Операционные среды и системное программирование
Л.р.1. Управление процессами, потоками, нитями

Цель:

Возобновление, закрепление и развитие навыков программирования приложений Windows.

Концепции вычислительных процессов, потоков, нитей; реализация в Windows. Основные этапы жизненного цикла процессов (потоков) и элементарное управление ими: порождение, завершение, получение и изменение состояния. Типичное (простое) использование многозадачности и многопоточности

Теоретическая и методическая часть

Вычислительные процессы. Вычислительный процесс как системный объект, объект Process в Windows. Атрибуты процесса. Образ процесса. Адресное пространство и пространство дескрипторов. Состояния процесса.

Вычислительные потоки. Вычислительный поток как системный объект, объект Thread в Windows. Атрибуты потока, в т.ч. приоритеты. Состояния потока. Разделяемые адресное пространство и пространство дескрипторов.

Создание (порождение) процессов. Иерархия процессов: родительские (parent) и дочерние (child) процессы. Завершение процессов. Код завершения.

Создание (порождение) потоков, главный поток. Диспетчирование потоков. Приостановка и возобновление потоков. Завершение потоков. Код завершения.

Нити (Fiber), особенности их выполнения. Взаимосвязь нитей и потоков. Создание, завершение, переключение нитей.

Программный интерфейс (API) для «базового» управления процессами, потоками и нитями.

Практическая часть

Общая постановка задачи:

Приложение (или приложения), демонстрирующие простые ситуации, связанные с управлением процессами, потоками, нитями, в основном их создание, завершение, получение информации о текущем состоянии, манипулирование приоритетами, а также с оценкой влияния на производительность приложения и на работу системы в целом. (Предполагается, что для выполнения задачи не требуется использовать специализированные механизмы взаимодействия, а также обращаться к подсистеме безопасности.)

Специальных требований к приложениям не предъявляется; в частности, во многих случаях они могут быть не обязательно оконными, но также и консольными.

Варианты заданий:

– Самовосстанавливающийся процесс
– Процесс-«диспетчер»
– Упрощенный аналог планировщика cron
– «Диспетчер зомби»
– Приоритеты
– Многозадачная (многопоточная) обработка
– Многопоточная работа с файлом
– Сравнение многопоточной и многонитевой реализаций
– Нагрузочная способность («стресс-тест»)
– ...

1 Самовосстанавливающийся процесс

Возобновление работы процесса с сохранением (продолжением) текущих функций после завершения его сообщением **`WM_CLOSE`**.

Игнорирование сообщения приводит, как правило, либо к объявлению его аварийным (автоматически), либо принудительному безусловному завершению посредством TaskManager.
Решение: процесс завершается штатным образом, но перед этим порождает свою копию (аналогичный процесс из того же исполняемого файла). Копия продолжает выполняться вместо родительского процесса.

Проблема: необходимость продолжить работу с теми же обрабатываемыми данными.
Вариант решения: хранение текущих рабочих данных в структуре («прикладной контекст»), которая может выгружаться на диск или сохраняться в глобальной памяти (родителем) и загружаться оттуда (потомком-«наследником»).

2 Процесс-«диспетчер»

Процесс, выполняющий:
– выбор исполняемого файла, запуск процесса из него
– хранение списка порожденных процессов
– отображение состояния контролируемых процессов (достаточно различать состояния «выполняется», «завершился»)
– возможность послать сообщение **`WM_CLOSE`** выбранному процессу
– отображение возникающих ошибок

В качестве контролируемых процессов можно использовать произвольные подходящие программы либо специально написанный процесс: оконное приложение, способное наглядно показывать свое выполнение.

Проверка состояния процессов – в простейшем случае периодический опрос, например с помощью Wait-функций (WaitForSingleObject(), WaitForMultipleObjects()). При этом проверка не должна блокировать на существенное время выполнение процесса-диспетчера.

3 Упрощенный аналог планировщика cron

Выполнение внешних программ в соответствии с заданным расписанием (конфигурацией), загружаемым из файла/файлов (формат можно заимствовать от cron или использовать свой упрощенный).

Расписание содержит имена исполняемых файлов, время (условие) их выполнения, параметры запуска.

Необходимо обрабатывать возможные ошибки выполнения.

4 «Диспетчер зомби»

Исследование условий появления процессов-«зомби» и способов предотвращения их появления.

Процесс в состоянии «зомби» – после завершения, но до окончательного удаления из системы. «Зомби» не выполняется и не конкурирует за время

процессора, большая часть выделенных для него ресурсов освобождена. Однако информация о соответствующем объекте сохраняется в системных таблицах, что перегружает их и может замедлять работу планировщика.

В Windows процесс остается в этом состоянии до тех пор, пока в системе существует хотя бы один описатель (handle), ссылающийся на этот объект. Следовательно, если родительский процесс закрывает принадлежащий ему handle порожденного процесса и если других открытых handle на этот процесс нет, то процесс после завершения может быть удален окончательно. Но, в то же время, не владея handle своего потомка, процесс-родитель не может управлять им.

Для задания потребуются два процесса:

– процесс-родитель (диспетчер): запуск процесса-потомка, сохраняя или нет его handle (переключатель в интерфейсе), отображение состояния запущенных процессов, закрытие handle завершившихся

– процесс-потомок: демонстрация своего выполнения, завершение по команде (элемент управления), по времени или посредством TaskManager.

Списки и состояния сопоставляются с TaskManager.

5 Приоритеты

Запуск нескольких (можно фиксированное количество) потоков с разными приоритетами (можно задать заранее) и оценка их производительности. Желательно отображение времени работы потока – в течение его выполнения и итоговое после завершения, и объема выполненной потоком работы – в простейшем случае счетчик итераций. «Содержимое» потока – произвольная задача с достаточной вычислительной сложностью (сортировка, умножение матриц и т.п.)

Дополнение: анализ поведения при наличии высокоприоритетных потоков.

6 Многозадачная (многопоточная) обработка

(Обработка – сортировка массива, матричные операции, криптография, статистические расчеты и т.д.)

Разбиение массива данных на несколько частей (фрагментов), обработка каждого отдельным потоком (или процессом), окончательная «сборка» результата. Количество потоков (процессов), в т.ч. единственный поток/процесс, размер массива и другие необходимые параметры задаются пользователем. Количество потоков (процессов) не слишком большое, чтобы оставалось удобным для отображения.

Отображение прогресса выполнения (достаточно готовности/неготовности каждого фрагмента), время выполнения для сравнения и анализа зависимости.

Соотнесение с уровнем загруженности системы (использование ЦП).

7 Многопоточная работа с файлом

Чтение файла несколькими потоками, «сборка» результата. Количество потоков, файл для чтения – выбор пользователем. Количество потоков – аналогично предыдущему (в т.ч. единственный поток).

Оценка времени, зависимость от начальных параметров.

В дальнейшем возможно сравнение с асинхронной реализацией.

8 Сравнение многопоточной и многонитевой реализаций

Сравнение эффективности (производительности) реализации распараллеливаемого алгоритма потоками и нитями (далее в качестве примера умножение матриц, можно подобрать другие алгоритмы).

Пусть размерность матриц – M×M, количество рабочих потоков (нитей) N – в 8..16 раз меньше. Каждый поток (каждая нить) выбирает свои строки и столбцы по определенному правилу (например, «блоками» подряд или чередованием через N).
После обработки очередной пары «строка-столбец» поток выполняет Sleep(1) для переключения на другой поток, нить явно передает управление другой нити.
Результат готов после завершения работы всех потоков (нитей).

Сравнение результатов, в т.ч. и с «линейной» (без распараллеливания) реализацией, соотнесение с уровнем загруженности системы (использование ЦП).

9 Нагрузочная способность («стресс-тест»)

Оценка влияния количества параллельно выполняющихся процессов (потоков) на загруженности системы и ее состояние в целом, оценка предельного количества активных потоков.
Аналогично – для принудительно приостановленных (suspended) потоков.

Алгоритм функций потоков выбирается таким, чтобы минимизировать зависимости между процессами (потоками) и избежать взаимных блокировок.

Также необходимо обеспечить достаточно плавное нарастание нагрузки, позволяющее наблюдать эффект.

10 ...

...