МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления

М. Г. ЖУРАВЛЕВА

ИЗУЧЕНИЕ WINDOWS API

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ по курсам
«Операционные системы»
и «Операционные системы и оболочки»

Липецк
Липецкий государственный технический университет
2013

УДК 004.451 (07)
Ж 911

Рецензент – В. В. Ведищев, канд. техн. наук, доцент

Журавлева, М. Г.

Ж 911 Изучение Windows API [Текст]: методические указания к выполнению лабораторных работ по курсам «Операционные системы» и «Операционные системы и оболочки» / М.Г. Журавлева. – Липецк: Изд-во ЛГТУ, 2013. – 34 с.

Рассмотрены функции Windows API для управления файлами, процессами, потоками, мьютексами, семафорами и событиями в классических приложениях современных ОС семейства Windows. Представлены порядок выполнения лабораторной работы и оформления отчета, задания к лабораторным работам с вариантами, теоретические сведения с пояснениями, контрольные вопросы.

Предназначены для студентов третьего курса факультета автоматизации и информатики специальности 010503.65, изучающих дисциплину «Операционные системы и оболочки», студентов третьего курса факультета автоматизации и информатики специальности 230102.65 и направлений 230100.62, 010500.62, 231000.62, а также студентов второго курса факультета автоматизации и информатики направления 230401.51, изучающих дисциплину «Операционные системы».

Табл. 6. Библиогр.: 4 назв.

Общие сведения

Лабораторный практикум предполагает изучение внутреннего устройства операционных систем (ОС) посредством разработки консольных приложений в ОС семейства Windows с использованием Windows API. Все программы следует разрабатывать на языках C/C++, используя там, где возможно, функции Windows API. Теоретические сведения для каждой лабораторной работы включают описание требуемых функций и/или пояснения. Некоторые детали описания представленных функций опущены, минимальные требования к ОС-клиенту для большей части функций – Windows XP [1, 2].

Рекомендуемый порядок выполнения лабораторной работы:

1. Построение алгоритма решения задачи.

2. Написание программы: проверка корректности вводимых пользователем данных (при вводе пользователем некорректных данных следует выводить сообщение об ошибке); написание функций, выполняющих выбранные в соответствии с вариантом операции; реализация интерфейса пользователя.

3. Тестирование программы, обработка результатов работы.

4. Оформление отчета в соответствии с СТО-13-2011, в основную часть должны быть включены следующие пункты: название лабораторной работы; цель работы; краткие теоретические сведения; описание экспериментальной установки и методики эксперимента; экспериментальные результаты; обработка и анализ результатов работы; выводы.

Перед началом работы в лаборатории студент обязан пройти инструктаж по технике безопасности, ознакомиться с расположенными на стенде лаборатории инструкцией № 396 по охране труда для операторов и пользователей ЭВМ, выпиской из норм и правил ЛГТУ по охране труда, извлечениями из инструкции по пожарной безопасности зданий, сооружений и помещений ЛГТУ.

Исходные данные для лабораторных работ выбираются из таблиц прил. по номеру варианта, который соответствует номеру студента в журнале преподавателя. С примерами программ можно ознакомиться в [1].

Лабораторная работа № 1

Разработка программ обработки файлов в ОС Windows

Цель работы

Получение навыков использования функций Windows API, обеспечивающих обработку файлов.

Задание кафедры

Написать программу вставки *n* строк исходного файла в заданную позицию результирующего файла и последующей установки/получения атрибутов результирующего файла. Исходные данные указываются пользователем в аргументах командной строки: количество строк – в качестве второго аргумента, номера строк – в качестве третьего – [(n+3)-1]-го аргументов, далее следуют имена исходного и результирующего файлов, следующий аргумент – позиция, в которую следует вставлять строки (исходные данные выбираются из табл. 1 прил.).

Теоретические сведения

Обработка файлов средствами Windows API предполагает использование традиционных функций создания и открытия, чтения и записи, перемещения указателя, закрытия файлов.

1. Создание и открытие файла:

```
«HANDLE WINAPI CreateFile(
        LPCTSTR lpFName,
        DWORD dwDesAccess,
        DWORD dwShMode,
        LPSECURITY_ATTRIBUTES lpSAttributes,
        DWORD dwCreatDisposition,
        DWORD dwFlgAndAttributes,
        HANDLE hTempFile
);»,
```

где *lpFName* – имя создаваемого или открываемого файла/устройства (под устройством понимается каталог, физический диск, том, буфер консоли, канал и др.). В этом имени можно использовать прямые (/) или обратные (\) слэши. В ANSI-версии функции длина имени ограничивается значением MAX_PATH,

для увеличения длины до 32767 символов можно вызывать UNICODE-версию, помещая перед именем префикс "\\?\";

dwDesAccess – требуемый тип доступа к файлу/устройству: GENERIC_READ (чтение), GENERIC_WRITE (запись), GENERIC_READ | GENERIC_WRITE (чтение или запись) и др.;

dwShMode – требуемый режим совместного использования файла/устройства: FILE_SHARE_READ (параллельное чтение), FILE_SHARE_WRITE (параллельная запись), FILE_SHARE_READ | FILE_SHARE_WRITE (параллельные чтение или запись), FILE_SHARE_DELETE (параллельный запрос на удаление/переименование), 0 (разделение запрещено) и др.;

lpSAttributes – указатель на структуру SECURITY_ATTRIBUTES, содержащую дескриптор безопасности, контролирующий доступ к файлу/устройству, и признак наследуемости возвращаемого функцией *CreateFile* дескриптора. Если значение данного параметра – NULL, дескриптор не наследуется;

dwCreatDisposition – действие, которое нужно совершить с файлом/устройством: CREATE_NEW (создать новый, если он еще не существует), CREATE_ALWAYS (создать новый, если существует, создать заново), OPEN_ALWAYS (открыть, при необходимости – создать и открыть), OPEN_EXISTING (открыть только существующий), TRUNCATE_EXISTING (открыть только существующий, сделав его размер нулевым);

dwFlgAndAttributes – атрибуты и флаги файла/устройства, параметр может включать комбинацию (с помощью «|») из любых доступных атрибутов: FILE_ATTRIBUTE_NORMAL (значение по умолчанию, не устанавливается одновременно с другими атрибутами), FILE_ATTRIBUTE_READONLY (только чтение), FILE_ATTRIBUTE_ARCHIVE (архивный), FILE_ATTRIBUTE_-HIDDEN (скрытый) и другие атрибуты и флаги;

hTempFile – дескриптор файла-шаблона с правом доступа «GENERIC_READ», предоставляет атрибуты файла и расширенные атрибуты для создаваемого файла, может принимать значение NULL.

Если операция успешна, функция возвращает открытый дескриптор файла/устройства, иначе - INVALID_HANDLE_VALUE.

2. Закрытие файла (и других объектов):

```
«BOOL WINAPI CloseHandle(
        HANDLE hObj
);»,
```

где *hObj* – дескриптор открытого объекта (файла).

Функция закрывает также дескрипторы процессов, потоков, событий, мьютексов, семафоров, каналов и других объектов. Если операция успешна, функция возвращает не ноль, иначе ноль.

3. Чтение файла:

```
«BOOL WINAPI ReadFile(
        HANDLE hF,
        LPVOID lpBuf,
        DWORD nNumberBytesToRead,
        LPDWORD lpNumberBytesRead,
        LPOVERLAPPED lpOverlap
);»,
```

где *hF* – дескриптор файла/устройства;

lpBuf – указатель на буфер для считанных данных;

nNumberBytesToRead – максимальное количество считываемых байтов;

lpNumberBytesRead – указатель на переменную, получающую количество считанных байтов при синхронном дескрипторе *hF*, при асинхронном дескрипторе (когда *hF* был открыт с флагом FILE_FLAG_OVERLAPPED) следует использовать NULL. Параметр может быть NULL, только если *lpOverlap* не NULL;

lpOverlap – указатель на структуру OVERLAPPED, передается в том случае, если *hF* - асинхронный. В этом случае в структуре OVERLAPPED в полях *Offset* и *OffsetHigh* предварительно устанавливается байтовое смещение, с которого будет осуществляться чтение файла.

Если операция успешна, возвращаемое значение – не ноль.

4. Запись в файл:

```
«BOOL WINAPI WriteFile(
        HANDLE hF,
        LPCVOID lpBuf,
        DWORD nNumBytesToWrite,
        LPDWORD lpNumBytesWritten,
        LPOVERLAPPED lpOverlap
);»,
```

где *hF* – дескриптор файла/устройства;

lpBuf – указатель на буфер с данными для записи в файл/устройство;

nNumberBytesToWrite – количество записываемых байтов;

lpNumberBytesWritten – указатель на переменную, получающую количество записанных байтов при синхронном дескрипторе *hF*, при асинхронном дескрипторе (когда *hF* был открыт с флагом FILE_FLAG_OVERLAPPED) следует использовать NULL. Параметр может быть NULL, только если *lpOverlap* не NULL;

lpOverlap – указатель на структуру OVERLAPPED, передается в том случае, если *hF* – асинхронный. В этом случае в структуре OVERLAPPED в полях *Offset* и *OffsetHigh* предварительно устанавливается байтовое смещение, с которого будет осуществляться запись в файл.

Если операция успешна, возвращаемое значение – не ноль.

5. Перемещение указателя файла:

```
«DWORD WINAPI SetFilePointer(
        HANDLE hF,
        LONG lDistanceMove,
        PLONG lpDistanceMoveHigh,
        DWORD dwMoveMeth
);»,
```

где *hF* – дескриптор файла, который должен быть создан с правами GENERIC_READ или GENERIC_WRITE;

lDistanceMove – младшие 32 разряда целого со знаком, указывающие количество байт, на которые нужно переместить указатель файла; положительное значение перемещает указатель вперед, отрицательное – назад; если *lpDistanceMoveHigh* – не NULL, *lpDistanceMoveHigh* и *lDistanceMove* – 64-битные целые со знаком, в противном случае – *lDistanceMove* является 32-битным целым со знаком;

lpDistanceMoveHigh – указатель на старшие 32 разряда 64-битного расстояния для перемещения указателя;

dwMoveMeth – начало для перемещения указателя:

начало файла – FILE_BEGIN(0), текущая позиция – FILE_CURRENT(1), конец файла – FILE_END(2).

Если операция успешна и *lpDistanceMoveHigh* – NULL, функция возвращает младшее слово новой величины указателя файла, если же *lpDistanceMoveHigh* – не NULL, в *lpDistanceMoveHigh* дополнительно записывается старшее слово новой величины указателя; если операция неуспешна, функция возвращает INVALID_SET_FILE_POINTER.

6. Получение атрибутов файла:

```
«DWORD WINAPI GetFileAttributes(
        LPCTSTR lpFName
);»,
```

где *lpFName* – имя файла или каталога.

Если операция успешна, функция возвращает атрибуты файла/каталога в виде целого числа, иначе – INVALID_FILE_ATTRIBUTES. Чтобы узнать, установлен ли конкретный атрибут, результат следует побитово умножить («&») на одно из значений атрибутов: FILE_ATTRIBUTE_ARCHIVE (архивный), FILE_ATTRIBUTE_HIDDEN (скрытый), FILE_ATTRIBUTE_NORMAL (не имеющий иных атрибутов), FILE_ATTRIBUTE_READONLY (только для чтения), FILE_ATTRIBUTE_SYSTEM (используется ОС), FILE_ATTRIBUTE_-TEMPORARY (используется для временного хранения), FILE_ATTRIBUTE_-COMPRESSED (сжатый), FILE_ATTRIBUTE_DIRECTORY (каталог), FILE_ATTRIBUTE_ENCRYPTED (зашифрованный), FILE_ATTRIBUTE_-NOT_CONTENT_INDEXED (неиндексированный) и др.

7. Установка атрибутов файла:

```
«BOOL WINAPI SetFileAttributes(
       LPCTSTR lpFName,
       DWORD dwFAttributes
);»,
```

где *lpFName* – имя файла или каталога, для которого устанавливаются атрибуты;

dwFAttributes – один или несколько устанавливаемых атрибутов файла из числа первых шести вышеперечисленных, а также некоторых других. Одновременная установка атрибутов выполняется с использованием побитового «или» («|»), с атрибутом FILE_ATTRIBUTE_NORMAL никакие другие атрибуты не устанавливаются.

Если операция успешна, функция возвращает не ноль, иначе – ноль.

Контрольные вопросы

1. Аргументы командной строки.
2. Функции управления файлами.
3. Создание, открытие и закрытие файла.
4. Чтение файла.
5. Запись в файл.
6. Перемещение указателя файла.
7. Получение и установка атрибутов файла.
8. Сжатые и зашифрованные файлы.

Лабораторная работа № 2

Создание процессов и перенаправление ввода/вывода

Цель работы

Получение базовых навыков создания и управления процессами с помощью функций Windows API, создания наследуемых дескрипторов объектов и реализации наследования дескрипторов, перенаправления стандартного ввода/вывода дочерних процессов на файлы.

Задание кафедры

Написать программу, запускающую несколько внешних программ (порождающую несколько процессов) в определенном порядке, а также создающую наследуемые дескрипторы файлов исходного процесса и перенаправляющую дескрипторы ввода/вывода первых трех дочерних процессов на файлы исходного процесса. При этом число процессов и порядок их создания, способ перенаправления (P: 1 – изменения в STARTUPINFO, 2 – с помощью *GetStdHandle*/*SetStdHandle*), порядок перенаправления (Q: 1 – ввод, вывод, ввод/вывод, 2 – вывод, ввод/вывод, ввод, 3 – ввод/вывод, ввод, вывод) выбираются из табл. 2 прил.).

Пути к внешним программам задаются в качестве аргументов командной строки.

Теоретические сведения

1. Создание процесса (функция создает процесс и его главный поток):

```
«BOOL WINAPI CreateProcess(
       LPCTSTR lpApplicName,
       LPTSTR lpComLine,
       LPSECURITY_ATTRIBUTES lpProcAttributes,
       LPSECURITY_ATTRIBUTES lpThrAttributes,
       BOOL bInhHandles,
       DWORD dwCreatFlags,
       LPVOID lpEnvir,
       LPCTSTR lpCurDirectory,
       LPSTARTUPINFO lpStartInfo,
       LPPROCESS_INFORMATION lpProcInformation
);»,
```

где *lpApplicName* – указатель на строку с именем запускаемого модуля (программы). Строка может содержать полный путь и имя файла или сокращенное имя (всегда следует указывать расширение). В последнем случае функция использует текущие диск и каталог для завершения определения. Данный параметр может быть равен NULL, в этом случае имя запускаемого модуля должно быть указано в первом слове строки, на которую указывает *lpComLine* (слова разделяются пробелами). Если имя файла содержит пробелы, следует заключать его в кавычки. Если запускаемый модуль – 16-битное приложение, параметр *lpApplicName* должен быть равен NULL, а строка, на которую указывает *lpComLine*, должна содержать имя модуля и его аргументы;

lpComLine – указатель на командную строку для выполнения. Максимальная длина –32768, включая символ конца строки. Если параметр *lpApplicName* равен NULL, длина имени модуля ограничивается MAX_PATH. Для UNICODE-версии функции данный параметр не может быть указателем на память «только для чтения». Если *lpComLine* – NULL, функция использует в качестве командной строки строку, на которую указывает *lpApplicName*.

Если *lpApplicName* – NULL, первое слово строки, на которую указывает *lpComLine*, является именем загружаемого модуля: оно может быть указано без расширения, если модуль – exe-файл. Если имя модуля не содержит путь, система ищет загружаемый модуль, последовательно просматривая:

– каталог, в котором загружено приложение;

– текущий каталог родительского процесса;

– 32-разрядный системный каталог Windows. Для получения пути к этому каталогу можно использовать функцию *GetSystemDirectory*;
– 16-разрядный системный каталог Windows («System»);
– каталог Windows, путь к которому можно получить, используя функцию *GetWindowsDirectory*;
– каталоги, перечисленные в переменной окружения PATH;
lpProcAttributes – указатель на структуру SECURITY_ATTRIBUTES, определяющую, будет ли дескриптор нового процесса наследоваться дочерними процессами. Если значение равно NULL, дескриптор не наследуется.
lpThrAttributes – указатель на структуру SECURITY_ATTRIBUTES, определяющую, будет ли дескриптор нового (главного) потока наследоваться дочерними процессами. Если значение равно NULL, дескриптор не наследуется. Поле *lpSecurityDescriptor* упомянутой структуры определяет дескриптор безопасности главного потока;
bInhHandles – если данный параметр установить в TRUE, каждый наследуемый дескриптор вызывающего процесса будет унаследован новым процессом, если – FALSE, дескрипторы не наследуются. Унаследованные дескрипторы имеют те же значения и права доступа, что и исходные;
dwCreatFlags – эти флаги управляют классом приоритета и созданием процесса (например, CREATE_NEW_CONSOLE – с новой консолью, CREATE_NO_-WINDOW – без консоли для консольных процессов и др.). Если класс приоритета не указан, процессу назначается класс NORMAL_PRIORITY_CLASS, несмотря на то, что приоритет родительского процесса может быть ниже;
lpEnvir – указатель на блок переменных окружения нового процесса, если NULL, новый процесс использует окружение вызывающего процесса.

Блок переменных окружения является перечнем строк вида: «имя=значение\0», оканчивающихся нулевым символом; сам блок также оканчивается нулевым символом;
lpCurDirectory – полный путь к текущему каталогу процесса. Если этот параметр равен NULL, новый процесс получает те же диск и каталог, что и исходный;

lpStartInfo – указатель на структуру STARTUPINFO или STARTUPINFOEX structure. Для установки расширенных атрибутов можно использовать STARTUPINFOEX (минимальный клиент – Windows Vista), указывая EXTENDED_STARTUPINFO_PRESENT в параметре *dwCreatFlags*. Дескрипторы в STARTUPINFO или STARTUPINFOEX должны быть закрыты с помощью *CloseHandle*, когда они больше не нужны;

lpProcInformation – указатель на структуру PROCESS_INFORMATION, получающую информацию по идентификации нового процесса (и его главного потока).

Если операция успешна, функция возвращает не ноль, иначе – ноль.

2. Ожидание перехода объекта в сигнальное состояние или истечения интервала таймаута (в частности, ожидание завершения одного процесса):

```
«DWORD WINAPI WaitForSingleObject(
        HANDLE hObj,
        DWORD dwMlseconds
);»,
```

где *hObj* – дескриптор объекта, у которого должны быть права доступа SYNCHRONIZE;

dwMlseconds – интервал таймаута в миллисекундах. Если значение *dwMlseconds* равно 0, сразу происходит возврат из функции, если значение *dwMlseconds* равно INFINITE, возврат из функции произойдет только тогда, когда объект перейдет в сигнальное состояние.

Функция возвращает одно из значений: WAIT_OBJECT_0 (объект перешел в сигнальное состояние), WAIT_TIMEOUT (таймаут истек, объект находится в несигнальном состоянии), WAIT_ABANDONED (объект является мьютексом, не освобожденным потоком, который уже завершился, т.е. покинутым мьютексом, владение мьютексом перешло к вызывающему потоку, и мьютекс находится в несигнальном состоянии), WAIT_FAILED (функция завершилась неуспешно).

3. Ожидание перехода нескольких объектов или одного из нескольких в сигнальное состояние или истечения интервала таймаута (в частности, ожидание завершения нескольких процессов или одного из нескольких процессов):

«DWORD WINAPI WaitForMultipleObjects(
        DWORD dwnCount,
        const HANDLE *lpHandl,
        BOOL bWaitAllObj,
        DWORD dwMlseconds
);»,

где *dwnCount* – количество дескрипторов объектов, максимум – MAXIMUM_WAIT_OBJECTS;

lpHandl – указатель на массив дескрипторов объектов, у которых должны быть права доступа SYNCHRONIZE;

bWaitAllObj – если этот параметр установлен в TRUE, возврат из функции происходит, когда состояние всех объектов становится сигнальным, если же параметр сброшен в FALSE, возврат из функции выполняется, когда состояние любого из объектов становится сигнальным. В последнем случае возвращаемое значение относится к объекту, состояние которого повлекло возврат из функции;

dwMlseconds – интервал таймаута в миллисекундах. Если значение *dwMlseconds* равно 0, сразу происходит возврат из функции, если значение *dwMlseconds* – INFINITE, возврат из функции произойдет только тогда, когда объекты перейдут в сигнальное состояние.

Функция возвращает одно из значений:

– от WAIT_OBJECT_0 до (WAIT_OBJECT_0 + *dwnCount* – 1) – если *bWaitAllObj* – TRUE, объекты перешли в сигнальное состояние. Если же *bWaitAllObj* – FALSE, то возвращенное значение минус WAIT_OBJECT_0 является индексом объекта [в массиве дескрипторов], который инициировал возврат (перешел в сигнальное состояние); если таких объектов несколько, то указанная величина является наименьшим из возможных индексов;

– WAIT_TIMEOUT – таймаут истек;

– от WAIT_ABANDONED_0 до (WAIT_ABANDONED_0 + *dwnCount* – 1) – если *bWaitAllObj* – TRUE, состояние всех объектов – сигнальное, и, как минимум, один из объектов является покинутым мьютексом. Если же *bWaitAllObj* – FALSE, то возвращенное значение минус WAIT_OBJECT_0 является индексом покинутого мьютекса, который инициировал возврат; владение мьютексом

перешло к вызывающему потоку, и мьютекс находится в несигнальном состоянии;

– WAIT_FAILED – функция завершилась неуспешно.

4. Получение дескриптора стандартного устройства:

```
«HANDLE WINAPI GetStdHandle(
         DWORD StdHandle
);»,
```

где *StdHandle* – стандартное устройство ввода, вывода или ошибок: STD_INPUT_HANDLE (входной буфер консоли), STD_OUTPUT_HANDLE (буфер экрана консоли), STD_ERROR_HANDLE (буфер экрана консоли).

Если операция успешна, функция возвращает дескриптор устройства, возможно, перенаправленный, иначе – INVALID_HANDLE_VALUE.

5. Установка дескриптора стандартного устройства:

```
«HANDLE WINAPI SetStdHandle(
         DWORD StdHandle,
         HANDLE hObj
);»,
```

где *StdHandle* – стандартное устройство (см. выше);

hObj – устанавливаемый дескриптор.

Если операция успешна, возвращаемое значение – не ноль.

Для того чтобы сделать дескриптор какого-либо объекта наследуемым, следует определить переменную типа SECURITY_ATTRIBUTES, установить в TRUE поле *bInheritHandle*, в поле *nLength* записать размер переменной, передать переменную (чаще указатель на нее) в качестве параметра в функцию, возвращающую дескриптор создаваемого объекта.

Пример – создание наследуемого дескриптора файла hFile:

```
«SECURITY_ATTRIBUTES sattr = { sizeof(sattr), NULL, TRUE};
  HANDLE hFile =CreateFile(…, &sattr, …);» .
```

Тот факт, что дескриптор потенциально является наследуемым, еще не означает, что он будет унаследован. В частности, при создании процесса с помощью *CreateProcess* наследуемые дескрипторы родительского процесса будут унаследованы дочерним, если установить в TRUE параметр *bInhHandles*.

Чтобы перенаправить ввод/вывод порождаемого процесса, можно изменить некоторые поля в структуре STARTUPINFO, предварительно обнулив ее

перед вызовом *CreateProcess*. После обнуления следует определить значения полей дескрипторов стандартных устройств структуры STARTUPINFO – *hStdInput, hStdOutput, hStdError* (например, присвоить соответствующие дескрипторы открытых файлов родительского процесса), при этом поле *dwFlags* структуры должно быть установлено в STARTF_USESTDHANDLES.

Другой способ – изменение дескрипторов стандартных устройств родительского процесса перед вызовом *CreateProcess* и после него. Для получения/установки дескрипторов стандартных устройств можно воспользоваться приведенными ниже функциями (минимальный клиент – Windows 2000 Prof.).

Контрольные вопросы

1. Создание и завершение процесса.
2. Ожидание завершения (освобождения) одного объекта.
3. Ожидание завершения (освобождения) нескольких объектов.
4. Дескрипторы стандартных устройств, наследование дескрипторов.
5. Перенаправление стандартного ввода/вывода на файлы.
6. Перенаправление с помощью анонимных и именованных каналов.

Лабораторная работа № 3

Реализация интерпретатора команд ОС

Цель работы

Изучение способов построения и получение навыков реализации интерпретаторов команд для многозадачных операционных систем.

Задание кафедры

Реализовать упрощенный интерпретатор команд Windows, выполняющий внешние команды, обрабатывающий одну встроенную команду и реализующий перенаправление ввода/вывода. В качестве встроенной команды следует использовать программу, реализованную в лабораторной работе № 1 (ее нужно оформить как функцию).

Теоретические сведения

Интерпретатор команд позволяет запускать программы и выполнять требуемую обработку данных пользователя. В качестве входных данных программа-интерпретатор получает строку символов, содержащую команду с аргументами и (или) управляющие символы. Команда может включать путь к внешней программе или являться внутренней, встроенной, командой, которая обрабатывается непосредственно интерпретатором. В первом случае интерпретатор создает новый процесс, который соответствует распознанной команде.

Обобщенный алгоритм работы упрощенного интерпретатора может предполагать последовательное (если не указаны переходы) выполнение следующих шагов:

1. Ожидать ввод строки символов.

2. Считать и осуществить интерпретацию (разбор) строки.

3. Если строка содержит ошибочные данные, вывести сообщение об ошибке и перейти на шаг 1.

4. *Если* строка содержит имя внешней программы и в ней нет управляющих символов, создать новый процесс, соответствующий этой программе (*CreateProcess*), и ожидать (*WaitForSingleObject*), пока он не завершится, выполнить переход на шаг 1;

иначе, если строка содержит имя внешней программы и в ней присутствуют управляющие символы, реализовать управление (выполнить перенаправление ввода/вывода на файлы), запустить программу, породив новый процесс (*CreateProcess*), и ожидать (*WaitForSingleObject*), пока он не завершится; при необходимости восстановить стандартные потоки ввода/вывода, выполнить переход на шаг 1;

иначе, если строка содержит имя встроенной команды, выполнить ее и перейти на шаг 1;

иначе, если строка содержит комбинацию символов, используемую для выхода из интерпретатора, завершить работу интерпретатора.

Контрольные вопросы

1. Открытие, чтение, запись и закрытие файлов.

2. Стандартные устройства и консольный ввод-вывод.
3. Создание процесса, идентификация процесса.
4. Выход из процесса и его завершение.
5. Перенаправление потоков ввода-вывода.

Лабораторная работа № 4

Основы работы с потоками

Цель работы

Получение навыков создания многопоточных приложений в ОС Windows.

Задание кафедры

Написать программу параллельной многопоточной генерации и обработки элементов строк двумерной матрицы (каждый поток генерирует одну строку и осуществляет ее обработку) с выводом в дополнительном потоке результата обработки (вид операции и размерность матрицы выбираются из табл. 3 прил.). В каждый поток, выполняющий генерацию и обработку, следует передавать в качестве параметра указатель на структуру, поля которой соответствуют строке матрицы и результату обработки.

Теоретические сведения

В контексте процесса могут выполняться один или несколько потоков. Поток (в современных ОС Windows) – это единица выполнения, объект, которому ОС выделяет процессорное время.

1. Создание потока для запуска в виртуальном адресном пространстве вызывающего процесса:

«
```
HANDLE WINAPI CreateThread(
         LPSECURITY_ATTRIBUTES lpThrAttributes,
         SIZE_T dwStackSz,
         LPTHREAD_START_ROUTINE lpStartAddr,
         LPVOID lpParam,
         DWORD dwCreatFlags,
         LPDWORD lpThrId
);
```
»,

где *lpThrAttributes* – указатель на структуру SECURITY_ATTRIBUTES, если задать NULL, возвращаемый дескриптор не будет наследуемым;

dwStackSz – первоначальный размер стека в байтах. Система округляет это значение до ближайшей страницы. Если этот параметр равен нулю, новый поток использует размер по умолчанию;

lpStartAddr – указатель на определенную в программе функцию, которая загружается потоком, стартовый адрес потока; функция потока *ThFunc* имеет сигнатуру: DWORD WINAPI *ThFunc*(LPVOID);

lpParam – указатель на переменную, передаваемую потоку;

dwCreatFlags – флаги, управляющие созданием потока: 0 (поток запускается сразу после создания), CREATE_SUSPENDED (поток приостанавливается после создания до тех пор, пока не выполнится функция *ResumeThread*), STACK_SIZE_PARAM_IS_A_RESERVATION (параметр *dwStackSz* задает первоначальный зарезервированный размер стека, если этот параметр не установлен, *dwStackSz* задает выделенный размер стека);

lpThrId – указатель на переменную, получающую идентификатор потока.

2. Завершение потока (поток завершится при явном вызове этой функции или при возврате из функции потока):

```
«VOID WINAPI ExitThread(
       DWORD dwExCode
);»,
```

где *dwExCode* – код завершения потока.

3. Приостановка указанного потока:

```
«DWORD WINAPI SuspendThread(
        HANDLE hThr
);»,
```

где *hThr* – дескриптор приостанавливаемого потока, который должен иметь права доступа THREAD_SUSPEND_RESUME.

Если операция успешна, возвращаемое значение – предыдущее количество приостановок потока, в противном случае – -1.

4. Возобновление выполнения указанного потока или уменьшение количества приостановок, если оно больше нуля:

```
«DWORD WINAPI ResumeThread(
        HANDLE hThr
);»,
```

где *hThr* – дескриптор потока, который должен иметь права доступа THREAD_SUSPEND_RESUME.

Если операция успешна, возвращаемое значение – предыдущее количество приостановок потока, в противном случае – -1.

Контрольные вопросы

1. Потоки. Реализация на уровне ядра и на уровне пользователя.
2. Создание и завершение потока.
3. Приостановка и возобновление выполнения потока.
4. Локальные данные потока.
5. Удаленные потоки, приоритеты потоков.

Лабораторная работа № 5

Реализация решения задачи «читателей и писателей» с использованием мьютексов (двоичных семафоров)

Цель работы

Изучение способов и получение навыков реализации решения одной из классических задач межпроцессного взаимодействия с помощью мьютексов.

Задание кафедры

Реализовать решение задачи «читателей и писателей» с использованием мьютексов в следующем виде: написать многопоточную программу, в которой есть несколько потоков-читателей и потоков-писателей. Поток-писатель генерирует заданное количество данных и после получения доступа к буферу помещает их случайным образом в буфер (позиции, в которые помещаются данные, генерируются случайно). Во время записи никакой другой поток не может получить доступ к буферу. Потоки-читатели считывают все данные из буфера и выводят на экран, возможно, параллельно. Как только один поток-читатель из очереди на чтение получил доступ к буферу, все остальные потоки-читатели сразу получают к нему доступ (количество потоков-читателей А, потоков-писателей Б, генерируемых данных М и размер буфера N заданы в табл. 4 прил. и выбираются в соответствии с вариантом).

Теоретические сведения

Один из способов реализации взаимного исключения в Windows состоит в использовании мьютекса – объекта синхронизации, который может быть либо свободен, либо захвачен каким-либо потоком. В период владения мьютексом поток имеет единоличный доступ к разделяемому ресурсу. При работе с мьютексами используются функции создания или открытия мьютекса, ожидания на мьютексе (или захвата мьютекса) и освобождения мьютекса.

1. Создание или открытие именованного или неименованного мьютекса:

```
«HANDLE WINAPI CreateMutex(
        LPSECURITY_ATTRIBUTES lpMutexAttrib,
        BOOL bInitOwner,
        LPCTSTR lpNm
);»,
```

где *lpMutexAttrib* – указатель на структуру SECURITY_ATTRIBUTES, если задать NULL, возвращаемый дескриптор не будет наследуемым;

bInitOwner – если параметр установлен в TRUE и вызывающий функцию поток создает мьютекс, то он одновременно с созданием захватывает мьютекс. В противном случае вызывающий поток не захватывает мьютекс;

lpNm – имя мьютекса, длина которого ограничивается MAX_PATH символами. Если *lpNm* соответствует имени существующего именованного мьютекса, функция требует права доступа MUTEX_ALL_ACCESS. В этом случае параметр *bInitOwner* игнорируется. При этом если параметр *lpMutexAttrib* – не NULL, он определяет, будет ли дескриптор наследуемым, но дескриптор безопасности из LPSECURITY_ATTRIBUTES игнорируется. Если *lpNm* – NULL, создается неименованный мьютекс.

Если операция успешна, возвращаемое значение – дескриптор вновь созданного мьютекса, в противном случае функция возвращает NULL. Если мьютекс является именованным и уже существующим на момент вызова функции, возвращаемое значение – дескриптор существующего мьютекса, параметр *bInitOwner* игнорируется, вызывающий поток не может сразу захватить мьютекс.

2. Открытие именованного мьютекса:

```
«HANDLE WINAPI OpenMutex(
```

```
        DWORD dwDesAccess,
        BOOL bInhHandle,
        LPCTSTR lpNm
);»,
```

где *dwDesAccess* – доступ к мьютексу. Для использования мьютекса требуется только право доступа SYNCHRONIZE; для изменения безопасности мьютекса указывают MUTEX_ALL_ACCESS;

bInhHandle – если этот параметр установлен в TRUE, дескриптор мьютекса будет наследуемым;

lpNm – имя открываемого мьютекса;

В случае успеха возвращается дескриптор мьютекса, иначе – NULL.

3. Освобождение мьютекса:

```
«BOOL WINAPI ReleaseMutex(
       HANDLE hMut
);»,
```

где *hMut* – дескриптор мьютекса.

Если операция успешна, то возвращаемое значение – не ноль, иначе – ноль.

Контрольные вопросы

1. Критические участки кода.
2. Синхронизация потоков.
3. Классические задачи межпроцессного взаимодействия.
4. Создание и открытие мьютекса.
5. Освобождение мьютекса.

Лабораторная работа № 6

Реализация решения задачи «производитель-потребитель» с использованием семафоров

Цель работы

Получение навыков практического применения семафоров для решения одной из классических задач межпроцессного взаимодействия.

Задание кафедры

Реализовать с помощью семафоров решение задачи «производитель-потребитель», которая сформулирована в следующем виде: написать программу-клиент – потребитель данных из кольцевого буфера и программу-сервер – генератор (производитель) данных в буфер. В кольцевом буфере можно выделить первую занятую, последнюю занятую, далее – первую свободную и затем – последнюю свободную позиции. Сервер генерирует заданное количество данных и после получения доступа к буферу помещает их в буфер, начиная с первой свободной позиции, если это возможно. В противном случае он ждет, когда освободится место в буфере. Клиент забирает заданное количество данных из буфера, освобождая соответствующие позиции в буфере, начиная с первой занятой, если это возможно. В противном случае клиент ждет, когда сервер поместит достаточное количество данных в буфер (количество данных M, генерируемых сервером, количество данных N, считываемых клиентом, размер буфера K заданы в табл. 5 прил.).

Теоретические сведения

Семафоры отличаются от мьютексов тем, что поддерживают счетчики, которые могут изменяться от нуля до некоторого максимального значения. Поток, выполняющий ожидание на семафоре, декрементирует значение счетчика. Поток, освобождающий семафор, – инкрементирует его значение. Если счетчик семафора больше нуля, семафор находится в сигнальном состоянии, если же счетчик равен нулю, состояние семафора является несигнальным. Выполнение функции ожидания на семафоре, находящемся в несигнальном состоянии, блокирует поток. Функции управления семафорами аналогичны функциям управления мьютексами.

1. Создание или открытие именованного или неименованного семафора:

```
«HANDLE WINAPI CreateSemaphore(
         LPSECURITY_ATTRIBUTES lpSemAttributes,
         LONG lInitCount,
         LONG lMaxCount,
         LPCTSTR lpNm
);»,
```

где *lpSemAttributes* – указатель на структуру SECURITY_ATTRIBUTES, если задать NULL, возвращаемый дескриптор не будет наследуемым;

lInitCount – начальное значение счетчика для семафора, оно должно быть больше или равно нулю и меньше или равно *lMaxCount*. Когда это значение больше нуля, состояние семафора – сигнальное, когда оно равно нулю, состояние семафора – несигнальное. Значение счетчика уменьшается на единицу функцией ожидания, освобождающей поток, который ждал на данном семафоре. Значение счетчика увеличивается на указанное количество вызовом *ReleaseSemaphore*;

lMaxCount – максимальное значение счетчика семафора, которое должно быть больше нуля;

lpNm – имя семафора, длина которого ограничивается MAX_PATH символами. Если *lpNm* соответствует имени существующего именованного семафора, функция требует права доступа SEMAPHORE_ALL_ACCESS. В этом случае параметры *lInitCount* и *lMaxCount* игнорируются, так как они уже были установлены ранее, при создании семафора. При этом если параметр *lpSemAttributes* – не NULL, он определяет, будет ли дескриптор наследуемым, но дескриптор безопасности из LPSECURITY_ATTRIBUTES игнорируется. Если *lpNm* – NULL, создается неименованный семафор.

Если операция успешна, возвращаемое значение – дескриптор вновь созданного семафора, в противном случае функция возвращает NULL. Если семафор является именованным и уже существующим на момент вызова функции, возвращаемое значение – дескриптор существующего семафора.

2. Открытие именованного семафора:

```
«HANDLE WINAPI OpenSemaphore(
         DWORD dwDesAccess,
         BOOL bInhHandle,
         LPCTSTR lpNm
);»,
```

где *dwDesAccess* – доступ к семафору. Для использования семафора требуется право доступа SYNCHRONIZE; для изменения безопасности семафора указывают SEMAPHORE_ALL_ACCESS;

bInhHandle – если этот параметр установлен в TRUE, дескриптор семафора будет наследуемым;

lpNm – имя открываемого семафора.

Если операция успешна, результат – дескриптор семафора, иначе – NULL.

3. Освобождение семафора:

```
«BOOL WINAPI ReleaseSemaphore(
       HANDLE hSem,
       LONG lRelCount,
       LPLONG lpPrevCount
);»,
```

где *hSem* – дескриптор семафора, который должен иметь право доступа SEMAPHORE_MODIFY_STATE;

lRelCount – количество, на которое увеличится счетчик семафора, которое должно быть больше нуля. Если указанное количество превысит максимальное значение счетчика семафора, оно не изменится, и функция вернет ноль;

lpPrevCount – указатель на переменную, в которую запишется предыдущее значение счетчика семафора, может быть NULL;

Если операция успешна, то возвращаемое значение – не ноль, иначе – ноль.

Контрольные вопросы

1. Объекты синхронизации процессов и потоков.
2. Взаимные исключения, критические секции.
3. Классические задачи межпроцессного взаимодействия.
4. Создание и открытие семафора.
5. Освобождение семафора.
6. События.

Лабораторная работа № 7

Реализация взаимодействия процессов на основе разделяемой памяти

Цель работы

Изучение принципов создания клиент-серверных приложений с организацией взаимодействия процессов через разделяемую память в многозадачных ОС и получение навыков практического использования средств системного

программного обеспечения для синхронизации процессов в условиях совместного доступа к разделяемой памяти.

Задание кафедры

Реализовать программу-сервер и группу программ-клиентов, взаимодействующих с сервером с помощью механизма разделяемой памяти. Каждая программа-клиент обращается к серверу для реализации запроса на выполнение функции, заполняя поля структуры запроса в блоке разделяемой памяти. Сервер обрабатывает запрос, выдает результат, помещая его в поле результата структуры запроса в разделяемой памяти. Если сервер занят выполнением запроса и очередь клиентов на обслуживание не заполнена, клиент ставится в конец очереди. Клиент «уходит» (заканчивает работу), если обнаруживает, что очередь заполнена (функцию и максимальное количество клиентов в очереди N следует взять из табл. 6 прил.).

Теоретические сведения

Для организации разделения памяти в ОС Windows предусмотрен механизм отображения (проецирования) файлов на память, позволяющий ассоциировать файл с виртуальным адресным пространством процесса. Его использование предполагает создание или открытие объекта отображения файла для предварительно открытого файла с помощью функций *CreateFileMapping*, *OpenFileMapping* и реализацию отображения посредством вызова *MapViewOfFile*.

1. Создание объекта отображения файла:

«
```
HANDLE WINAPI CreateFileMapping(
         HANDLE hF,
         LPSECURITY_ATTRIBUTES lpAttrib,
         DWORD Protect,
         DWORD dwMaxSizeHigh,
         DWORD dwMaxSizeLow,
         LPCTSTR lpNm
);
```
»,

где *hF* – дескриптор файла, с которого создается объект отображения файла. Файл должен быть открыт с правами доступа, совместимыми с флагами защиты параметра *Protect*. Если задать hF равным INVALID_HANDLE_VALUE, вызывающий процесс должен также указать размеры для объекта файлового отоб-

ражения в параметрах *dwMaxSizeHigh* and *dwMaxSizeLow*. В этом случае функция создает объект отображения для участка памяти в системном файле подкачки;

lpAttrib – указатель на структуру SECURITY_ATTRIBUTES, определяющую, будет ли возвращаемый дескриптор наследоваться дочерними процессами. Если значение равно NULL, дескриптор не наследуется;

Protect – определяет страничную защиту для создаваемого объекта. Все отображенные области объекта должны быть совместимы с этой защитой. Параметр может принимать следующие значения (можно комбинировать эти значения с дополнительными атрибутами [1]):

- PAGE_EXECUTE_READ – только чтение, копирование по записи (измененные страницы записываются только в файл подкачки), загрузка (дескриптор отображаемого файла должен быть создан с правами GENERIC_READ и GENERIC_EXECUTE);
- PAGE_EXECUTE_READWRITE – только чтение, копирование по записи, загрузка, чтение/запись;
- PAGE_READONLY – только чтение, копирование по записи;
- PAGE_READWRITE – только чтение, копирование по записи, чтение/запись и др.;

dwMaxSizeHigh – старшие 32 разряда значения максимального размера объекта отображения файла;

dwMaxSizeLow – младшие 32 разряда значения максимального размера объекта отображения файла. Если параметры *dwMaxSizeHigh* и *dwMaxSizeLow* равны нулю, максимальный размер объекта отображения равен текущему размеру отображаемого файла;

lpNm – имя объекта отображения файла. Если параметр равен NULL, создается неименованный объект отображения файлов.

Если операция успешна, возвращается созданный или полученный дескриптор, иначе – NULL.

2. Открытие именованного объекта отображения файла:

«HANDLE WINAPI OpenFileMapping(

```
        DWORD dwDesAccess,
        BOOL bInhHandle,
        LPCTSTR lpNm
);»,
```

где *dwDesAccess* – доступ к объекту отображения файла, который должен соответствовать флагам параметра *Protect* функции *CreateFileMapping*: FILE_MAP_ALL_ACCESS (все права доступа, кроме FILE_MAP_EXECUTE), FILE_MAP_EXECUTE (загружаемый объект отображения), FILE_MAP_READ (только чтение, копирование по записи), FILE_MAP_WRITE (только чтение, копирование по записи, чтение/запись);

bInhHandle – если этот параметр установлен в TRUE, дескриптор объекта будет наследуемым;

lpNm – имя открываемого объекта отображения файла.

Если операция успешна, функция возвращает открытый дескриптор объекта отображения файла, иначе – NULL.

3. Отображение области файла на адресное пространство процесса:

```
«LPVOID WINAPI MapViewOfFile(
        HANDLE hFMappingObject,
        DWORD dwDesAccess,
        DWORD dwFOffsetHigh,
        DWORD dwFOffsetLow,
        SIZE_T dwNumOfBytesToMap
);»,
```

где *hFMappingObject* – дескриптор объекта отображения файла;

dwDesAccess – тип доступа к объекту отображения файла, определяющий защиту страниц. Может принимать значения: FILE_MAP_ALL_ACCESS (чтение/запись, эквивалентно FILE_MAP_WRITE), FILE_MAP_COPY (копирование по записи), FILE_MAP_READ (только чтение), FILE_MAP_WRITE (чтение/запись); каждая из перечисленных констант может использоваться с FILE_MAP_EXECUTE (загрузка);

dwFOffsetHigh – старшие 32 разряда значения смещения начала области отображения;

dwFOffsetLow – младшие 32 разряда значения смещения начала области отображения;

dwNumOfBytesToMap – размер области отображения в байтах; если параметр задать равным нулю, отображение будет выполнено с указанного смещения до конца отображаемого файла.

Если операция успешна, возвращается начальный адрес области отображения (указатель), в противном случае – NULL.

4. Отмена отображения (освобождение памяти) области файла на адресное пространство процесса:

```
«BOOL WINAPI UnmapViewOfFile(
       LPCVOID lpBaseAddr
);»,
```

где *hFBaseAddr* – указатель на базовый адрес области отображения в виртуальном адресном пространстве процесса.

Если операция успешна, возвращаемое значение – не ноль, иначе – ноль.

При выполнении лабораторной работы, помимо мьютексов и семафоров, могут использоваться события. Событие – это объект синхронизации, состояние которого явно устанавливается в сигнальное с помощью функции *SetEvent*. Windows поддерживает два вида событий:

– сбрасываемое вручную событие – такое событие, состояние которого остается сигнальным до тех пор, пока оно не будет сброшено явно в несигнальное с помощью функции *ResetEvent*; оно может сигнализировать одновременно всем потокам, ожидающим его наступления;

– автоматически сбрасываемое событие – событие, остающееся в сигнальном состоянии до тех пор, пока один из ожидающих его потоков не будет освобожден, после чего состояние автоматически сбросится в несигнальное; если нет ожидающих потоков, событие остается в сигнальном состоянии.

Далее представлены лишь некоторые аспекты использования функций, управляющих событиями, так как они аналогичны описанным выше функциям для семафоров и мьютексов. Функции *SetEvent* и *ResetEvent* принимают в качестве параметра дескриптор события и возвращают TRUE в случае успеха, FALSE в случае неудачи.

Функция создания или открытия именованного или неименованного события *CreateEvent* включает параметры:

bManualReset – признак вида: если TRUE, создается сбрасываемое вручную событие, иначе – автоматически сбрасываемое;

bInitialState – признак начального состояния: если TRUE, начальное состояние – сигнальное, иначе – нет; описание остальных параметров и возвращаемого значения аналогично представленному для мьютексов/семафоров.

Контрольные вопросы

1. Объекты синхронизации Windows: критические секции, мьютексы, семафоры, события.
2. Объекты синхронизации POSIX: семафоры, условные переменные.
3. Управление памятью. Файлы, отображаемые в память.
4. Создание объекта отображения файлов в Windows.
5. Отображение адресного пространства процесса в объекты отображения.
6. Схема взаимодействия процессов через разделяемую память.

Библиографический список

1. Microsoft Developer Network [Эл. ресурс]. – Режим доступа: msdn.microsoft.com.

2. Харт, Дж. М. Системное программирование в среде Windows [Текст] / Джонсон М. Харт. – М.: Вильямс, 2005. – 592 с.

3. Руссинович, М. Внутреннее устройство Microsoft Windows [Текст] / М. Руссинович, Д. Соломон. – СПб.: Питер, 2013. – 800 с.

4. Гордеев, А.В. Операционные системы [Текст]: учебн. для вузов / А.В. Гордеев. – СПб.: Питер, 2009. – 416 с.

Приложение

Таблица 1

Задания к лабораторной работе № 1

№ вар.	Количество строк, *n*	Номера строк	Позиция в результирующем файле	Установка атрибутов
1	3	1,3,9	Начало файла	«только для чтения», «скрытый»
2	3	4,7,10	Текущая позиция	«архивный», «скрытый»
3	3	2,3,10	Конец файла	«системный», «скрытый»
4	3	3,5,6	Начало файла	«только для чтения», «архивный»
5	3	1,8,10	Текущая позиция	«только для чтения», «системный»
6	4	2,4,6,8	Конец файла	«только для чтения», «скрытый»
7	4	1,5,7,8	Начало файла	«только для чтения», «зашифрованный»
8	4	2,3,7,8	Текущая позиция	«зашифрованный», «скрытый»
9	4	4,7,9,10	Конец файла	«зашифрованный», «архивный»
10	4	1,3,6,7	Начало файла	«только для чтения», «скрытый», «архивный»
11	3	6,8,10	Текущая позиция	«только для чтения», «скрытый», «системный»
12	3	7,9,12	Конец файла	«зашифрованный», «скрытый», «системный»
13	3	4,5,10	Начало файла	«зашифрованный», «скрытый», «архивный»
14	3	2,7,11	Текущая позиция	«только для чтения», «зашифрованный», «скрытый»
15	3	3,8,10	Конец файла	«только для чтения», «зашифрованный», «системный»
16	4	1,2,5,6	Начало файла	«только для чтения», «зашифрованный», «архивный»
17	4	1,5,6,8	Текущая позиция	«зашифрованный», «системный», «архивный»
18	4	2,3,5,8	Конец файла	«только для чтения», «системный», «неиндексированный»
19	4	2,6,9,10	Начало файла	«только для чтения», «скрытый», «неиндексированный»
20	4	5,6,8,9	Текущая позиция	«только для чтения», «архивный», «неиндексированный»
21	3	1,4,5	Конец файла	«только для чтения», «зашифрованный», «неиндексированный»
22	3	1,5,9	Начало файла	«скрытый», «системный», «неиндексированный»
23	3	4,6,8	Текущая позиция	«архивный», «системный», «неиндексированный»
24	3	6,8,10	Конец файла	«зашифрованный», «системный», «неиндексированный»
25	3	2,3,7	Начало файла	«только для чтения», «системный», «неиндексированный», «скрытый»

Примечание. Здесь и далее использовано сокращение: вар. – варианта.

Таблица 2

Задания к лабораторной работе № 2

№ вар.	Число процессов	Порядок создания процессов	P	Q
1	3	Вначале первый, после его завершения – параллельно второй и третий	1	1
2	3	Вначале параллельно первый и второй, после того, как хотя бы один из них завершится, – третий	2	2
3	3	Вначале параллельно первый и второй, после того, как хотя бы оба завершатся, – третий	1	3
4	4	Параллельно первые два, после завершения обоих – параллельно последние два	2	1
5	4	Параллельно первые два, после завершения хотя бы одного – параллельно последние два	1	2
6	4	Параллельно первые два, после завершения обоих – третий, после его завершения – четвертый	2	3
7	4	Параллельно первые три, после завершения трех – последний	1	1
8	4	Параллельно первые три, после завершения хотя бы одного из трех – последний	2	2
9	4	Первый, после его завершения – второй, после его завершения – последние два	1	3
10	5	Параллельно первые три, после завершения трех – параллельно последние два	2	1
11	5	Параллельно первые три, после завершения хотя бы одного из трех – параллельно последние два	1	2
12	5	Параллельно первые два, после завершения двух – параллельно последние три	2	3
13	5	Параллельно первые два, после завершения хотя бы одного из двух – параллельно последние три	1	1
14	5	Параллельно первые четыре, после завершения четырех – пятый	2	2
15	5	Параллельно первые четыре, после завершения хотя бы одного из четырех – пятый	1	3
16	6	Параллельно первые три, после завершения хотя бы одного из трех – параллельно последние три	2	1
17	6	Параллельно первые два, после завершения двух третий, после завершения третьего – параллельно последние три	1	2
18	6	Параллельно первые два, после завершения хотя бы одного из двух – параллельно последние четыре	2	3
19	6	Параллельно первые четыре, после завершения четырех – пятый, после завершения пятого – шестой	1	1
20	6	Параллельно первые четыре, после завершения хотя бы одного из четырех – последние два	2	2
21	6	Параллельно первые два, после завершения обоих – параллельно следующие два, после завершения хотя бы одного из этих двух – параллельно последние два	1	3
22	6	Параллельно первые два, после завершения хотя бы одного – параллельно последние четыре	2	1
23	6	Параллельно первые два, после завершения обоих – третий, после его завершения – четвертый, после его завершения – параллельно последние два	1	2
24	6	Параллельно первые три, после их завершения – параллельно последние три	2	3
25	6	Вначале первый, после его завершения – параллельно последние пять	1	1

Таблица 3

Задания к лабораторной работе № 4

№ вар.	Операция, выполняемая над элементами строки	Размер матрицы	№ вар.	Операция, выполняемая над элементами строки	Размер матрицы
1	Поиск заданного элемента	4	14	Сумма четных	5
2	Поиск максимума	5	15	Сумма нечетных	6
3	Поиск минимума	6	16	Произведение четных	7
4	Сумма	7	17	Произведение нечетных	4
5	Произведение	4	18	Квадрат первого элемента	5
6	Среднее	5	19	Квадрат последнего	6
7	Квадратный корень из суммы модулей	6	20	Сумма квадратов	7
8	Поиск первого четного	7	21	Сумма квадратов четных	4
9	Поиск первого нечетного	4	22	Сумма квадратов нечетных	5
10	Поиск первого элемента, кратного трем	5	23	Поиск элемента, кратного трем и пяти	6
11	Поиск первого элемента, кратного пяти	6	24	Поиск заданного элемента	7
12	Поиск первого элемента, кратного семи	7	25	Поиск максимума	4
13	Квадратный корень из суммы квадратов	4			

Таблица 4

Задания к лабораторной работе № 5

№ вар.	А, Б	M, N	№ вар.	А, Б	M, N
1	3, 1	4, 20	14	8,2	5,25
2	4, 2	5, 20	15	8,3	6,30
3	5, 2	6, 20	16	8,4	7,28
4	3, 2	7, 25	17	4, 3	8,30
5	4, 1	4, 21	18	4,2	6,15
6	5, 1	5, 22	19	4,1	6,34
7	6, 1	6, 26	20	7,5	7,18
8	6, 2	7, 24	21	8,4	4,24
9	6, 3	4, 12	22	5,4	5,56
10	6, 4	5, 25	23	7,2	7,23
11	5, 2	6, 34	24	7,3	6,45
12	5, 3	7, 35	25	7,1	6,78
13	7, 4	4, 25			

Таблица 5

Задания к лабораторной работе № 6

№ вар.	M, N,K	№ вар.	M, N, K	№ вар.	M, N, K
1	4, 20, 40	9	4, 12,60	17	8,16,30
2	5, 20, 50	10	5, 7,60	18	6,15,40
3	10, 20, 40	11	6, 14, 40	19	16,14,50
4	15, 25,60	12	7, 8,30	20	17,18,100
5	22, 21,50	13	4, 8,40	21	14,24,100
6	16, 22,40	14	5,25,50	22	15,15,40
7	6, 26,50	15	6,12,40	23	17,23,50
8	7, 24, 40	16	7,15,40	24	16,25,50

Таблица 6

Задания к лабораторной работе № 7

№ вар.	Операция	N
1	Выдача первой позиции в строке s1, в которой встретился любой символ из строки s2	10
2	Расширение записей вида a-z в строке s1 в соответствующий список abc…xyz в строке s2	12
3	Выдача позиции самого правого вхождения строки t в строку s	14
4	Замена каждой строки, содержащей более одного пробела, одним пробелом	15
5	«Переворачивание» строки s	9
6	Получение строки s, состоящей из символов, принадлежащих одновременно строкам s1 и s2	6
7	Замена всех троек пробелов в строке одним символом	8
8	Получение строки s, состоящей из символов строки s1, имеющих в ней нечетные позиции, и символов строки s2, имеющих в ней четные позиции	7
9	Поиск последнего вхождения подстроки s в строку s1	5
10	Замена в строке s подстроки, начинающейся с определенной позиции, на заданную подстроку	4
11	Замена всех пар одинаковых символов в строке одним символом	3
12	Подсчет числа вхождений подстроки s1 в строку s	13
13	Сравнение строк s1 и s2	6
14	Выдача первой подстроки, которая встречается в строках s1 и s2	5
15	Удаление хвостовых пробелов и табуляций у каждой вводимой строки	7
16	Проверка введенной строки на наличие в ней непарных круглых и квадратных скобок	17
17	Замена всех пар, троек и т.д. одинаковых символов строки s на один символ	12
18	Слияние первой и второй четвертей двух строк s1 и s2 соответственно в строку s	13
19	Удаление из строки s символов, встречающихся не менее одного раза в строке s1	11
20	Выдача последней позиции в строке s, в которой встретился заданный символ	8
21	Поочередное копирование символов из строк s1 и s2 в строку s	3
22	Слияние первой и второй половин строк s1 и s2 соответственно в строку s	8
23	Разбиение строки s на две подстроки s1 и s2 следующим образом: подстрока s1 образуется из всех нечетных символов s; подстрока s2 - из оставшихся символов s	6
24	Слияние второй и первой половин строк s1 и s2 соответственно в строку s	7

Журавлева Марина Гарриевна

Изучение Windows API

Методические указания к лабораторным работам по курсам
«Операционные системы»
и «Операционные системы и оболочки»

Редактор М. Ю. Болгова
Подписано в печать Формат 60x84 1/16. Бумага офсетная. Ризография.
Печ.л. 2,2. Тираж 85 экз. Заказ *N* .
Издательство Липецкого государственного технического университета.
Полиграфическое подразделение Издательства ЛГТУ.
398600 Липецк, ул. Московская, 30.