Тестовое задание для Связного FixedThreadPool на C#. Что здесь не так? UPD

**UPDATE:** Я не мог отказать себе в удовольствии исправить свой ошибочный код. Добавил раздел «Работа над ошибками», в котором привожу исправленный код и описание исправлений, основанное на полученных комментариях.  
  
Это скорее пост-вопрос к специалистам, нежели просто кусок полезной информации. Приглашаю к дискуссии.   
Недавно я имел счастье послать своё резюме в Связной на позицию .NET разработчика. В ответ меня попросили сделать тестовое задание на знание многопоточности. Я не могу назвать себя экспертом в этой области, но, тем не менее, прекрасно понял, как мне показалось, как реализовать следующие требования:  
  
Требуется реализация класса на языке C#, аналогичного FixedThreadPool в Java, со следующими требованиями:

* В конструктор этого класса должно передаваться количество потоков, которые будут выполнять задачи.
* Интерфейс класса должен предоставлять методы: boolean execute(Task task, Priority priority) и void stop()
* Интерфейс Task должен содержать один метод: void execute(), который вызывается в произвольном потоке.
* Тип Priority — это перечисление из трёх приоритетов: HIGH, NORMAL, LOW. При этом во время выбора следующего задания из очереди действуют такие правила: на три задачи с приоритетом HIGH выполняется одна задача с приоритетом NORMAL, задачи с приоритетом LOW не выполняются, пока в очереди есть хоть одна задача с другим приоритетом.
* До вызова метода stop() задачи ставятся в очередь на выполнение и метод boolean execute(Task task, Priority priority) сразу же возвращает true, не дожидаясь завершения выполнения задачи; а после вызова stop() новые задачи не добавляются в очередь на выполнение, и метод boolean execute(Task task, Priority priority) сразу же возвращает false.
* Метод stop() ожидает завершения всех текущих задач (не очищая очередь).

Поскольку в задании не было сказано какими примитивами я должен пользоваться, должен ли сделать всё на простейших Thread или же использовать ThreadPool, TPL и т.п., я решил, что задание предполагает использование самых базовых элементов: Thread, ManualResetEvents и т.п. Написал за несколько часов, отослал. Сегодня позвонил и получил ответ через кадровика, который звучал примерно так: «это даже не близко к том, что надо». Это меня озадачило, ибо код работает и протестирован, явных огрехов, на мой взгляд нету.  
  
Итак, на ваш суд представляю мою реализацию FixedThreadPool и сопутствующих классов. Сразу предупреждаю, что, по их мнению реализация ошибочна, и, соответственно, брать мою идею за основу не стоит. Некоторые коментарии по коду:

* я решил инкапсулировать потоки задач в самом классе задачи,
* два параметра с типом ILog нужны только для тестовых целей, к основной функциональности они, понятное дело, отношения не имеют,
* весь проект, включая тестовое приложение можно загрузить по ссылке (27 килобайт): [тестовый проект на ifolder](http://ifolder.ru/31007649) ( классы этого варианта в Solution помечены как INCORRECT)

#### Работа над ошибками

Спасибо большое за комментарии и конструктивную критику. Я решил, что тема будет не закрыта, если не опубликовать исправленное решение. Для начала перечислю рекомендации и опишу какие вошли в код, а какие нет, и почему.

* **Создание потока — операция очень затратная. Пул потоков нужен для того, чтобы хранить в нём несколько уже созданных потоков.** Это было моей главной ошибкой и, судя по всему, именно это вызвало столь негативную реакцию со стороны технического специалиста, проверяющего моё решение. Я просто слишком сконцентрировался на условиях задачи и проглядел требование, которое следовало из самого названия класса. Спасибо, [Unrul](http://habrahabr.ru/users/unrul/), [iaroshenko](http://habrahabr.ru/users/iaroshenko/). Теперь я создаю необходимое количество потоков в конструкторе пула. Класс Task больше не занимается созданием потоков.
* **Поток планировщика не нужен.** Действительно, после того, как создание потока было перенесено из класса задачи в класс пула, стало возможно переложить функцию извлечения задач из очереди на сами потоки. Спасибо, [iaroshenko](http://habrahabr.ru/users/iaroshenko/), [romik](http://habrahabr.ru/users/romik/).
* **В интерфейсе Task не должно быть event.** Поскольку класс задачи больше не занимается созданием потоков, событие окончания обработки задачи теперь действительно не нужен. Спасибо, [Nagg](http://habrahabr.ru/users/nagg/).
* **Блокировку на IsStopped можно заменить обычным volatile поле.** С этим я не согласен. Несмотря на утверждение MSDN о том, что благодаря модификатору volatile читающий значение такого поля код будет получать самое актуальное значение, это не так. На самом деле, если говорить упрощённо, volatile позволяет гарантировать то, что компилятор, CLR и процессор не поменяют местами чтение/запись в volatile поле и другие поля. [Более подробно можно почитать здесь](http://www.albahari.com/threading/part4.aspx#_Memory_Barriers_and_Volatility). Однако в моём случае блокировка IsStopped нужна для другого. Без неё потенциально возможно, что в очередь попадёт новая задача после фактической остановки пула, а также возможен случай, когда поток задачи останется в состоянии ожидания, не смотря на то, что был дан сигнал завершения. В любом случае, небольшая перестраховка в многопоточном коде никогда не повредит. Это лучше, чем потом ловить нерегулярные ошибки. Не стану с пеной у рта защищать это решение. Если кто сможет опровергнуть мой довод на основе кода, буду признателен. Спасибо, [elw00d](http://habrahabr.ru/users/elw00d/).
* **Очередь с приоритетами реализовать отдельно от логики собственно ThreadPool'а.** Этого я тоже не стал делать. Фактически пул содержит не так много кода, там больше комментариев и логгирования. Но, тем не менее, основная причина не в этом. Представьте, что условие отбора изменилось. Или же нужно обеспечить возможность настройки принципа выбора следующей задачи в момент создания экземпляра пула. Придётся создавать новый класс сортированной коллекции (очереди) и, соответственно, плодить по числу классов коллекций классы пулов. Это неправильное решение. В данном случае будет лучше вынести алгоритм выбора следующей задачи в отдельный класс, реализовав шаблон Strategy. Объект «отборщика» в таком случае будет передаваться в конструктор. Но я не буду усложнять своё решение, так как по условию задачи это не требуется. Спасибо, [elw00d](http://habrahabr.ru/users/elw00d/).
* **Interlocked-методы тоже неясно зачем (если уже используются мониторы и event'ы).** Эту часть я тоже не стал трогать. Interlocked-методы позволяют избежать нарушение условий отбора задач с приоритетами High и Normal. Поскольку нет блокировок на доступ к счётчику mQueuedHighPriorityTaskCounter я предпочёл на всякий случай перестраховаться, использовав Interlocked-методы. Возможно более глубокий анализ кода выявит мою неправоту, но как я уже говорил, я не самый большой специалист в многопоточности и предпочитаю подстелить соломку, тем более когда она стоит дёшево. Спасибо, [elw00d](http://habrahabr.ru/users/elw00d/).
* **Было бы очень интересно, если бы автор решил озвученные в комментариях недочеты и снова отправил в Связной обновленную версию решения.** К коду это конечно имеет лишь опосредованное отношение, но тем не менее. Думаю, я не стану отправлять исправленное решение в Связной. Это будет нечестно, поскольку я воспользовался вашей помощью. Скорее всего, я отправлю им ссылку на эту статью, а там пусть сами решают нужен ли им «уже готовый специалист» или подходит человек, который умеет признавать свои ошибки и развиваться, двигаться дальше. Спасибо за поддержку, [ARC\_Programmer](http://habrahabr.ru/users/arc_programmer/).