность запуска нескольких экземпляров оболочки. Перезапуск оболочки не должен повреждать список заданий.

*Входные данные*

Входными данными являются команды пользователя следующего вида:

|  |  |
| --- | --- |
| Команда | Назначение |
| ПУСК путь\_к\_приложению [параметры] | Запуск указанной команды с заданными параметрами (если они указаны) |
| СПИСОК | Получение списка всех выполняющихся процессов, запущенных при помощи командного процессора |
| КОНЕЦ идентификатор\_программы | Остановка процесса с заданным идентификатором. Процесс должен быть запущен через командный процессор |
| ВЫХОД | Выход из командного процессора (не приводит к завершению запущенных с его использованием процессов |

*Выходные данные*

Выходными данными является результат выполнения пользовательских команд.

**Интерфейс**

После запуска программа выводит приглашение

>

и ожидает ввода одной из пользовательских команд.

В случае ошибочного ввода выходит уведомление,

>123

Доступные команды:

ПУСК путь\_к\_приложению [передаваемые параметры]

КОНЕЦ идентификатор\_процесса

СПИСОК

ВЫХОД

После выполнения каждой пользовательской команды выходит результат её исполнения. Успешное исполнение команды:

команда выполнена успешно

Ошибочное исполнение команды:

выполнение команды прервано

Выполнение команды **ПУСК**

Ошибочное выполнение команды:

>ПУСК D:\123

выполнение команды прервано

Успешное исполнение команды:

>ПУСК D:\delaycut.exe

команда выполнена успешно

Выполнение команды **СПИСОК**

Успешное исполнение команды:

>СПИСОК

Всего процессов: 2

PID Процесс

2212 D:\delaycut.exe

5756 D:\delaycut.exe

команда выполнена успешно

Выполнение команды **КОНЕЦ**

Ошибочное выполнение команды:

>КОНЕЦ 1234

1234 запущен извне

выполнение команды прервано

Успешное выполнение команды:

>КОНЕЦ 2212

2212 успешно остановлен

команда выполнена успешно

Выполнение команды **ВЫХОД**

>ВЫХОД

выполнение команды прервано

**Используемые функции WinApi**

Функция **CreateJobObject** создает или открывает объект задания.

**Синтаксис**

|  |
| --- |
| **HANDLE** **CreateJobObject(**  **LPSECURITY**\_**ATTRIBUTES** *lpJobAttributes*, // SD  **LPCTSTR** *lpName* // имя задания  **);** |

**Параметры**

***lpJobAttributes***

[in] Указатель на структуру **SECURITY\_ATTRIBUTES**, которая устанавливает дескриптор безопасности для объекта задания и обуславливает могут ли дочерние процессы наследовать возвращенный дескриптор. Если ***lpJobAttributes*** имеет значение ПУСТО (NULL), объект задания получает заданный по умолчанию дескриптор безопасности, и он не может быть унаследован. Списки контроля доступа (**ACL**) в заданном по умолчанию дескрипторе безопасности для объекта задания происходят от первичного маркера или маркера заимствования прав создателя.

***lpName***

[in] Указатель на символьную строку с нулем в конце, определяющую имя задания. Имя ограничивается символами **MAX\_PATH**. Имя сравнивается с учетом регистра.

Если ***lpName*** имеет значение ПУСТО (NULL), задание создано без имени(названия).

Если ***lpName*** соответствует имени существующего события, семафора, мьютекса, ждущего таймера или отображаемого в памяти объекта, функция завершается ошибкой, а функция [**GetLastError**](http://vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm) возвращает значение **ERROR\_INVALID\_HANDLE**. Это происходит потому, что эти объекты совместно используют одно и то же пространство имен.

**Обслуживание терминала:** Имя может иметь префикс "Global\" или "Local\", чтобы явно создать объект в пространстве имен сеанса или глобальной переменной. Остаточный член от имени может содержать любой символ кроме символа наклонной черты влево (\). Для получения дополнительной информации, см. Пространство имен объекта ядра.

**Windows 2000/XP:** Если Обслуживание терминала не выполняется, префиксы "Global\" и "Local\",, игнорируются. Остаточный член от имени может содержать любой символ кроме символа наклонной черты влево.

**Возвращаемые значения**

Если функция завершается успешно, возвращаемое значение - дескриптор объекта задания. Дескриптор имеет право доступа **JOB\_OBJECT\_ALL\_ACCESS**. Если объект существовал перед вызовом функции, функция возвращает дескриптор существующего объекта задания, а **GetLastError** возвращает значение **ERROR\_ALREADY\_EXISTS**.

Если функция завершается с ошибкой, возвращаемое значение ПУСТО (NULL). Чтобы получить дополнительные данные об ошибках, вызовите **GetLastError**.

**Замечания**

Когда задание создано, его поясняющая информация инициализируется нулями, все ограничения неактивны, и нет никаких связанных процессов. Чтобы связать процесс с заданием, используйте функцию [**AssignProcessToJobObject**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_assignprocesstojobobject.htm). Чтобы установить ограничения для задания, используйте функцию[**SetInformationJobObject**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_setinformationjobobject.htm). Чтобы сделать запрос поясняющей информации, используйте функцию **[QueryInformationJobObject](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_queryinformationjobobjec.htm)**.

Чтобы закрыть дескриптор объекта задания, используйте функцию [**CloseHandle**](http://vsokovikov.narod.ru/New_MSDN_API/Handles_objects/fn_closehandle.htm). Задание разрушается, когда был закрыт его последний дескриптор. Если имеются запущенные процессы, все еще связанные с заданием, когда оно разрушено, они продолжат запускаться даже после того, как задание разрушено.

Чтобы компилировать приложение, которое использует эту функцию, определите макрокоманду **\_WIN32\_WINNT** как 0x0500 или позже. Для получения дополнительной информации, см. статью [**Использование заголовков SDK**](http://vsokovikov.narod.ru/New_MSDN_API/Platform_SDK/use_sdk_header.htm).

Функция **AssignProcessToJobObject** связывает процесс с существующим объектом задания.

**Синтаксис**

|  |
| --- |
| **BOOL** **AssignProcessToJobObject(**  **HANDLE** *hJob***,** // дескриптор задания  **HANDLE** *hProcess* // дескриптор процесса  **);** |

**Параметры**

***hJob***

[in] Дескриптор объекта задания, с которым процесс должен быть связан. Функция [**CreateJobObject**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_createjobobject.htm)или [**OpenJobObject**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_openjobobject.htm)возвращает этот дескриптор. Дескриптор должен иметь право доступа **JOB\_OBJECT\_ASSIGN\_PROCESS**. Для получения дополнительной информации, см. статью [**Защита объекта задания и права доступа**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/security_jobobject.htm).

***hProcess***

[in] Дескриптор процесса, который связан с объектом задания. Процесс ранее не должен быть назначен заданию. Дескриптор должен иметь права доступа **PROCESS\_SET\_QUOTA** и **PROCESS\_TERMINATE**. Для получения дополнительной информации, см. статью [**Защита процесса и права доступа**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/security_access_process.htm).

**Обслуживание терминала:** Все процессы в пределах задания должны запускаться в пределах того же самого сеанса.

**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - не ноль.

Если функция завершается с ошибкой, величина возвращаемого значения - ноль. Чтобы получать дополнительные данные об ошибках, вызовите [**GetLastError**](http://vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm).

**Замечания**

После того, как Вы связываете процесс с объектом задания, используя функцию **AssignProcessToJobObject**, процесс подчиняется набору ограничений для задания. Чтобы устанавливать ограничения для задания, используйте функцию **[SetInformationJobObject](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_setinformationjobobject.htm)**.

Если задание имеет регламент непривилегированного (пользовательского) режима и регламент был исчерпан, функция **AssignProcessToJobObject** завершается ошибкой и заданный процесс завершает работу. Если регламент нарушен связанным процессом, **AssignProcessToJobObject** все еще завершается успешно. Однако о нарушении регламента должен быть отчет. Если задание имеет ограничение активного процесса, и это ограничение было нарушено связанным с ним процессом, функция **AssignProcessToJobObject** завершается ошибкой, а заданный процесс завершает работу.

Операции с памятью, выполненные процессом, связанным с заданием, которое имеет ограничение памяти, подчинены ограничению памяти. Операции с памятью, выполненные процессом прежде, чем он был связан с заданием, функцией **AssignProcessToJobObject** не рассматриваются.

Если процесс еще не выполняется, а задание имеет ограничения системы безопасности, функция **AssignProcessToJobObject** может завершаться ошибкой. Например, если первичный маркер процесса содержит локальную группу администраторов, но объект задания имеет ограничение системы безопасности **JOB\_OBJECT\_SECURITY\_NO\_ADMIN**, функция завершается ошибкой. Если задание имеет ограничение системы безопасности **JOB\_OBJECT\_SECURITY\_ONLY\_TOKEN**, процесс должен быть создан приостановленным. Чтобы создать приостановленный процесс, вызовите функцию[**CreateProcess**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_createprocess.htm) с флажком **CREATE\_SUSPENDED**.

Процесс может быть связан только с единственным заданием. Процесс наследует ограничения задания, с которым он связан и добавляет свою поясняющую информацию к заданию. Если процесс связан с заданием, все процессы, которые он создает, связываются с этим заданием по умолчанию. Чтобы создать процесс, который не является частью того же самого задания, вызовите функцию **CreateProcess** с флажком **CREATE\_BREAKAWAY\_FROM\_JOB**.

Чтобы компилировать приложение, которое использует эту функцию, определите макрокоманду **\_WIN32\_WINNT** как **0x0500** или позже. Для получения дополнительной информации, см. статью [**Использование заголовков SDK**](http://vsokovikov.narod.ru/New_MSDN_API/Platform_SDK/use_sdk_header.htm).

Функция **IsProcessInJob** выясняет, выполняется ли процесс в заданном задании.

**Синтаксис**

|  |
| --- |
| **BOOL IsProcessInJob (**  **HANDLE***ProcessHandle*,  **HANDLE***JobHandle*,  **PBOOL***Result*  **);** |

**Параметры**

***ProcessHandle***

[in] Дескриптор процесса, который будет тестирован

***JobHandle***

[in] Дескриптор задания. Если этот параметр имеет значение ПУСТО (NULL), функция проверяет, если процесс выполняется в каком-либо задании.

***Result***

[out] Указатель на значение, которое получает ИСТИНУ (TRUE) если процесс выполняется в задании и в противном случае ЛОЖЬ (FALSE).

**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - не ноль.

Если функция завершается с ошибкой, величина возвращаемого значения - ноль. Чтобы получить дополнительные данные об ошибках, вызовите **[GetLastError](http://vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm)**.

**Замечания**

Чтобы компилировать приложение, которое использует эту функцию, определите макрокоманду **\_WIN32\_WINNT** как 0x0500 или позже. Для получения дополнительной информации, см. статью [**Использование заголовков SDK**](http://vsokovikov.narod.ru/New_MSDN_API/Platform_SDK/use_sdk_header.htm).

Функция **QueryInformationJobObject** извлекает ограничение и информацию о состоянии задания из объекта задания.

**Синтаксис**

|  |
| --- |
| **BOOL** **QueryInformationJobObject(**  **HANDLE** *hJob*, // дескриптор задания  **JOBOBJECTINFOCLASS** *JobObjectInfoClass*, // класс информации  **LPVOID** *lpJobObjectInfo*, // информация об ограничении  **DWORD** *cbJobObjectInfoLength*, // размер информации об ограничении  **LPDWORD** *lpReturnLength*  // запись данных  **);** |

**Параметры**

***hJob***

[in] Дескриптор задания, информация о котором запрашивается. Функция **[CreateJobObject](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_createjobobject.htm)** или **[OpenJobObject](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_openjobobject.htm)** возвращает этот дескриптор. Дескриптор должен иметь право доступа **JOB\_OBJECT\_QUERY**. Для получения дополнительной информации, см. [**Защита объекта задания и права доступа**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/security_jobobject.htm).

Если это значение ПУСТО (NULL), а вызывающий процесс связан с заданием, используется задание, связанное с вызывающим процессом.

***JobObjectInfoClass***

[in] Класс информации об ограничениях, на которые был сделан запрос. Этот параметр может быть одним из нижеследующих значений.

|  |  |
| --- | --- |
| **Значение** | **Предназначение** |
| **JobObjectBasicAccountingInformation** | **Параметр *lpJobObjectInfo* является указателем на структуру**[**JOBOBJECT\_BASIC\_ACCOUNTING\_INFORMATION**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_jobobject_basic_accounting_information.htm)**.** |
| **JobObjectBasicAndIoAccountingInformation** | **Параметр *lpJobObjectInfo* является указателем на структуру**[**JOBOBJECT\_BASIC\_AND\_IO\_ACCOUNTING\_INFORMATION**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_jobobject_basic_and_io_accounting_information.htm)**.** |
| **JobObjectBasicLimitInformation** | **Параметр *lpJobObjectInfo* является указателем на структуру**[**JOBOBJECT\_BASIC\_LIMIT\_INFORMATION**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_jobobject_basic_limit_information.htm)**.** |
| **JobObjectBasicProcessIdList** | **Параметр *lpJobObjectInfo* является указателем на структуру**[**JOBOBJECT\_BASIC\_PROCESS\_ID\_LIST**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_jobobject_basic_process_id_list.htm)**.** |
| **JobObjectBasicUIRestrictions** | **Параметр *lpJobObjectInfo* является указателем на структуру**[**JOBOBJECT\_BASIC\_UI\_RESTRICTIONS**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_jobobject_basic_ui_restrictions.htm)**.** |
| **JobObjectExtendedLimitInformation** | **Параметр *lpJobObjectInfo* является указателем на структуру**[**JOBOBJECT\_EXTENDED\_LIMIT\_INFORMATION**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_jobobject_extended_limit_information.htm)**.** |
| **JobObjectSecurityLimitInformation** | **Параметр *lpJobObjectInfo* является указателем на структуру**[**JOBOBJECT\_SECURITY\_LIMIT\_INFORMATION**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_jobobject_security_limit_information.htm)**.** |

***lpJobObjectInfo***

[out] Информация об ограничении. Формат этих данных зависит от значение параметра ***JobObjectInfoClass.***

***cbJobObjectInfoLength***

[in] Итоговое число запрошенной информации о задании, в байтах.

***lpReturnLength***

[out] Указатель на переменную, которая получает длину данных, записанных в структуре, указанной параметром***lpJobObjectInfo***. Если Вы не желаете получать эту информацию, установите значение ПУСТО (NULL).

**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - не ноль.

Если функция завершается с ошибкой, величина возвращаемого значения - ноль. Чтобы получить дополнительные данные об ошибках, вызовите **[GetLastError](http://vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm).**

**Замечания**

Вы можете использовать функцию **QueryInformationJobObject**, чтобы получить текущие ограничения, изменять их, а затем использовать функцию [**SetInformationJobObject**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_setinformationjobobject.htm), чтобы установить новые ограничения.

Чтобы компилировать приложение, которое использует эту функцию, определите макрокоманду **\_WIN32\_WINNT** как 0x0500 или позже. Для получения дополнительной информации, см. статью [**Использование заголовков SDK**](http://vsokovikov.narod.ru/New_MSDN_API/Platform_SDK/use_sdk_header.htm).

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](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_createprocessasuser.htm)**или**[CreateProcessWithLogonW](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_createprocesswithlogonw.htm)**.

**Синтаксис**

|  |
| --- |
| **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  **);** |

**Параметры**

***lpApplicationName***

[in] Указатель на строку, которая определяет модуль исполняемого кода, с символом нуля в конце. Заданный модуль может быть базирующейся на Windows прикладной программой. Это может быть какой-то другой тип модуля (например, **MS-DOS** или **OS/2**), если соответствующая подсистема доступна на локальном компьютере.

Строка может указывать полный путь и имя файла исполняемого модуля или онаможет определять часть имени. В случае если это часть имени, функция использует текущий диск и текущий каталог, чтобы завершить определение. Функция не будет использовать найденный путь. Если имя файла не содержит расширения, то подразумевается расширение ***.exe***. Поэтому, если расширение имени файла - ***.com***, в этот параметр должно включатся расширение ***.com***.

Параметр ***lpApplicationName*** может быть значением ПУСТО (NULL). В этом случае, имя модуля должно быть в строке ***lpCommandLine*** как первое незаполненное пространство, разграниченное маркером.

Если Вы используете длинное имя файла, которое содержит пробел, применяйте строки в кавычках, чтобы обозначить, где имя файла заканчивается, и начинаются параметры; иначе, имя файла становится неоднозначным. Например, рассмотрим строку " c:\program files\sub dir\program name ". Эта строка может интерпретироваться несколькими способами. Попытки системы интерпретировать ее, возможны в нижеследующем порядке:

|  |
| --- |
| **c:\program.exe** files\sub dir\program name **c:\program files\sub.exe** dir\program name **c:\program files\sub dir\program.exe** name **c:\program files\sub dir\program name.exe** |

**Windows NT/2000/XP:** Если выполняемый модуль - 16-разрядное приложение, параметр ***lpApplicationName*** должен быть значением ПУСТО (NULL), а строка, указанная в параметре ***lpCommandLine*** должна задать выполняемый модуль.

***lpCommandLine***

[in, out] Указатель на строку с символом нуля в конце, определяющую командную строку для выполнения.

**Windows NT /2000/XP:** версия Уникода этой функции, **CreateProcessW**, завершится ошибкой, если этот параметр является строкой типа**const**.

Параметр ***lpCommandLine*** может быть значением ПУСТО (NULL). В этом случае, функция использует строку, указанную параметром ***lpApplicationName*** как командную строку.

Если и ***lpApplicationName*** и ***lpCommandLine*** не пустые (non-NULL), \****lpApplicationName*** задает модуль выполнения, а \****lpCommandLine*** определяет командную строку. Новый процесс может использовать функцию [**GetCommandLine**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_getcommandline.htm), чтобы извлечь взятую в целом командную строку. Консольные процессы, **C** процессы периода выполнения могут использовать параметры **argc** и **argv**.

Консольные процессы, написанные на языке C, могут использовать параметры**argc** и **argv**, чтобы подробно анализировать командную строку. Поскольку **argv [0]** - имя модуля, C - программисты обычно повторяют имя модуля как первый маркер в командной строке.

Если ***lpApplicationName*** имеет значение ПУСТО (NULL), первое незаполненное пространство, ограниченное маркером командной строки, определяет имя модуля. Если Вы используете длинное имя файла, которое содержит пробел, используйте строки в кавычках, чтобы обозначить, где заканчивается имя файла, и начинаются параметры (см. объяснение параметра***lpApplicationName***). Если имя файла не содержит расширения, предполагается расширение ***.exe***. Поэтому, если расширение имени файла - ***.com***, этот параметр должен включить в себя расширение ***.com***. Если имя файла заканчивается точкой (.) без расширения, или имя файла содержит путь, расширение ***.exe*** не присоединяется. Если имя файла не содержит путь к каталогу, система ищет исполняемый файл в нижеследующей последовательности:

|  |
| --- |
| 1. Каталог, из которого загружена прикладная программа. 2. Текущий каталог родительского процесса. 3. **Windows 95/98/Me:** системный каталог Windows. Используйте функцию **GetSystemDirectory**, чтобы получить путь к этому каталогу.   **Windows NT /2000/XP:** 32-разрядный системный каталог Windows. Используйте функцию **GetSystemDirectory**, чтобы получить путь к этому каталогу. Имя (название) этого каталога - **System32**.   1. **Windows NT/2000/XP:** 16-разрядный системный каталог Windows. Нет функции, которая получает путь к этому каталогу, но он находится. Имя этого каталога - **SYSTEM**. 2. Каталог Windows. Используйте функцию **GetWindowsDirectory**, чтобы получить путь к этому каталогу. 3. Каталоги, которые внесены в список в **PATH** переменной окружения. |

Система добавляет нулевой символ к командной строке, чтобы отделить имя файла от параметров. Он делит исходную строку на две строки для внутренней обработки.

***lpProcessAttributes***

[in] Указатель на структуру **SECURITY\_ATTRIBUTES**, которая обуславливает, может ли возвращенный дескриптор быть унаследован дочерними процессами. Если ***lpProcessAttributes*** имеет значение ПУСТО (NULL), дескриптор не может быть унаследован.

**Windows NT /2000/XP:** член ***lpSecurityDescriptor*** структуры определяющей дескриптор безопасности для нового процесса. Если ***lpProcessAttributes*** имеет значение ПУСТО (NULL), или ***lpSecurityDescriptor*** имеет значение ПУСТО (NULL), процесс получает заданный по умолчанию дескриптор безопасности. Списки контроля доступа (**ACL**) в заданном по умолчанию дескрипторе безопасности для процесса происходят от первичного маркера или маркера заимствования прав создателя.

***lpThreadAttributes***

[in] Указатель на структуру **SECURITY\_ATTRIBUTES**, которая обуславливает, может ли возвращенный дескриптор быть унаследован дочерними процессами. Если ***lpThreadAttributes*** имеет значение ПУСТО (NULL), дескриптор не может быть унаследован.

**Windows NT /2000/XP:** член ***lpSecurityDescriptor*** структуры определяющей дескриптор безопасности для главного потока. Если ***lpThreadAttributes*** имеет значение ПУСТО (NULL), или **lpSecurityDescriptor** имеет значение ПУСТО (NULL), поток получает заданный по умолчанию дескриптор безопасности. Списки контроля доступа (**ACL**) в заданном по умолчанию дескрипторе безопасности для потока происходят от первичного маркера или маркера заимствования прав создателя.

***bInheritHandles***

[in] Если этот параметр - ИСТИНА (TRUE), каждый наследуемый дескриптор в вызывающем процессе наследуется новым процессом. Если этот параметр - ЛОЖЬ (FALSE), дескрипторы не наследуются.

|  |
| --- |
| **Обратите внимание!**на то, что унаследованные дескрипторы имеют то же самое значение и права доступа, что и первоначальные дескрипторы. |

***dwCreationFlags***

[in] Флажки, которые управляют классом приоритета и созданием процесса. За перечнем значений обратитесь к статье [**Флажки создания процесса**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/flags_create_process.htm).

Этот параметр также управляет и классом приоритета нового процесса, который используется в определении приоритетов диспетчеризации потоков процесса. За перечнем значений обратитесь к статье о функции **[GetPriorityClass](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_getpriorityclass.htm)**. Если ни один из флажков класса приоритета не установлен, значения по умолчанию класса приоритета **NORMAL\_PRIORITY\_CLASS**, если класс приоритета процесса созданного процесса не является **IDLE\_PRIORITY\_CLASS**или**BELOW\_NORMAL\_PRIORITY\_CLASS**. В данном случае дочерние процессы получают заданный по умолчанию класс приоритета вызывающего процесса.

***lpEnvironment***

[in] Указатель на блок конфигурации нового процесса. Если этот параметр имеет значение ПУСТО (NULL), новый процесс использует конфигурацию вызывающего процесса.

Блок конфигурации состоит из блока строк с символом нуля в конце, который завершается также нулем. Каждая строка представляется в форме:

|  |
| --- |
| **name=value** |

Поскольку знак "=" используется как разделитель, он не должен быть использован в имени переменной окружения.

Блок конфигурации может содержать или символы**Unicode** или **ANSI**. Если блок конфигурации, указанный параметром ***lpEnvironment***, содержит символы **Unicode**, убедитесь, что в параметре***dwCreationFlags*** установлен флажок **CREATE\_UNICODE\_ENVIRONMENT**. Если блок содержит символы **ANSI**, этот флажок будет сброшен.

|  |
| --- |
| **Обратите внимание!** на то, что блок конфигурации в **ANSI** заканчивается двумя нулевыми байтами: один для последней строки, еще один, чтобы завершить блок. Блок конфигурации Уникода заканчивается четырьмя нулевыми байтами: два - для последней строки, еще два, чтобы завершить блок. |

***lpCurrentDirectory***

[in] Указатель на строку с символом нуля в конце, определяющую текущий диск и каталог для дочернего процесса. Строка должна быть полным путем, который включает в себя букву (имя) диска. Если этот параметр является значением ПУСТО (NULL), новый процесс создается с тем же самым текущим диском и каталогом, что и вызывающий процесс. (Этот параметр дается, прежде всего, для оболочек, которым нужно запустить прикладную программу и установить ее исходный диск и рабочий каталог).

***lpStartupInfo***

[in] Указатель на структуру [**STARTUPINFO**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_startupinfo.htm), которая устанавливает оконный режим терминала, рабочий стол, стандартные дескрипторы и внешний вид главного окна для нового процесса.

***lpProcessInformation***

[out] Указатель на структуру [**PROCESS\_INFORMATION**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/str_process_information.htm), которая принимает идентифицирующую информацию о новом процессе.

Дескрипторы в структуре **PROCESS\_INFORMATION**, когда они больше не нужны, должны быть закрыты функцией **[CloseHandle](http://vsokovikov.narod.ru/New_MSDN_API/Handles_objects/fn_closehandle.htm)**.

#### Возвращаемые значения

Если функция завершается успешно, величина возвращаемого значения - не ноль.

Если функция завершается с ошибкой, величина возвращаемого значения - ноль. Чтобы получить дополнительные данные об ошибках, вызовите **[GetLastError](http://vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm)**.

#### Замечания

Процессу присваивается идентификатор. Идентификатор является правильным до тех пор, пока процесс не завершит работу. Он может быть использован, чтобы идентифицировать процесс, или открыть определяемый в функции **[OpenProcess](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_openprocess.htm)** дескриптор процесса. Начальный поток в процессе также получает свой идентификатор. Идентификатор правильный до тех пор, пока поток закончит свою работу и может быть использован, чтобы уникально идентифицировать поток в пределах системы. Эти идентификаторы возвращаются в структуре **PROCESS\_INFORMATION**.

Вызывающий поток может использовать функцию **[WaitForInputIdle](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_waitforinputidle.htm)**, чтобы ждать до тех пор, пока новый процесс не завершит свою инициализацию и станет ждать от пользователя ввода данных без задержки ввода. Этот прием может быть полезным для синхронизации родительского и дочернего процессов, поскольку функция **CreateProcess** возвращает значение не ожидая, когда новый процесс закончит свою инициализацию. Например, создающий процесс должен использовать функцию **WaitForInputIdle** перед попыткой найти окно, связанное с новым процессом.

Предпочтительным способом выключить процесс является использование функции [**ExitProcess**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_exitprocess.htm), потому что эта функция уведомляет все динамически подключаемые библиотеки (**DLL**), связанные с процессом о приближающемся завершении работы. Другой способ завершения процесса не уведомляет связанные **DLL**.

|  |
| --- |
| **Обратите внимание!** на то, что когда поток вызывает **ExitProcess**, другие потоки процесса, завершают работу без возможности выполнить какой-либо дополнительный код (включая код завершения потока связанных **DLL**). За дополнительной информацией обратитесь к статье [**Завершение работы процесса**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/exit_process.htm). |

Если приложение предусматривает блок конфигурации, информация текущего каталога системных дисков автоматически не передается новому процессу. Например, имеется переменная окружения, с названием =C:, значение которой - текущий каталог на диске C. Приложение должно позаботиться, чтобы передать новому процессу информацию о текущем каталоге. Поступая таким образом, приложение должно явно создать эти строки переменной окружения, отсортировывать их в алфавитном порядке (потому что система использует отсортированные переменные) и поместить их в блок конфигурации. Как правило, они пойдут в начале блока конфигурации благодаря порядку установленному сортировкой блока конфигурации.

Один из способов получить переменную текущего каталога для диска **X** состоит в том, чтобы вызвать **[GetFullPathName](http://vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_getfullpathname.htm)** **(" X: ",..)**. Это освобождает прикладную программу от необходимости сканировать блок конфигурации. Если полный возвращенный путь является **X:\**, нет нужды передавать это значение как данные конфигурации, так как корневой директорией является заданный по умолчанию текущий каталог для диска **X** нового процесса.

|  |
| --- |
| **Обратите внимание!** на то, что имя исполняемой программы в командной строке, которое операционная система предоставляет процессу, необязательно идентично тому, которое вызывающий процесс передает функции **CreateProcess** в командной строке. Операционная система может предварительно обуславливать полностью уточненный путь к имени исполняемой программы, которое дано без полного, имеющего необходимые данные пути. |

**Windows NT/2000/XP:** Когда процесс создается с установленным флажком **CREATE\_NEW\_PROCESS\_GROUP**, неявный вызов функции [**SetConsoleCtrlHandler**](http://vsokovikov.narod.ru/New_MSDN_API/Console/fn_setconsolectrlhandler.htm) (**NULL**, **TRUE**) делается от имени нового процесса; это означает то, что новый процесс имеет заблокированным сочетание клавиш **CTRL+C**. Это позволяет оболочкам обрабатывать **CTRL+C** непосредственно и выборочно передавать этот сигнал подпроцессам. Сочетание клавиш **CTRL+BREAK** не блокируется и может быть использовано для прерывания процесса / группы процессов.

**Windows 95/98/Me:** Функция **CreateProcessW** поддерживается подпрограммой [**Microsoft Layer for Unicode**](http://vsokovikov.narod.ru/New_MSDN_API/MSLU/begin_mslu.htm). Чтобы использовать ее, Вы должны добавить некоторые файлы к вашему приложению, как изложено в требованиях этой подпрограммы для систем **Windows 95/98/Me**.

**Замечания по безопасности**

Первый параметр, ***lpApplicationName***, может иметь значение ПУСТО (NULL), в этом случае имя исполняемой программы должно быть первое незаполненное пространство разграничивающее строку в параметре ***lpCommandLine***. Если имя пути или исполняемой программы имеют пробел, имеется риск того, что может быть запущена другая исполняемая программа из-за способа, которым функция подробно анализирует пробелы. Нижеследующий пример демонстрирует эту опасность, потому что функция вместо "MyApp.exe" будет пытаться запустить "Program.exe", если таковая существует.

|  |
| --- |
| **CreateProcess**(NULL, "C:\\Program Files\\MyApp", ...) |

Если неграмотный пользователь создаст в системе прикладную программу, называемую "Program.exe", любая программа, которая неправильно вызывает функцию**CreateProcess**, используя каталог Program Files, будет запускать это приложение вместо заданной программы.

Чтобы избежать этой проблемы, не передавайте значение ПУСТО (NULL) для параметра ***lpApplicationName***. Если Вы передаете это значение ПУСТО (NULL) для ***lpApplicationName***, используйте кавычки вокруг пути к исполняемой программе в параметре ***lpCommandLine***, как показано в примере ниже.

|  |
| --- |
| **CreateProcess**(NULL, "\"C:\\Program Files\\MyApp.exe\" -L -S", ...) |

Функция **TerminateProcess** завершает работу заданного процесса и всех его потоков.

**Синтаксис**

|  |
| --- |
| **BOOL** **TerminateProcess**(  **HANDLE** *hProcess*, // дескриптор процесса  **UINT** *uExitCode*   // код выхода из процесса  ); |

**Параметры**

***hProcess***

[in] Дескриптор процесса, который завершает работу.

**Windows NT/2000/XP:** дескриптор должен иметь право доступа **PROCESS\_TERMINATE**. Для получения дополнительной информации, см. статью [**Защита процесса и права доступа**](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/security_access_process.htm).

***uExitCode***

[in] Код выхода, который использует процесс и потоки, чтобы завершить работу в результате этого вызова. Используйте функцию **[GetExitCodeProcess](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_getexitcodeprocess.htm)**, чтобы извлечь значение выхода процесса. Используйте функцию **[GetExitCodeThread](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_getexitcodethread.htm)**, чтобы извлечь значение выхода потока.

**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - не ноль.

Если функция завершается с ошибкой, величина возвращаемого значения - ноль. Чтобы получать расширенные данные об ошибках, вызовите **[GetLastError](http://vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm)**.

**Замечания**

Функция **TerminateProcess** используется, чтобы безоговорочно заставить процесс выйти из работы. Используйте ее только в экстремальных обстоятельствах. Состояние глобальных данных, поддерживаемых библиотеками динамической связи (**DLL**) может быть поставлено под угрозу, если используется**TerminateProcess**, а не функция **[ExitProcess](http://vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_exitprocess.htm)**.

Функция**TerminateProcess** инициализирует завершение работы процесса и немедленно возвращает значение. Это останавливает выполнение всех потоков в пределах процесса и запрашивает отмену обработки всех ждущих ввода – вывода данных. Завершающий исполнение кода процесс не может выйти из работы, пока не будет закончена обработка всех ждущих ввода - вывода данных или эти действия будут отменены.

Завершение процесса вызывает ниже перечисленное:

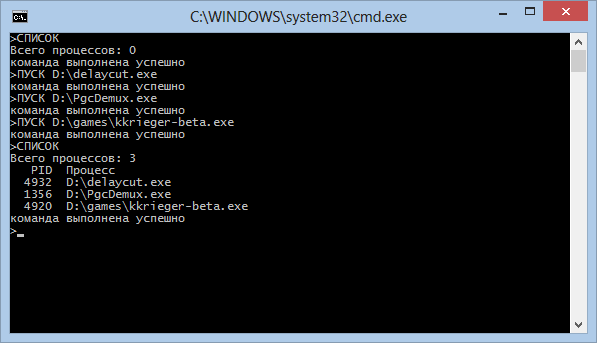
* 1. Все дескрипторы объектов, открытые процессом, закрываются.
  2. Все потоки в процессе завершают исполнение своего кода.
  3. Состояние объекта процесса становится сигнальным, удовлетворяя требования любых потоков, которые ждали когда закончиться процесс. Объект процесса удаляется, когда закрывается последний дескриптор процесса.
  4. Состояние всех потоков процесса становится сигнальным, удовлетворяя требования любых потоков, которые ждали завершения работы потоков.
  5. Состояние завершения процесса изменяется от **STILL\_ACTIVE** до значения выхода процесса.

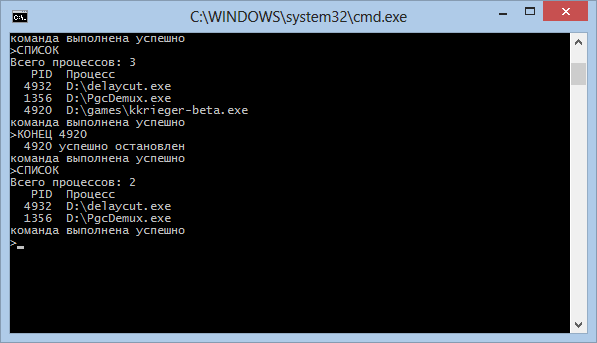
Завершение процесса не заставляет дочерние процессы завершать работу.

Динамически подключаемые библиотеки (DLL), связанные с процессом не уведомляются, что процесс заканчивает работу.

Завершение процесса не создает уведомления для фильтр (hook) - процедуры [**WH\_CBT**](http://vsokovikov.narod.ru/New_MSDN_API/Hook/wh_cbt%20.htm).

**Примеры работы программы**





**Текст программы**

#include <tchar.h>

#include <windows.h>

#include <malloc.h>

#include <iostream>

#include <string>

#include <sstream>

#include <algorithm>

#include <clocale>

#ifdef \_UNICODE

#ifndef UNICODE

#define UNICODE

#endif

#endif

#ifdef UNICODE

#ifndef \_UNICODE

#define \_UNICODE

#endif

#endif

typedef std::basic\_string<TCHAR> tstring;

typedef std::basic\_ostream<TCHAR> tostream;

typedef std::basic\_istream<TCHAR> tistream;

typedef std::basic\_ostringstream<TCHAR> tostringstream;

typedef std::basic\_istringstream<TCHAR> tistringstream;

#ifdef UNICODE

#define tcin std::wcin

#define tcout std::wcout

#else

#define tcin std::cin

#define tcout std::cout

#endif

const int STANDARD\_PROCESS\_IN\_LIST=20;

const int ADDITIONAL\_PROCESS\_IN\_LIST=5;

TCHAR \*g\_startPrefix=\_T("ПУСК");

TCHAR \*g\_listPrefix=\_T("СПИСОК");

TCHAR \*g\_stopPrefix=\_T("КОНЕЦ");

TCHAR \*g\_exitPrefix=\_T("ВЫХОД");

SECURITY\_ATTRIBUTES g\_mutexSA;

HANDLE g\_mutex;

SECURITY\_ATTRIBUTES g\_jobSA;

HANDLE g\_job;

bool isExit=false;

bool initMutexAndJob()

{

// создание (получение) именованного мьютекса

g\_mutexSA.lpSecurityDescriptor = (PSECURITY\_DESCRIPTOR)malloc(SECURITY\_DESCRIPTOR\_MIN\_LENGTH);

InitializeSecurityDescriptor(g\_mutexSA.lpSecurityDescriptor, SECURITY\_DESCRIPTOR\_REVISION);

// ACL is set as NULL in order to allow all access to the object.

SetSecurityDescriptorDacl(g\_mutexSA.lpSecurityDescriptor, TRUE, NULL, FALSE);

g\_mutexSA.nLength = sizeof(g\_mutexSA);

g\_mutexSA.bInheritHandle = FALSE;

g\_mutex=CreateMutex(&g\_mutexSA, FALSE, \_T("Global\\courseTaskManagerMutex"));

//\_tcprintf(\_T("Mutex exists: %d\n"), GetLastError()==ERROR\_ALREADY\_EXISTS);

// создание (получение) именнованного списка процессов

g\_jobSA.lpSecurityDescriptor = (PSECURITY\_DESCRIPTOR)malloc(SECURITY\_DESCRIPTOR\_MIN\_LENGTH);

InitializeSecurityDescriptor(g\_jobSA.lpSecurityDescriptor, SECURITY\_DESCRIPTOR\_REVISION);

// ACL is set as NULL in order to allow all access to the object.

SetSecurityDescriptorDacl(g\_jobSA.lpSecurityDescriptor, TRUE, NULL, FALSE);

g\_jobSA.nLength = sizeof(g\_jobSA);

g\_jobSA.bInheritHandle = TRUE;

g\_job=CreateJobObject(&g\_jobSA, \_T("Global\\courseTaskManagerJob"));

//\_tcprintf(\_T("Job exists: %d\n"), GetLastError()==ERROR\_ALREADY\_EXISTS);

return (g\_mutex!=NULL) && (g\_job!=NULL);

}

bool startProcess(LPCTSTR a\_commandLine)

{

bool \_retVal=false;

DWORD \_mutexState=WaitForSingleObject(g\_mutex, INFINITE);

if (\_mutexState==WAIT\_OBJECT\_0 || \_mutexState==WAIT\_ABANDONED)

{

STARTUPINFO \_startupInfo;

//ZeroMemory(&\_startupInfo,sizeof(STARTUPINFO));

memset(&\_startupInfo, 0, sizeof(\_startupInfo));

\_startupInfo.cb = sizeof(\_startupInfo);

PROCESS\_INFORMATION \_processInformation;

memset(&\_processInformation, 0, sizeof(\_processInformation));

TCHAR \_tempCmdLine[MAX\_PATH \* 2]; //Needed since CreateProcessW may change the contents of CmdLine

\_tcscpy\_s(\_tempCmdLine, MAX\_PATH \*2, a\_commandLine);

BOOL \_result=CreateProcess(NULL, \_tempCmdLine, NULL, NULL, TRUE, NORMAL\_PRIORITY\_CLASS | DETACHED\_PROCESS, NULL, NULL, &\_startupInfo, &\_processInformation);

if (\_result==TRUE)

{

\_result=AssignProcessToJobObject(g\_job, \_processInformation.hProcess);

CloseHandle(\_processInformation.hProcess);

CloseHandle(\_processInformation.hThread);

\_retVal=\_result==TRUE;

}

ReleaseMutex(g\_mutex);

}

return \_retVal;

}

bool list()

{

bool \_retVal=false;

DWORD \_mutexState=WaitForSingleObject(g\_mutex, INFINITE);

if (\_mutexState==WAIT\_OBJECT\_0 || \_mutexState==WAIT\_ABANDONED)

{

JOBOBJECT\_BASIC\_PROCESS\_ID\_LIST \_pidList;

BOOL \_queryResult=QueryInformationJobObject(g\_job, JobObjectBasicProcessIdList, &\_pidList, sizeof(\_pidList), NULL);

int \_maxIds=max(\_pidList.NumberOfAssignedProcesses, STANDARD\_PROCESS\_IN\_LIST)+ADDITIONAL\_PROCESS\_IN\_LIST;

DWORD \_requiredSize=sizeof(JOBOBJECT\_BASIC\_PROCESS\_ID\_LIST)+(\_maxIds)\*sizeof(DWORD);

PJOBOBJECT\_BASIC\_PROCESS\_ID\_LIST \_extendedPidList=(PJOBOBJECT\_BASIC\_PROCESS\_ID\_LIST)\_alloca(\_requiredSize);

\_extendedPidList->NumberOfAssignedProcesses=\_maxIds;

\_queryResult=QueryInformationJobObject(g\_job, JobObjectBasicProcessIdList, \_extendedPidList, \_requiredSize, NULL);

if (\_queryResult==TRUE)

{

TCHAR \_path[MAX\_PATH];

\_tcprintf(\_T("Всего процессов: %d\n"), \_extendedPidList->NumberOfProcessIdsInList);

if (\_extendedPidList->NumberOfProcessIdsInList>0)

{

\_tcprintf(\_T("%6s\t%s\n"), \_T("PID"), \_T("Процесс"));

for (DWORD i=0; i<\_extendedPidList->NumberOfProcessIdsInList; i++)

{

DWORD \_pathSize=MAX\_PATH;

HANDLE \_process=OpenProcess(PROCESS\_QUERY\_INFORMATION, FALSE, \_extendedPidList->ProcessIdList[i]);

if (\_process!=INVALID\_HANDLE\_VALUE)

{

BOOL \_nameQueryResult=QueryFullProcessImageName(\_process, 0, \_path, &\_pathSize);

if (\_nameQueryResult==TRUE)

{

\_tcprintf(\_T("%6d\t%s\n"), \_extendedPidList->ProcessIdList[i], \_path);

} else

{

\_tcprintf(\_T("%6d\t%s\n"), \_extendedPidList->ProcessIdList[i], \_T("не удалось получить путь"));

}

CloseHandle(\_process);

} else

{

\_tcprintf(\_T("%6d\t%s\n"), \_extendedPidList->ProcessIdList[i], \_T("не удалось получить информацию"));

}

}

}

\_retVal=true;

}

ReleaseMutex(g\_mutex);

}

return \_retVal;

}

bool killProcess(DWORD a\_pid)

{

bool \_retVal=false;

DWORD \_mutexState=WaitForSingleObject(g\_mutex, INFINITE);

if (\_mutexState==WAIT\_OBJECT\_0 || \_mutexState==WAIT\_ABANDONED)

{

HANDLE \_process=OpenProcess(PROCESS\_TERMINATE | PROCESS\_QUERY\_INFORMATION, FALSE, a\_pid);

if (\_process!=INVALID\_HANDLE\_VALUE)

{

BOOL \_isProcessInOurJob=FALSE;

BOOL \_execResult=IsProcessInJob(\_process, g\_job, &\_isProcessInOurJob);

if ((\_execResult==TRUE) && (\_isProcessInOurJob==TRUE))

{

BOOL \_terminateResult=TerminateProcess(\_process, -1);

if (\_terminateResult==TRUE)

{

\_retVal=true;

\_tcprintf(\_T("%6d успешно остановлен\n"), a\_pid);

} else

{

\_tcprintf(\_T("%6d не может быть остановлен\n"), a\_pid);

}

} else

{

\_tcprintf(\_T("%6d запущен извне\n"), a\_pid);

}

} else

{

\_tcprintf(\_T("%6d некорректный идентификатор\n"), a\_pid);

}

ReleaseMutex(g\_mutex);

}

return \_retVal;

}

void exitTaskManager()

{

isExit=true;

}

void deinitialize()

{

free(g\_mutexSA.lpSecurityDescriptor);

free(g\_jobSA.lpSecurityDescriptor);

}

void trim(tstring& a\_string, const TCHAR\* a\_delimeters)

{

a\_string.erase(0, a\_string.find\_first\_not\_of(a\_delimeters));

a\_string.erase(a\_string.find\_last\_not\_of(a\_delimeters)+1, tstring::npos);

}

int \_tmain(int argC, LPCTSTR argV[])

{

setlocale( LC\_ALL, ".866" );

bool \_initResult=initMutexAndJob();

if (\_initResult)

{

atexit(deinitialize);

tstring \_line=\_T("");

do

{

tcout<<\_T(">");

std::getline(tcin, \_line);

trim(\_line, \_T(" \t\r\n"));

bool \_commandResult=false, \_isCommand=true;

if (!\_tcsnicoll(\_line.c\_str(), g\_startPrefix, \_tcslen(g\_startPrefix)))

{

\_line.erase(0, \_tcslen(g\_startPrefix));

trim(\_line, \_T(" \t\r\n"));

if (\_tcslen(\_line.c\_str())>0)

{

\_commandResult=startProcess(\_line.c\_str());

} else

{

\_isCommand=false;

}

} else

{

if (!\_tcsnicoll(\_line.c\_str(), g\_stopPrefix, \_tcslen(g\_stopPrefix)))

{

\_line.erase(0, \_tcslen(g\_stopPrefix));

trim(\_line, \_T(" \t\r\n"));

if (\_tcslen(\_line.c\_str())>0)

{

TCHAR \*\_endPtr;

DWORD \_pid=\_tcstol(\_line.c\_str(), &\_endPtr, 0);

\_commandResult=killProcess(\_pid);

} else

{

\_isCommand=false;

}

} else

{

if (!\_tcsicoll(\_line.c\_str(), g\_listPrefix))

{

\_commandResult=list();

} else

{

if (!\_tcsicoll(\_line.c\_str(), g\_exitPrefix))

{

exitTaskManager();

} else

{

\_isCommand=false;

}

}

}

}

if (\_isCommand)

{

if (\_commandResult)

{

tcout<<\_T("команда выполнена успешно")<<std::endl;

} else

{

tcout<<\_T("выполнение команды прервано")<<std::endl;

}

} else

{

tcout<<\_T("Доступные команды: ")<<std::endl;

tcout<<\_T("\tПУСК путь\_к\_приложению [передаваемые параметры]")<<std::endl;

tcout<<\_T("\tКОНЕЦ идентификатор\_процесса")<<std::endl;

tcout<<\_T("\tСПИСОК")<<std::endl;

tcout<<\_T("\tВЫХОД")<<std::endl;

}

} while(!isExit);

}

}