РЕАЛИЗАЦИЯ ПОТОКОВ В ПОЛЬЗОВАТЕЛЬСКОМ ПРОСТРАНСТВЕ И ЯДРЕ

Первый способ — это поместить весь набор потоков в пользовательском пространстве. И об этом наборе ядру ничего не известно. Что касается ядра, оно управляет обычными, однопотоковыми процессами. Первое и самое очевидное преимущество состоит в том, что набор потоков на пользовательском уровне может быть реализован в операционной системе, которая не поддерживает потоки. Также Они позволяют каждому процессу иметь собственные настройки алгоритма плани- рования. Например, для некоторых приложений, которые имеют поток сборщика мусора, есть еще один плюс — им не следует беспокоиться о потоках, остановленных в неподходящий момент. Эти потоки также лучше масштабируются, поскольку потоки в памяти ядра безусловно требуют в ядре пространства для таблицы и стека, что при очень большом количестве потоков может вызвать затруднения.

Но несмотря на лучшую производительность, у потоков, реализованных на пользова- тельском уровне, есть ряд существенных проблем. Первая из них — как реализовать блокирующие системные вызовы. Представьте, что поток считывает информацию с клавиатуры перед нажатием какой-нибудь клавиши. Мы не можем разрешить по- току осуществить настоящий системный вызов, поскольку это остановит выполнение всех потоков. Одна из главных целей организации потоков в первую очередь состояла в том, чтобы позволить каждому потоку использовать блокирующие вызовы, но при этом предотвратить влияние одного заблокированного потока на выполнение других потоков. Работая с блокирующими системными вызовами, довольно трудно понять, как можно достичь этой цели без особого труда.

Использование набора потоков, реализованного на пользовательском уровне, связано еще с одной проблемой: если начинается выполнение одного из потоков, то никакой другой поток, принадлежащий этому процессу, не сможет выполняться до тех пор, пока первый поток добровольно не уступит центральный процессор. Другой наиболее сильный аргумент против потоков, реализованных на пользователь- ском уровне, состоит в том, что программистам потоки обычно требуются именно в тех приложениях, где они часто блокируются, как, к примеру, в многопоточном веб-сервере. Эти потоки часто совершают системные вызовы.

РЕАЛИЗАЦИЯ ПОТОКОВ В ЯДРЕ

здесь уже не нужна система поддержки ис- полнения программ. Также здесь нет и таблицы процессов в каждом потоке. Вместо этого у ядра есть таблица потоков, в которой отслеживаются все потоки, имеющиеся в системе. Когда потоку необходимо создать новый или уничтожить существующий поток, он обращается к ядру, которое и создает или разрушает путем обновления та- блицы потоков в ядре.

В таблице потоков, находящейся в ядре, содержатся регистры каждого потока, состояние и другая информация. Вся информация аналогична той, которая ис- пользовалась для потоков, создаваемых на пользовательском уровне, но теперь она содержится в ядре, а не в пространстве пользователя

Для потоков, реализованных на уровне ядра, не требуется никаких новых, неблокиру- ющих системных вызовов. Более того, если один из выполняемых потоков столкнется с ошибкой обращения к отсутствующей странице, ядро может с легкостью проверить наличие у процесса любых других готовых к выполнению потоков и при наличии та- ковых запустить один из них на выполнение, пока будет длиться ожидание извлечения запрошенной страницы с диска. Главный недостаток этих потоков состоит в весьма существенных затратах времени на системный вызов, поэтому, если операции над по- токами (создание, удаление и т. п.) выполняются довольно часто, это влечет за собой более существенные издержки.