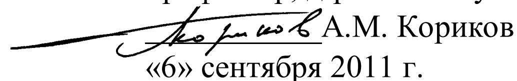
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
Томский государственный университет систем управления и радиоэлектроники

Кафедра автоматизированных систем управления

УТВЕРЖДАЮ
Зав. кафедрой АСУ
профессор, д-р. техн. наук
А.М. Кориков
«6» сентября 2011 г.

Ефремов В.А.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ И САМОСТОЯТЕЛЬНОЙ РАБОТЕ

По дисциплине «Теория вычислительных процессов»

для студентов специальности 230105 «Программное обеспечение вычислительной техники и автоматизированных систем»

Томск 2012

Методические указания по практическим занятиям и самостоятельной работе по дисциплине «Теория вычислительных процессов» составлены в соответствии с программой и включают в себя тематику практических занятий, их содержание, список тем предложенных для самостоятельного изучения, список вопросов для подготовки к экзамену и рекомендуемую литературу. Методические указания предназначены для студентов специальности 230105 «Программное обеспечение вычислительной техники и автоматизированных систем».

Практикум по предмету «Теория вычислительных процессов и структур» имеет целью закрепить теоретические знания принципов управления вычислительным процессом, методов синхронизации параллельных процессов, алгоритмов управления процессами в очередях к совместно используемым ресурсам, принципов синхронизации параллельных процессов, алгоритмов и способов предотвращения дедлоков.

Практические работы представляют собой разработку и реализацию алгоритмов моделирования управления процессами. Разработанная программа должна быть продемонстрирована преподавателю.

В результате выполнения практикума студенты должны уметь:

– моделировать работу дисциплин распределения ресурсов вычислительной системы;
– моделировать работу алгоритмов взаимодействия процессов и ресурсов в работе вычислительной системы;
– моделировать работу дисциплин синхронизации вычислительных процессов, методов предотвращения тупиковых ситуаций.

В ходе выполнения лабораторно-практических работ должен быть оформлен отчет, содержащий:

– цель работы;
– краткое описание метода управления процессами и алгоритма, моделирующего данный метод;
– набор тестов;
– выводы.

Практическая работа №1 «Моделирование организации и обслуживания очередей процессов к совместно используемым ресурсам» - 6 часов

Цель работы: научиться организовывать очереди процессов к совместно используемым ресурсам, моделировать дисциплины обслуживания созданных очередей.

Краткие теоретические сведения:

Идея мультипрограммирования непосредственно связана с наличием очередей процессов. Так процессор - основной ресурс мультипрограммной вычислительной системы по очереди предоставляется процессам. Подобные очереди неизбежно имеют место при обращении к внешним устройствам, наборам данных, модулям ОС. Использование многими процессами того или иного ресурса, который в каждый момент времени может обслуживать лишь один процесс, осуществляется с помощью дисциплин распределения ресурса. Их основой являются:

- дисциплины формирования очередей на ресурсы или совокупность правил, определяющих размещение процессов в очереди;
- дисциплины обслуживания очереди или совокупность правил извлечения одного из процессов очереди с последующим предоставлением выбранному процессу ресурса для использования.

Основным конструктивным, согласующим элементом при реализации той или иной дисциплины диспетчеризации является очередь, в которую по определенным правилам заносятся и извлекаются запросы.

Определяющее влияние на сущность дисциплины формирования очередей оказывают:

- информация о классах и приоритетах заданий и шагов заданий, информация о необходимости обращения к тем или иным устройствам, массивам данных, зафиксированных в операторах языка управления заданиями;
- соглашения о приоритете уровней запросов прерывания и прерывающих программ, принимаемых при проектировании, разработке ВС;
- используемая дисциплина обслуживания очередей, которая зачастую определяет и дисциплину формирования очереди.

Дисциплины формирования очередей разделяются на *два класса*:

- статический, где приоритеты назначаются до выполнения пакета заданий;
- динамический, при котором приоритеты определяются в процессе выполнения пакета.

Оба класса широко используются в практике организации вычислительного процесса. В любых ВС широко используется ряд дисциплин обслуживания очередей, ставших классическими, их называют базовыми.

Классическими дисциплинами формирования и обслуживания запросов на предоставление ресурса процессам являются:

Дисциплина обслуживания в *порядке поступления*. (FIFO)

Дисциплина обслуживания в порядке, *обратном поступлению*. (LIFO)

Общим для этих дисциплин является простота их реализации и определенная «справедливость» в обслуживании всего потока запросов. Среднее время ожидания запросов в очереди будет примерно одинаковым, независимо от характеристик процессов.

Круговой циклический алгоритм

В основе лежит дисциплина FIFO. Но время обслуживания каждого процесса ограничено и определяется так называемым квантом времени. Если запрос на использование ресурса из начала очереди обслуживается до конца за время выделенного кванта, то он покидает очередь, в противном случае - становится в конец очереди. Дисциплина реализует широко принятый режим разделения времени. Хотя в данной дисциплине нет явных приоритетов, автоматически происходит дискриминация «длинных» запросов.

Многоочередные дисциплины

Здесь приоритет определяется очередью (ее номером). Первый запрос из очереди i поступает на обслуживания лишь тогда, когда все очереди от 1 до i-1 пустые. На обслуживание выделяется квант времени. Если за это время обслуживание запроса завершается полностью, то он покидает систему. В

противном случае недообслуженный запрос поступает в конец очереди с номером $i+1$.

После обслуживания из очереди i система выбирает для обслуживания запрос из непустой очереди с самым младшим номером.

Обслуживание с абсолютным приоритетом.

В многоочередной дисциплине, где вновь поступающий запрос имеет определенный приоритет, используется обслуживание с абсолютным приоритетом. Первыми обслуживаются запросы с наивысшим приоритетом (из очереди с меньшим номером). При поступлении запроса с более высоким приоритетом обслуживание текущего запроса прерывается и обслуживается вновь поступивший. После этого дообслуживается прерванный запрос.

Обслуживание с относительным приоритетом.

При данной дисциплине заявка, входящая в систему, не вызывает прерывания обслуживаемой заявки, даже если последняя и менее приоритетная. В данном случае только после окончания обслуживания менее приоритетной начинается обслуживание более приоритетной.

Все вышеизложенное касается дисциплины распределения ресурсов без учета взаимосвязи процессов. Процессы, как правило, используют различные ресурсы и этот факт оказывает влияние на рассмотрение дисциплины распределения. Но практике д. б. построена стратегия распределения, удовлетворительная не только в отношении некоторого конкретного ресурса, но и согласованная со стратегиями распределения других ресурсов.

Реализация дисциплин распределения ресурсов на практике также усложняется из-за необходимости анализа возможности возникновения тупиковых ситуаций, проверки анализа полномочий на использование каждым процессом распределяемого ресурса.

Наряду с рассмотренными дисциплинами распределения ресурсов существуют и другие, содержание которых определяется спецификой распределяемого ресурса. Много таких оригинальных дисциплин имеет место при статическом и динамическом распределении между процессами ОП.

Порядок выполнения работы:

Моделирование формирования и обслуживания очередей процессов к совместно используемым ресурсам можно производить на основе абстрактных процессов, которым назначают имена, приоритеты, время обслуживания запросов (в условных единицах). Программа модели формирования и обслуживания очереди должна наглядно представлять порядок прохождения процессов в очереди. При организации многоочередных дисциплин с относительными приоритетами необходимо показать изменения приоритетов во времени и место процесса в очереди.

Разработать программу, которая будет реализовывать:

- круговой циклический алгоритм обслуживания запросов процессов на использование некого ресурса;
- многоочередной алгоритм обслуживания процессов с абсолютным приоритетом;
- многоочередной алгоритм обслуживания процессов с относительным приоритетом.

Программное приложение должно иметь графический интерфейс с возможностью интерактивного выбора алгоритма обработки очереди. Процесс обслуживания очередей должен быть наглядно отображен в программе. В любой момент времени обслуживание может быть приостановлено и возобновлено, а так же добавлен новый процесс.

Входные данные должны быть загружены из файла input.xml.

time_measurement – единицы измерения времени.

mm – минуты

ss – секунды

ms – миллисекунды

resource_time_limit – максимальный временной интервал, который может быть выделен отдельно взятому процессу ресурсом.

Отчет должен содержать набор тестов – таблицу следующего вида

Название	Входные данные	Выходные данные	Результат
Название теста	Описание входных данных	Описание выходных данных	Успешно/Ошибка

Практическая работа №2 «Управление процессами Windows» - 6 часов.

Цель работы: научиться управлять процессами Windows

Краткая теоретические сведения:

Объекты ядра

Объекты ядра используются операционной системой и приложениями для работы с управления множеством ресурсов таких как: процессы, потоки, файлы и т.д. Создание, уничтожение и прочие операции над объектами ядра происходят с помощью соответствующих вызовов Windows-функций. Объект ядра – это блок памяти, выделенный ядром и доступный только ему. Этот блок представляет структуру данных, в которой содержится информация об объекте. Некоторые элементы структуры содержатся во всех объектах, такие как дескриптор защиты, счетчик числа пользователей и т.д., а некоторые сугубо специфичными.

Когда происходит вызов функции создающей объект, она возвращает описатель(HANDLE), идентифицирующий созданный объект. Этот описатель стоит рассматривать как непрозрачное значение, действительное в рамках текущего процесса и всех его потоках. В последствии использование ресурса будет происходить посредством вызова соответствующей функции Windows с передачей ей в качестве параметра этого значения.

Создание объекта ядра происходит в момент вызова функций Windows, предназначенных для этого. Для создания разных объектов, предусмотрены разные вызовы. Так для создания (или открытия) файла необходимо вызвать функцию CreateFile, для создания семафора – CreateSemaphore и т.д. В качестве возвращаемого значения эти функции передают описатели объекта.

Закрытие объекта ядра происходит функцией BOOL CloseHandle(HANDLE hObj), независимо от типа объекта.

Иногда необходимо использовать один объект во многих процессах. Но так как таблица описателей является индивидуальной для каждого процесса, то просто передав описатель объекта другому процессу мы не сделаем его доступным для этого процесса. С нашим описателем в другом процессе может

быть сопоставлен совершенно другой объект, созданный в том процессе. В Windows для некоторых объектов предусмлтрена возможность сопоставление с ними имени. Так другой процесс может получить к уже созданному в другом процессе объекту по его имени.

Процессы

Процесс – это экземпляр выполняемой программы и состоит он из следующих компонентов:

– объект ядра, через который операционная система управляет процессом;
– адресное пространство, в котором содержится код exe-модуля.

Для того чтобы процесс начал выполнять какие-то операции, необходимо создать в нем поток. Именно потоки отвечают за исполнение кода. В каждом процессе есть минимум один поток. При создании процесса, операционная система автоматически создает в нем один поток, так называемый первичный поток. Далее этот поток может порождать новые.

Для создания процесса, необходимо вызвать функцию CreateProcess. Прототип приведен ниже:

```
BOOL CreateProcess(
  LPCTSTR lpApplicationName,                 // name of executable
module
  LPTSTR lpCommandLine,                      // command line string
  LPSECURITY_ATTRIBUTES lpProcessAttributes, // SD
  LPSECURITY_ATTRIBUTES lpThreadAttributes,  // SD
  BOOL bInheritHandles,                      // handle inheritance
option
  DWORD dwCreationFlags,                     // creation flags
  LPVOID lpEnvironment,                      // new environment block
  LPCTSTR lpCurrentDirectory,                // current directory
name
  LPSTARTUPINFO lpStartupInfo,        // startup information
 LPPROCESS_INFORMATION lpProcessInformation // process information
);
```

Первый параметр должен содержать адрес строки с именем исполняемого файла, который надо запустить. При передаче сюда имени файла, необходимо передавать так же его расширение. Если файл не находится в том же каталоге, что и исполняющийся в данный момент exe-модуль, то вызов окончится неудачей. В большинстве случаев в отот параметр следует передавать NULL, а имя файла и его параметры передавать в параметр lpCommandLine.

Параметры lpProcessAttributes и lpThreadAttributes позволяют родительскому процессу установить атрибуты защиты новых объектов “процесс” и ”поток”. Для использования атрибутов защиты по умолчанию необходимо предавать в этих параметрах значение NULL.

Параметр bInheritHandles показывает, будет ли порождаемый процесс иметь доступ к объектам ядра, созданным в родительском процесс. Если этот параметр равен TRUE, то при создании нового процесса, операционная система создаст в новом процессе таблицу дескрипторов, в которой будут содержаться дескрипторы объектов родительского процесса.

Параметр dwCreationFlags определяет, как будет создаваться новый процесс. Список возможных значений, которые могут комбинироваться методом OR приведен в таблице ниже.

Значение	**Описание**
CREATE_NEW_CONSOLE	Новый процесс использует новую консоль вместо консоли родительского процесса, этот флаг не может быть использован флагом DETACHED_PROCESS.
CREATE_NEW_PROCESS_GROUP	Новый процесс является коренным для новой группы процессов.
CREATE_SUSPENDED	Первичный поток процесса не начинает исполнение кода, пока не будет вызвана функция ResumeThread.
CREATE_UNICODE_ENVIRONMENT	Если этот флаг установлен, то формат окружения, которое передается в переменной lpEnvironment, является Unicode-строкой, иначе – ANSI.
DEBUG_PROCESS	Если этот флаг установлен, то вызывающий процесс рассматривается как отладчик, а новый процесс как отлаживаемый. Операционная система уведомляет родительский процесс обо всех событиях, происходящих в отлаживаемом процессе.
DETACHED_PROCESS	Этот флаг используется для консольных приложений. Новый процесс не может использовать консоль родительского процесса. Новый процесс может вызвать функцию AllocConsole для создания новой консоли.

В таблице приведены не все возможные значения флагов, так как многие из них были введены в Windows 2000 или XP. Для задания флагов по умолчанию нужно передать в этот параметр значение NULL.

Переменная lpEnvironment указывает на строки с переменными окружения. Обычно имеет значение NULL.

Параметр lpCurrentDirectory позволяет установить рабочий каталог для порожденного процесса.

Параметр lpStartupInfo указывает на структуру, которая говорит как должно появляться главное окно дочернего процесса. Она имеет следующий вид:

```
typedef struct _STARTUPINFO {
    DWORD   cb;
    LPTSTR  lpReserved;
```

```
    LPTSTR   lpDesktop;
    LPTSTR   lpTitle;
    DWORD    dwX;
    DWORD    dwY;
    DWORD    dwXSize;
    DWORD    dwYSize;
    DWORD    dwXCountChars;
    DWORD    dwYCountChars;
    DWORD    dwFillAttribute;
    DWORD    dwFlags;
    WORD     wShowWindow;
    WORD     cbReserved2;
    LPBYTE   lpReserved2;
    HANDLE   hStdInput;
   HANDLE   hStdOutput;
    HANDLE   hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

В большинстве случаев используются параметры по умолчанию, но для создания процесса необходимо создать эту структуру и установить все поля в 0, а полю cb передать размер структуры.

В параметр lpProcessInformation нужно передать указатель на структуру типа PROCESS_INFORMATION, в которую после вызова будет записана информация о созданном поцессе и его первичном потоке. Эта структура имеет вид:

```
typedef struct _PROCESS_INFORMATION {
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
} PROCESS_INFORMATION;
```

Где первые два параметра поля – это описатели созданного процесса и его первичного потока, а вторые, соответственно их идентификаторы, или так называемые псевдоописатели.

Таким образом, вызов для создания нового процесса, в котором выполняется код приложения калькулятора выглядит следующим образом:

```
STARTUPINFO si={sizeof(si)};
PROCESS_INFORMATION pi;
TCHAR szPath[]=TEXT("calc");
CreateProcess(NULL,szPath,NULL,NULL,FALSE,0,NULL,NULL,&si,&pi);
```

После того как описатели нового процесса и его первичного потока больше не нужны, необходимо закрыть их.

```
CloseHandle(pi.hThread);
CloseHandle(pi.hProcess);
```

Для завершения процесса используется вызов TerminateProcess. Прототип приведен ниже:

```
BOOL TerminateProcess(
  HANDLE hProcess, // handle to the process
  UINT uExitCode   // exit code for the process
);
```

Параметр hProcess содержит описатель процесса, подлежащего убиению, параметр uExitCode – код ошибки с которой завершается этот процесс.

Для получения описателя процесса по его идентификатору используется вызов OpenProcess. Он имеет следующий прототип:

```
HANDLE OpenProcess(
  DWORD dwDesiredAccess,  // access flag
  BOOL bInheritHandle,    // handle inheritance option
 DWORD dwProcessId       // process identifier
);
```

Параметр dwDesiredAccess показывает какие операции, мы собираемся производить с процессом и какие права доступа нам для этого потребуются. Возможные значения приведены в таблице ниже, они могут комбинироваться операцией OR.

Значение	**Описание**
PROCESS_ALL_ACCESS	Все возможные права доступа к процессу.
PROCESS_CREATE_THREAD	Позволяет создавать потоки внутри откоываемого процесса с помощью функции CreateRemoteThread.
PROCESS_QUERY_INFORMATION	Позволяет читать информацию о процессе с помощью функций GetExitCodeProcess и GetPriorityClass.
PROCESS_SET_INFORMATION	Позволяет изменять приоритет процесса с помощью функции SetPriorityClass.
PROCESS_TERMINATE	Позволяет завершить процесс вызовом TerminateProcess.
PROCESS_VM_OPERATION	Позволяет модифицировать виртуальную память процесса.
PROCESS_VM_READ	Позволяет читать память процесса.

Приоритеты процессов

Для получения приоритета процесса используется функция GetPriorityBoost.

```
DWORD GetPriorityClass(
  HANDLE hProcess   // handle to process
);
```

Возвращаемые значения перечислены в таблице:

Значение	**Описание**
HIGH_PRIORITY_CLASS	Высокий приоритет
IDLE_PRIORITY_CLASS	Самый низкий приоритет
NORMAL_PRIORITY_CLASS	Нормальный
REALTIME_PRIORITY_CLASS	Приоритет реального времени

Для изменения приоритета необходимо вызвать функцию:

```
BOOL SetPriorityClass(
  HANDLE hProcess,        // handle to process
 DWORD dwPriorityClass   // priority class
);
```

Перечисление выполняемых процессов

Для получения информации о работающих процессах в Windows 9x используется набор функций под названием ToolHelp API. В Windows NT программный интерфейс для решения таких задач используется Process Status API. Windows NT 4 не имеет поддержки интерфейса ToolHelp, а W2K поддерживает как первый так и второй интерфейсы. Так как лабораторные работы выполняются под WinNT4, то рассмотрим только Process Status.

Для работы с этим набором функций необходимо подключить заголовочный файл psapi.h и включить в проект библиотеку psapi.lib.

Функция EnumProcesses возвращает список идентификаторов процессов выполняющихся в системе.

```
BOOL EnumProcesses(
  DWORD *lpidProcess,   // array of process identifiers
  DWORD cb,             // size of array
  DWORD *cbNeeded    // number of bytes returned
);
```

В параметр lpidProcess необходимо передать указатель на массив типа DWORD, массив должен иметь достаточно места для размещения в нем списка идентификаторов, заранее предугадать требуемый размер невозможно, поэтому создадим массив для размещения 1024 элементов.

В параметре cb передается количество байт в массиве lpidProcess.

В результате выполнения функции cbNeeded содержит количество байт, заполненных в lpidProcess. Для того чтобы узнать количество процессов необходимо поделить это значение на размер DWORD.

Для получения списка модулей процесса используется функция EnumProcessModules.

```
BOOL EnumProcessModules(
  HANDLE hProcess,       // handle to process
  HMODULE *lphModule,    // array of module handles
  DWORD cb,              // size of array
  LPDWORD lpcbNeeded  // number of bytes required
);
```

hProcess – описатель процесса, список модулей в котором получаем.

lphModule – указатель на массив, куда будут помещены описатели загруженных модулей. Так как первый загруженный модуль в пространство процесса был его EXE-модуль, то нам потребуется массив размером в один элемент.

cb – размер массива lphModule в байтах, в нашем случае он будет равен sizeof(HMODULE).

lpcbNeeded – размер заполненного пространства в массиве lphModule в байтах.

По описателю модуля можно узнать его имя вызвав функцию GetModuleBaseName.

```
DWORD GetModuleBaseName(
  HANDLE hProcess,     // handle to process
  HMODULE hModule,     // handle to module
  LPTSTR lpBaseName,   // base name buffer
  DWORD nSize      // maximum characters to retrieve
);
```

где hProcess – описатель процесса.
hModule – описатель модуля в процессе.
lpBaseName – указатель на строку в которую помещается имя модуля.

Задание

Написать программу, которая позволила бы просматривать список процессов работающих в системе, их приоритеты и идентификаторы. В программе предусмотреть возможность создания новых процессов и уничтожение существующих, а так же изменение приоритетов работающих процессов.

Лабораторная работа №3 «Организация пула потоков в Windows»
Краткая теоретические сведения:
Потоки

В каждом процессе есть по крайней мере один поток (см. Практическую работу №2). Этот поток создается при создании нового процесса и называется первичным потоком процесса. Первичный поток может создавать дополнительные потоки для параллельной работы нескольких задач. Во время создания потока ему передается функция, код которой он и будет исполнять. Для первичного потока этой функцией является main, wmain, wWinMain или WinMain. Для вторичных потоков функция имеет следующий вид:
DWORD WINAPI ThreadFunc(PVOID pvParam).

Эта функция может выполнять любые операции. По завершении, она должна вернуть код завершения. При этом счетчик пользователей объекта ядра потока уменьшится на 1.

Для создания потока, используется функция CreateThread.

```
HANDLE CreateThread(
 LPSECURITY_ATTRIBUTES lpThreadAttributes, // SD
  SIZE_T dwStackSize,                       // initial stack size
  LPTHREAD_START_ROUTINE lpStartAddress,    // thread function
  LPVOID lpParameter,                       // thread argument
  DWORD dwCreationFlags,                    // creation option
  LPDWORD lpThreadId                        // thread identifier
);
```

Где:
lpThreadAttributes – атрибуты безопасности, для значения по умолчанию, необходимо передать NULL.
dwStackSize – размер стека потока, NULL.
lpStartAddress – адрес функции потока, прототип которой показан выше.
lpParameter – параметр функции потока, это значение будет передаваться функции потока.
dwCreationFlags – флаг создания потока, если это значение равно CREATE_SUSPENDED, то поток создается в приостановленом состоянии, для начала работы потоканеобходимо будет вызвать функцию ResumeThread. В случае если этот параметр равен 0, то поток начинает выполняться немедленно.
lpThreadId – указатель на переменную, куда будет записан идентификатор потока в результате вызова.
Завершение потока происходит в результате вызова функции TerminateThread.

```
BOOL TerminateThread(
  HANDLE hThread,     // handle to thread
 DWORD dwExitCode    // exit code
);
```

hThread – описатель потока, который необходимо завершить.
dwExitCode – код завершения потока.

Синхронизация потоков в пользовательском режиме

В этом разделе рассмотри семейство функций Interlocked. Это самый незначительный и неинтересный раздел из темы синхронизации, но необходимый для выполнения лабораторной работы. Подробнее о примитивах для синхронизации смотрите в MSDN или в файлах chm на ftp-сервере.

При параллельной работе нескольких потоков неизбежно возникает ситуация в которой несколько потоков нуждаются в одном и том же ресурсе. Синхронизация помогает избежать повреждения данных и нарушение их целостности. Кроме того, с помощью объектов синхронизации, потоки могут уведомлять друг друга об определенных событиях.

Семейство функций interlocked существуют для организации атомарного доступа к переменным. Они помогают избежать ситуации, когда несколько потоков одновременно пытаются писать в переменную или один производит запись, а второй – не дожидаясь, окончания изменения значения переменной производит попытку чтения этого значения.

Атомарный доступ – монопольный захват ресурса обращающимся к нему потоком. То есть до завершения операции инициированной одним потоком, остальные потоки не получат доступа к этому ресурсу.

```
LONG InterlockedDecrement(
  LPLONG volatile lpAddend   // variable address
);
```

Эта функция производит вычитание единицы из переменной, адрес которой передается в lpAddend.

Для увеличения переменной на единицу используют функцию:

```
LONG InterlockedIncrement(
  LPLONG volatile lpAddend   // variable to increment
);
```

Функция InterlockedExchange атомарно меняет значение переменной на указанное.

```
LONG InterlockedExchange(
  LPLONG volatile Target, // value to exchange
  LONG Value              // new value
);
```

Где: *Target* – указатель на переменную.
Value – новое значение переменной.
Все эти функции возвращают предыдущее значение переменной.

Пулы потоков

Пулы потоков, как правило, используются в серверных приложениях. Серверные приложения имеют в общем следующую структуру. Есть поток – слушатель, который принимает запросы и производит их диспетчеризацию. При получении запроса, поток-слушатель создает новый поток и передает обработку запроса ему. В этом случае много времени тратится на создание и уничтожение потоков. Выход из этой ситуации находится в пуле потоков. В пуле потоков на этапе создания, создаются несколько потоков, которые будут заниматься обработкой запросов. При появлении запроса, он передается свободному потоку из пула, если все потоки заняты обработкой каких-либо запросов, то он ставится в очередь и начинает обрабатываться после освобождения одного из потоков. Таким образом мы избегаем операций по многократному созданию и уничтожению потоков. В Windows 2000 предусмотрен объект ядра, реализующий функции пула потоков, но в NT4, этот примитив не доступен. По этому мы с вами рассмотрим возможность организации пула с помощью объекта ядра – порта завершения ввода/вывода.

Порт завершения ввода/вывода

Создание порта завершения происходит вызовом функции CreateIoCompletionPort. Ее прототип приведем ниже.

```
HANDLE CreateIoCompletionPort (
  HANDLE FileHandle,                  // handle to file
 HANDLE ExistingCompletionPort,   // handle to I/O completion port
  ULONG_PTR CompletionKey,          // completion key
  DWORD NumberOfConcurrentThreads // number of threads to execute
concurrently
);
```

Создание происходит в два этапа. На первом этапе вы вызываете эту функцию со следующими параметрами.

HANDLE hCompPort=CreateIoCompletionPort (NULL,NULL,0,Number).

Где Number – это количество потоков в пуле.

Затем вы создаете файл. Это необходимо для связи порта с устройством. Файл создается потому, что вся работа с устройствами в виндах происходит через файл. Файл создается с помощью вызова CreateFile (см. MSDN или msdn.microsoft.com).

При следующем вызове функции CreateIoCompletionPort вы передаете в параметре FileHandle описатель созданного файла, в ExistingCompletionPort – hCompPort, полученный в результате предыдущего вызова этой функции. Параметр – это занчение которое будет передаваться потокам пула при регистрации им завершения операции ввода/вывода. В параметре NumberOfConcurrentThreads передаете то же самое значение, что и при предыдущем вызове, то есть максимальное число потоков в пуле.

После создания пула, вам необходимо создать потоки, которые будут находиться в пуле и ожидать окончания операции ввода/вывода. Это делается вызовом CreateThread, который был описан выше. При создании потоков, вам необходимо запоминать их описатели, потому как после удаления пула потоков,

вам необходимо будет принудительно завершить все созданные потоки вызовом TerminateThread.

Функции всех созданных вами потоков, должны быть одними и теми же, то есть в параметре – адрес функции потока в вызове CreateThread вы во всех потоках указываете одну и ту же функцию.

Каждый из созданных потоков должен впадать в бесконечный цикл ожидания и обработки запросов. Для получения нотификации о завершении операции, связанной с нашим устройством, поток должен вызвать функцию

```
BOOL GetQueuedCompletionStatus(
  HANDLE CompletionPort,       // handle to completion port
  LPDWORD lpNumberOfBytes,     // bytes transferred
  PULONG_PTR lpCompletionKey,  // file completion key
  LPOVERLAPPED *lpOverlapped,  // buffer
  DWORD dwMilliseconds     // optional timeout value
);
```

Где:

CompletionPort – описатель созданного нами порта завершения ввода/вывода.

lpNumbersOfBytes – указатель на переменную, куда будет записано количество байтов переданных в результате операции ввода/вывода.

lpCompletionKey – указатель на параметр, передаваемый нам потоком, который инициирует операцию.

lpOverlapped – указатель на структуру OVERLAPPED, эта структура используется при асинхронном вводе/выводе, вы должны создать экземпляр этой структуры и передаьб в параметр указатнль на нее, в результате вызова она будет заполнена информацией о которой можете узнать в MSDN.

dwMilliseconds – время в миллисекундах, которое наш потто может находиться в состоянии ожидания завершения операции, для нашего случая используется константа INFINITE, которая заставляет поток ожидать бесконечно.

В результате вызова этой функции поток впадет в спячку до тех пор, пока другой поток не инициирует завершение операции ввода/вывода и порт завершения не передаст управление этому потоку.

Для эмуляции завершения операции ввода/вывода, то есть, именно той операции, которая повлечет за собой пробуждение одного из потоков в пуле, необходимо вызвать функцию PostQueuedCompletionStatus.

```
BOOL PostQueuedCompletionStatus(
  HANDLE CompletionPort,              // handle to an I/O completion
port
  DWORD dwNumberOfBytesTransferred,   // bytes transferred
  ULONG_PTR dwCompletionKey,          // completion key
  LPOVERLAPPED lpOverlapped           // overlapped buffer
);
```

Где:

CompletionPort – описатель созданного нами порта завершения ввода/вывода.

DwNumberOfBytesTransferred – количество байт, которые мы якобы транслируем, должно быть больше 0.

dwCompletionKey – указатель на переменную, или на что захотите, этот указатель будет передан в lpCompletionKey потоку из пула по завершении ожидания в функции GetQueuedCompletionStatus/

lpOverlapped – указатель на структуру OVERLAPPED, сдесь, если она вам не нужна можете спокойно передавать NULL.
Таким образом мы передадим потоку в пуле задание на обработку.

Задание

Написать программу, которая создавала бы пул потоков с заданным количеством потоков и позволяла бы передавать им текстовое сообщение, которое параллельно выводилось в окнах сообщений (функция MessageBox) в разных потоках. Кроме того отдельный поток каждые 1000 миллисекунд должен выводить количество занятых потоков в пуле и общее количество потоков в пуле.

Проработка лекционного материала (27 часов)

Самостоятельная работа с материалами лекций и литературой для более глубокого и детального изучения разделов дисциплины, подготовка к их обсуждению на практических занятиях.

Подготовка к практическим занятиям (13 часа)

Самостоятельная работа с материалами лекций и литературой по темам практических занятий, выполнение практических заданий.

Список вопросов к экзамену по курсу «Теория вычислительных процессов»

1. Программа как формализованное описание процесса обработки данных
2. Правильная программа и надежная программа
3. . Функции и графы
4. Вычислимость и разрешимость
5. Программы и схемы программ
6. Базис класса стандартных схем программ
7. Линейная форма стандартной схемы
8. Интерпретация стандартных схем программ
9. Эквивалентность, тотальность, пустота, свобода
10. Свободные интерпретации
11. Согласованные свободные интерпретации
12. Логико-термальная эквивалентность
13. . Одноленточные автоматы
14. Многоленточные автоматы
15. Двухголовочные автоматы
16. . Двоичный двухголовочный автомат
17. Построение схемы, моделирующей автомат
18. Рекурсивное программирование
19. Определение рекурсивной схемы
20. О сравнении класс сов схем
21. Схемы с процедурами
22. Классы обогащенных схем
23. Трансляция обогащенных схем
24. Структурированные схемы
25. Операционная семантика
26. Аксиоматическая семантика

27. Денотационная семантика
28. Декларативная семантика
29. . Взаимодействующие последовательные процессы. Определения
30. ВПП, Префиксы, Рекурсия, Выбор
31. ВПП, Взаимная рекурсия
32. ВПП, Реализация процессов, Протоколы, Операции над протоколами
33. Задача об обедающих философах
34. Языки формальных спецификаций
35. Спецификации задачи взаимодействия процессов
36. Параллельные процессы. Определения и законы.
37. Взаимодействие – обмен сообщениями.
38. Разделяемые ресурсы.
39. Программирование параллельных процессов. Основные понятия.
40. Модели параллельных процессов.
41. Сформулируйте определение сети Петри.
42. Дайте теоретико-множественное определение сетей Петри.
43. Что такое графы сетей Петри?
44. Правила Маркировки Сетей Петри.
45. Правила выполнения сетей Петри.
46. Технолгия моделирование систем на основе сетей Петри.
47. События и условия в сетях Петри.
48. Модель одновременности и конфликта в сетях Петри.
49. Моделирование параллельных систем взаимодействующих процессов основе сетей Петри.
50. Моделирование последовательных процессов основе сетей Петри.
51. Моделирование взаимодействия процессов основе сетей Петри.
52. Задача о взаимном исключении.
53. Задача о производителе/потребителе.
54. Задача об обедающих философах.
55. Задачи анализа сетей Петри
56. Методы анализа сетей Петри.
57. Анализ свойств сетей Петри на основе дерева достижимости.
58. Матричные уравнения сети Петри.

Основная литература:

1. Калайда В.Т. Теория языков программирования и методов трансляции. Учебное пособие. – Томск: Изд. – во, ТУСУР, 2007 – 244 с. (45 экз.).

Дополнительная литература:

1. Рейуорд-Смит, В. Дж. Теория формальных языков. Вводный курс : Пер. с англ. / В. Дж. Рейуорд-Смит ; пер. Б. А. Кузьмин, ред. пер. Б. А. Шестаков. - М.: Радио и связь, 1988. - 124 с. (10 экз.)

2. Льюис Ф., Розешкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. — М.: Мир, 1979. – 656 с. (2 экз.)