**Задания на лабораторные работы по дисциплине «Системы реального времени»**

**Лабораторная работа №1. Задача «Производитель-Потребитель» (Producer-Consumer)**

Разработать приложение, состоящее из двух функциональных частей. Первая время от времени отправляет сообщения (например, это может происходить при нажатии некоторой кнопки, или клавиши на клавиатуре) второй, вторая каким-то образом на них реагирует. Размер буфера передачи между этими частями – 1 сообщение, при этом первая задача (Производитель) никогда не ждет очистки буфера. Т.е. если вторая часть (Потребитель) не успевает получить помещенное в буфер сообщение, а уже возникает новое сообщение, то это новое сообщение заменяет старое, а старое - теряется. Здесь Потребитель имитирует аппаратное обеспечение СРВ. Время обработки сообщения приемником должно быть 1-2 секунды (можно воспользоваться функцией Sleep). Необходимо добиться того, чтобы при возникновении 2-3 сообщений подряд (до окончания обработки первого из них) ни одно из них не терялось, т.е. все сообщения обрабатывались Потребителем.

Ожидается использование таких синхронизационных примитивов, как событие, критическая секция или мьютекс. Возможно использование семафоров или семейства Interlocked-функций.

Предлагается такая схема работы. Кроме главного, создаются еще два потока. Один из них с нужной частотой пишет в «буфер» (тот самый, который одно сообщение) какие-то символы, а другой, как только в буфере появляется символ, читает его оттуда и помещает в какой-то свой буфер. Главный же поток ждет, пока в этом втором буфере появится очередной символ, и затем его «обрабатывает» (например, выводит на экран). Второй буфер должен быть достаточно большим, чтобы вместить 2-3 сообщения (см. условие задачи), где они будут ожидать, пока главный поток их не обработает.

Для ожидания какого-то события (например, прихода символа) удобно использовать объект Event (событие).

**Лабораторная работа №2. Пул потоков (Thread Pool)**

В случаях, когда целевая система является многопроцессорной (или даже с одним многоядерным процессором, или с технологией HyperThreading) часто есть возможность ускорить обработку поступающих событий, выполняя ее параллельно. Для обработки каждого из событий используется отдельный поток; максимальное количество потоков, разумеется, ограничивается каким-то числом, например, 2, 4 или даже 10 – зависит от мощности системы. Т.к. создание потока – дорогостоящая операция, то обычно используют так называемый пул потоков: единожды созданные N потоков в своем нормальном состоянии простаивают, ожидая запроса на выполнение задачи, для решения которой они написаны. Когда такой запрос приходит, поток переходит в активное состояние, в котором он начинает потреблять процессорное время и решает задачу. По окончании обработки поток возвращается в состояние простоя. Если в момент возникновения очередной задачи свободных потоков нет, приходится, как правило, ждать, пока один из них освободится.

Реализовать пул потоков для усовершенствования решения лабораторной работы № 1.

Предлагается такая схема работы. Кроме главного, создаются еще N потоков (пул потоков), а также заводится семафор с начальным значением 0. При поступлении очередного символа во внутренний буфер (на 2-3 символа, созданный в первой задаче) тот поток, который помещает символ в буфер, увеличивает значение семафора на 1. Фактически, будем стремиться к тому, чтобы значение семафора равнялось количеству помещенных в буфер, но еще не взятых в обработку символов.

Каждый из N потоков пула имеет такую логику работы: ждем, пока не появится очередное задание, затем забираем это задание (чтобы другие его уже не получили), обрабатываем его и снова ждем и т.д. В переводе на нашу схему с семафором новое задание есть тогда, когда в буфере есть не переданные ни одному потоку символы, т.е. когда счетчик семафора больше 1. Тогда шаг «дождаться нового задания» будет выражаться одним вызовом WaitForSingleObject, примененным к дескриптору семафора. При этом произойдет следующее:

1. Поток будет ждать по крайней мере до тех пор, пока счетчик ссылок семафора не станет больше 0.
2. Когда это случится (счетчик станет больше 0), система выберет из ждущих потоков один, уменьшит счетчик семафора и возобновит выполнение этого потока. Если после этого счетчик все еще будет больше 0, то система может разморозить еще один ждущий поток и т.д.

Но при этом, если разморозились два или более потока, оба потока попытаются прочитать первый символ из буфера. А нужно, чтобы один из них прочитал первый символ, другой прочитал следующий, и так далее. Значит, уже после WaitForSingleObject само чтение символа из буфера надо поставить на мьютекс (собственно, мьютекс в этом месте уже должен быть в коде, решающем лабораторную работу 1, чтобы не возникло состояние гонки (Race Conditions) с тем потоком, который пишет символы в буфер). Если операция «забрать следующий символ из буфера» целиком будет заключена в этот мьютекс (т.е. перед операцией мы захватим мьютекс, а после – отпустим), то размороженные потоки смогут выполнить эту операцию только последовательно, по очереди – но не одновременно. Тогда один размороженный поток заберет очередной символ из буфера, в результате «очередным символом» станет уже следующий за ним; его прочитает другой поток и т.д.

**Лабораторная работа №3. Применение вторичных потоков для обслуживания ввода-вывода**

Поговорим о применении потоков для работы с каналами ввода-вывода (традиционно, их в русском компьютерном языке также называют потоками, но это уже английское stream, а не thread; говоря здесь о потоках, подразумеваем именно потоки ввода-вывода) – на примере довольно часто возникающей задачи. Пусть есть некое консольное приложение (неважно, написанное для Window или для MS-DOS), которое выводит данные в консоль «строчка за строчкой». Более продвинутые приложения, такие, как Far Manager, осуществляют вывод в консоль по другому принципу, и их мы рассматривать не будем.

Разработать программу, которая для решения одной из подзадач вызывает это консольное приложение. Необходимо считывать информацию, которую вызываемое (дочернее) приложение выводит в консоль, и показывать ее в окне нашего приложения. И наоборот, когда дочернему приложению нужен ввод с клавиатуры, мы должны вводить текстовую строку в главном приложении, и эта строка будет отсылаться приложению дочернему. Т.е. выдает и принимает данные дочернее приложение, а реально они отображаются на экран и вводятся с клавиатуры нашим, главным приложением – как будто всю работу делает главное приложение.

Для решения задачи следует использовать механизм пайпов (pipes): в открытый пайп можно писать, как в любой другой поток вывода с последовательным доступом, или читать из. Поскольку рассматриваемый тип консольных приложений использует стандартные потоки ввода-вывода именно для последовательного доступа (поток ввода – для чтения с клавиатуры, поток вывода – для вывода в окно), пайпом легко подменить стандартные потоки ввода и вывода консольного приложения (т.е., соответственно, клавиатуру и окно консоли). Сделать это можно при запуске дочернего процесса (функция CreateProcess).

Написать GUI-приложение, предоставляющее поле ввода для указания командной строки, например, «cmd.exe /c dir», кнопку «Пуск» для запуска указанной команды и какой-либо визуальный элемент (список, Memo или что-то в этом роде) для просмотра выводимого запущенной командой текста, а также еще одно поле ввода для ввода текста в дочернее приложение. Не забудьте при пересылке строки текста дочернему приложению добавлять в конец строки символ возврата каретки (символ с кодом 10).

Следить за выводом дочернего приложения следует постоянно, т.е. не допускаются ухищрения вроде попытки прочитать что-нибудь из пайпа каждые 100 миллисекунд. Как только в пайпе появляется очередной выведенный приложением текст, этот текст следует сразу же прочитать.