Оглавление

ЛАБОРАТОРНАЯ РАБОТА ПРОЦЕССЫ И ПОТОКИ (ПРОГРАММИРОВАНИЕ)

Теория

В этой лабораторной работе представлена методология создания ***параллельных приложений*** (***concurrent application***). Такое приложение может исполняться в двух и более местах одновременно. Написание параллельных приложений полезно по двум причинам. Во-первых, современное ПО разрабатывается по частям, которые могут выполняться параллельно. Во-вторых, когда приложение использует параллельную модель, современные ОС, такие как Windows, представляют много возможностей для разработчиков для управления параллельными приложениями. В этой лабораторной работе параллельность рассматривается на уровне процесса и потока. Обычные программы, которые вы писали, выполнялись как один процесс с одним потоком (последовательное программирование). Современное поколение ПО использует преимущества многих процессоров на вашем компьютере или доступных по сети. Таким образом, традиционные последовательные модели вычислений заменяются параллельными моделями со многими процессами и потоками. В лабораторной работе будет представлена информация о процессах и потоках Windows и рассмотрено создание процессов и потоков, используя Win32/64 API.

Создание процесса

Один процесс может создать другой, вызывая ***CreateProcess*** (при этом вызове используются Native API *NTCreateProcess* и *NTCreate Thread*). Когда создается процесс исполнительная система (Executive) выполняет большой объем работы. Она выделяет новое адресное пространство и ресурсы для процесса, а также создает для него новый базовый поток. Когда новый процесс создан, старый процесс будет продолжать исполняться, используя старое адресное пространство, а новый будет выполняться в новом адресном пространстве с новым базовым потоком. Существует много различных опций для создания процесса, поэтому функция ***CreateProcess*** имеет много параметров, причем некоторые из них достаточно сложные. После того, как исполнительная система создала новый процесс, она возвращает его описатель (handle), а также описатель его базового потока. Рассмотрим прототип функции ***CreateProcess***. В прототипе не используются стандартные типы С, а вместо этого используется набор типов, определенный в **windows.h (см**. типы_Windows_LPSTR _и_др.pdf).

```
BOOL CreateProcess(
        LPCTSTR lpApplicationName,
        // имя исполняемого модуля (указатель)
        LPTSTR lpCommandLine,
        // командная строка (указатель)
        LPSECURITY_ATTRIBUTES lpProcessAttributes,
        // атрибуты безопасности процесса (указатель)
        LPSECURITY_ATTRIBUTES lpThreadAttributes,
        // атрибуты безопасности потока (указатель)
        BOOL bInheritHandeles, // флаг наследования описателя
        DWORD dwCreationFlags,      // флаги создания
        LPVOID lpEnvironment,
        // новый блок окружения (указатель)
```

```
    LPCTSTR lpCurrentDirectory,
    // имя текущей директории (указатель)
    LPSTARTUPINFO lpStartupInfo,
    // STARTUPINFO (указатель)
    LPPROCESS_INFORMATION lpProcessInformation
    // PROCESS_INFORMATION (указатель)
                    )
```

Десять параметров CreateProcess обеспечивают большую гибкость в использовании для программиста, хотя в простейшем случае для многих параметров можно использовать значения по умолчанию (by default). Здесь мы рассмотрим относительно простой набор параметров, а более детально можно посмотреть в Microsoft Developers Network (MSDN).

Параметры lpApplicationName и lpCommandLine

Два первых параметра обеспечивают два различных способа определения имени файла, который будет выполняться базовым потоком процесса. lpApplicationName – эта строка содержит имя файла, который будет выполняться, а lpCommandLine – эта строка содержит командную строку для запуска процесса в cmd.exe. Существует набор правил, определяющий в каком случае какое имя использовать (см. MSDN).

Параметр lpApplicationName определяет исполняемый модуль. В строке, на которую указывает этот указатель, надо задать:

- полный путь доступа и имя файла, включая расширение (**.exe** или **.bat**)
- только имя тогда будет использоваться текущий диск и каталог.

Если значение параметра lpApplicationName равно NULL, то именем исполняемого модуля должна быть первая из разделенных пробелами лексем, заданная параметром lpCommandLine. Если полный путь доступа не задан, то поиск файла будет производится в следующем порядке:

- текущий каталог
- системный каталог Windows (GetSystemDirectory)
- каталог Windows (GetWindowsDirectory)

- каталоги, заданные в переменной окружения PATH.

Будем задавать NULL для lpApplicationName и командную строку для lpCommandLine. Предположим, вы хотите создать процесс для запуска notepad.exe с файлом temp.txt.

Параметры lpProcessAttributes, lpThreadAttributes и bInheritHandeles

В данном случае будем использовать значения по умолчанию для атрибутов безопасности процесса и потока – NULL и FALSE для флага наследования (атрибуты безопасности и флаг наследования будут рассмотрены далее).

CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE, …);

Параметр DwCreationFlags

Этот параметр используется для управления приоритетом нового процесса и другими особенностями.

Порожденный процесс - процесс-ребенок (child) может быть создан с одним из четырех классов приоритетов HIGH_PRIORITY_CLASS, IDLE_PRIORITY_CLASS, NORMAL_PRIORITY_CLASS или REALTIME_PRIORITY_CLASS. Значение по умолчанию - NORMAL_PRIORITY_CLASS, но если порождающий процесс (процесс родитель parent) имеет приоритет IDLE_PRIORITY_CLASS, то и процесс-ребенок также будет иметь приоритет IDLE_PRIORITY_CLASS.

Потоки процесса с приоритетом HIGH_PRIORITY_CLASS будут вытеснять потоки процессов с приоритетами IDLE_PRIORITY_CLASS или NORMAL_PRIORITY_CLASS. Потоки процесса с приоритетом IDLE_PRIORITY_CLASS будут выполняться только в том случае, если нет других потоков для выполнения.

Потоки процесса с приоритетом REAL_PRIORITY_CLASS будут вытеснять потоки всех других классов.

Можно задать и другие флаги, а также использовать вместе (все возможные значения см. в MSDN). Здесь нам будет полезен флаг CREATE_NEW_CONSOLE – новый процесс будет создан в своем собственном окне cmd.exe.

Для создания нового процесса (child) с высоким приоритетом в его собственном окне используйте - HIGH_PRIORITY_CLASS | CREATE_NEW_CONSOLE.

CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
HIGH_PRIORITY_CLASS | CREATE_NEW_CONSOLE, …);

Параметр lpEnvironment

Используется для передачи нового блока переменных окружения порожденному процессу-ребенку (child). Если NULL, то ребенок использует то же окружение, что и родитель. Если не NULL, то lpEnvironment должен указывать на массив строк, каждая name=value. Здесь мы будем использовать NULL.

CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
HIGH_PRIORITY_CLASS | CREATE_NEW_CONSOLE, NULL, …);

Параметр lpCurrentDirectory

Указатель на строку, содержащую путь к текущему каталогу нового процесса. Если использовать NULL, то в качестве текущего каталога будет использован каталог родителя.

CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
HIGH_PRIORITY_CLASS | CREATE_NEW_CONSOLE, NULL, NULL, …);

Параметр lpStartupInfo

Это указатель на следующую структуру, которая определяет внешний вид окна и содержит дескрипторы стандартных устройств ввода/вывода/ошибки нового процесса:

```
typedef struct_STARTUPINFO { //si
    DWORD cb; // длина структуры
    LPTSTR lpReserved;
    LPTSTR lpDesktop;
    LPTSTR lpTitle;
    DWORD dwX;
    DWORD dwY;
    DWORD dwXSize;
    DWORD dwSizeY;
    DWORD dwXCountChars;
    DWORD dwYCountChars;
    DWORD dwFillAttribute;
    DWORD dwFlags;
    DWORD dwShowWindow;
    WORD    cbReserved2;
    LPBYTE lpReserved2;
    HANDLE hStdInput;
    HANDLE hStdOutput;
    HANDLE hStdError; } STARTUPINFO, *LPSTARTUPINFO;
```

Экземпляр этой структуры данных должен быть создан в вызывающей программе (описание полей структуры см. в MSDN). Затем ее адрес должен быть передан как параметр в CreateProcess.

Внимание!!! CreateProcess не знает длины структуры, поэтому обязательно инициализируйте поле размера в STARTUPINFO (поле .cb) до того как передать указатель.

```
STARTUPINFO startupInfo;
…
ZeroMemory(&startupInfo,sizeof(STARTUPINFO));
startupInfo.cb=sizeof(startupInfo);
CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
 HIGH_PRIORITY_CLASS | CREATE_NEW_CONSOLE, NULL, NULL,
&startupInfo, …);
```

Параметр lpProcessInformation

Это указатель на структуру, в которую будут помещены возвращаемые функцией значения дескрипторов и глобальных идентификаторов процесса и потока:

```
Typedef struct_PROCESS_INFORMATION {
        HANDLE hProcess;
        HANDLE hThread;
        DWORD  dwProcessId;
        DWORD  dwThreadId; } PROCESS_INFORMATION;
```

В этой структуре нет поля длины, поэтому процессу нужно только создать экземпляр PROCESS_INFORMATION и передать указатель на него функции CreateProcess. После возврата, поля структуры будут заполнены, и вы получите:

- Описатель вновь созданного процесса (hProcess)
- Описатель его базового потока (hThread)
- Идентификатор процесса (dwProcessId)
- Идентификатор потока (dwThreadId).

CreateProcess отводит место под объекты процесс и поток и возвращает значения их описателей (индексы в таблице описателей) в структуре PROCESS_INFORMATION. Вы можете освободить выделенное место, используя CloseHandle. Конечно, когда процесс заканчивается, все его описатели автоматически освобождаются. Хотя описатель — это системный

ресурс, хорошая практика, если вы будете закрывать описатель, если не используете его.

CreateProcess возвращает ноль, если все плохо и не ноль в противном случае.

Далее приводится код для создания процесса:

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
…
STARTUPINFO startInfo;
PROCESS_INFORMATION processInfo;
LPCWSTR p = L"C:\\WINDOWS\\SYSTEM32\\NOTEPAD.EXE";
…
strcpy(lpCommandLine,p);
ZeroMemory(&startupInfo,sizeof(STARTUPINFO));
startupInfo.cb=sizeof(startupInfo);
if(!CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
HIGH_PRIORITY_CLASS | CREATE_NEW_CONSOLE, NULL, NULL,
&startupInfo, &processInfo))
{
fprintf(stderr,”CreateProcess failed on error %d\n”,GetLastError());
ExitProcess(1);
}
…
CloseHandle(processInfo.hThread);
CloseHandle(processInfo.hProcess);
```

Создание потока

Вы можете создать дополнительные потоки в текущем процессе, используя функцию ***CreateThread*** Win32/64 API (которая использует *NTCreateThread* Native API). Каждый поток — это отдельная исполняемая сущность внутри

адресного пространства процесса. Для создания потока программист должен задать необходимую информацию.

Рассмотрим прототип функции:

```
HANDLE CreateThread(
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    // атрибуты безопасности потока (указатель)
    DWORD dwStackSize,
    // начальный размер стека потока в байтах
    LPTHREAD_START_ROUTINE lpStartAddress,
    // функция потока (указатель)
    LPVOID lpParameter, // аргумент для нового потока
    DWORD dwCreationFlags, // флаги создания
    LPDWORD lpThreadId
    // возвращаемый идентификатор потока (указатель)
            );
```

Этот прототип использует шесть параметров для описания характеристик нового потока. Функция создает описатель потока, и возвращать его как результат вызова. Это означает, что системные ресурсы размещаются при удачном вызове ***CreateThread*** и программист должен закрыть описатель, когда он больше не используется.

Параметр lpThreadAttributes

Это то же самое, что и атрибуты безопасности в CreateProcess только для потока. В простейшем случае NULL (будет рассмотрено далее).

```
CreateThread(NULL, …);
```

Параметр dwStackSize

Каждый поток выполняется независимо от других потоков процесса и имеет свой собственный стек. Используя этот параметр, программист может установить размер стека. Обычно используется значение по умолчанию – 0.

CreateThread(NULL, 0, …);

Параметры lpStartAddress и lpParameter

Для процесса необходимо было задать имя выполняемого файла. Для потока необходимо задать адрес в текущем адресном пространстве, где новый поток должен выполняться. LpStartAddress – это и есть такой адрес. В языках программирования (таких как C) обычно невозможно задать в качестве стартового адреса адрес в середине процедуры. Нужно использовать точку входа в процедуру (entry point). LpStartAddress – это точка входа для функции, которая имеет следующий прототип:

DWORD WINAPI ThreadFunc(LPVOID);

Как передать параметр функции, которая будет выполняться новым потоком? Необходимо использовать lpParameter, он будет передан функции как параметр, когда новый поток начнет ее выполнение.

Пусть дан прототип функции, которую будет выполнять новый поток:

DWORD WINAPI myFunc(LPVOID);

Предположим, что родительский поток хочет передать целый аргумент новому детскому потоку:

int theArg;

…

CreateThread(NULL, 0, myFunc, &theArg, …);

Параметр dwCreationFlags

Этот параметр используется для определения, каким образом создать новый поток. Возможное значение CREATE_SUSPENDED – в этом случае новый поток будет создан, но приостановлен до тех пор, пока другой поток не выполнит вызов:

ResumeThread(targetThreadHandle);

Где targetThreadHandle описатель нового потока. Значение по умолчанию 0 – в этом случае поток будет активизирован сразу после создания.

CreateThread(NULL, 0, myFunc, &theArg, 0, …);

Параметр lpThreadId

Это указатель на идентификатор потока (аналог поля dwThreadId в структуре PROCESS_INFORMATION, которую возвращает CreateProcess).

DWORD targetThreadId;

…

CreateThread(NULL, 0, myFunc, &theArg, 0, & targetThreadId);

Чему нужно научиться

Интерпретатор командной строки (cmd.exe в NT, command.com в DOS или один из shell UNIX) это программа, которая обеспечивает текстовый интерфейс для общения пользователя с ОС. Пользователь запускает программу из командной строки интерпретатора, печатая имя файла, который содержит выполняемую программу, с набором параметров. Это таже самая информация, которая задается параметром lpCommandLine в CreateProcess. Интерпретатор разбирает строку, чтобы получить имя файла, а затем заставляет процесс выполнить команду. В UNIX shell процесс командной строки действительно создает новый процесс, который загрузит и выполнит программу из файла. Интерпретатор командной строки ждет, пока команда выполнится, и ее поток закончится, перед тем как предоставить пользователю возможность дальше вводить команды.

Нужно научиться писать программы на C, используя функции ***CreateProcess*** и ***CreateThread*** для создания:

- одного или нескольких процессов (каждый с базовым потоком) – задание **A;**
- нескольких потоков в одном процессе – задание **B**.

Задания

Уровень 1 (A)

Используя функцию ***CreateProcess***, создайте процесс для запуска notepad.exe.

Уровень 2 (A)

Используйте текстовый редактор (например, notepad) для подготовки конфигурационного файла. Он может содержать следующий перечень:
C:\\WINDOWS\\SYSTEM32\\NOTEPAD.EXE our.txt
C:\\WINDOWS\\SYSTEM32\\CALC.EXE
C:\\WINDOWS\\SYSTEM32\\CHARMAP.EXE
Напишите программу, которая получает имя конфигурационного файла из командной строки, должна открыть конфигурационный файл, прочитать строки и создать процесс для запуска каждой команды.

Уровень 3 (A)

Необходимо выполнить задание Уровень 3 (B).

Уровень 1 (B)

Создайте два дополнительных потока в вашем процессе. Пусть функция вашего первого потока в бесконечном цикле выводит сообщение (например, “Я поток 1 и я выполняюсь!”), а функция вашего второго потока в бесконечном цикле выводит сообщение (например, “Я поток 2 и я выполняюсь!”). Базовый поток после создания этих двух дополнительных потоков пусть тоже в бесконечном цикле выводит сообщение (например, “Я базовый поток и я выполняюсь!”). Затем измените программу так, чтобы после создания двух дополнительных потоков, базовый поток з**ак**анчивал работу. Объясните полученный вывод.
Теперь измените программу так, чтобы после создания двух дополнительных потоков, базовый поток вызывал функцию ***Sleep***. Объясните полученный вывод.

Уровень 2 (В)

Создать в цикле несколько дополнительных потоков в процессе. В качестве аргументов командной строки задать число создаваемых потоков и время их существования. Пусть функция потока в бесконечном цикле выводит сообщение (например, “Я поток с номером N и я выполняюсь!”).

Используйте системное время. Время, которое процесс и потоки в нем должны существовать, задается как параметр, и по истечении этого времени сообщество должно погибнуть. После того как рабочие потоки созданы и выполняют свою работу, поток координатор проверяет текущее время, чтобы определить, не наступил ли назначенный час. Если нет, то он засыпает на 1сек, а потом просыпается и проверяет время снова. Когда время истечет поток координатор устанавливает глобальную переменную runFlag=FALSE, ждет 5сек и заканчивается. Используйте разделяемую всеми потоками глобальную переменную runFlag.

Уровень 3 (В)

Разработайте программу, которая позволит изменять класс приоритета процесса и приоритеты потов этого процесса. Как значение приоритета влияет на выделение процессорного времени? Что будет, если запретить динамическое изменение приоритета?

Изучите и используйте в своей программе функции: Изучите и используйте в своей программе функции: ***GetPriorityClass, SetPriorityClass, SetProcessPriorityBoost, GetProcessPriorityBoost, ExitProcess, TerminateProcess, GetThreadPriority, SetThreadPriority, SetThreadPriorityBoost, ThreadPriorityBoost, SuspendThread и ResumeThread.***