Современные средства разработки ПО

Тестирование и отладка многопоточных приложений.

Фетисов Михаил Вчеславович

Типы ошибок, связанных с параллелизмом

- Кроме обычных ошибок в параллельной программе могут быть специфические, которые попадают в одну из следующих категорий:
  - нежелательное блокирование,
  - состояния гонки.

Нежелательное блокирование
Определение

- **Нежелательное блокирование** — состояние потока, когда он не может продолжать выполнение, так как чего-то ждет.

Нежелательное блокирование
Варианты

- **Взаимоблокировка —** когда один поток ждет другого, а тот, в свою очередь, ждет первого.
- **Активная блокировка —** взаимоблокировка при цикле активной проверки, например спинлоке.
  - В одних случаях программа потребляет много процессорного времени, потому что потоки блокируют друг друга, продолжая работать.
  - В других — блокировка рано или поздно может «рассасаться» из-за вероятностной природы планирования, но заблокированная задача испытывает ощутимые задержки, характеризуемые высоким потреблением процессорного времени.
- **Блокировка в ожидании завершения ввода/вывода или поступления данных из внешнего источника —** если поток блокируется в ожидании данных из внешнего источника, то он не может продолжать работу, даже если данные так никогда и не поступят.
  - Поэтому крайне нежелательно, когда такая блокировка происходит в потоке, от работы которого зависят другие потоки.

Состояния гонки
Варианты

- **Гонка за данными** — это особый тип гонки, который приводит к неопределенному поведению из-за несинхронизированного одновременного доступа к разделяемой ячейке памяти.
  - Обычно гонка за данными возникает вследствие неправильного использования атомарных операций для синхронизации потоков или в результате доступа к разделяемым данным, не защищенного подходящим мьютексом.
- **Нарушение инвариантов** — такие гонки могут проявляться в форме висячих указателей (другой поток уже удалил данные, к которым мы пытаемся обратиться), случайного повреждения памяти (из-за того, что поток читает данные, оказавшиеся несогласованными в результате частичного обновления) и двойного освобождения (например, два потока извлекают из очереди одно и то же значение, и потом оба удаляют ассоциированные с ним данные).
- **Проблемы со временем жизни** — такого рода проблемы можно было бы отнести к нарушению инвариантов, но на самом деле это отдельная категория.
  - Основная проблема в том, что поток живет дольше, чем данные, к которым он обращается, поэтому может попытаться получить доступ к уже удаленным или разрушенным иным способом данным. Не исключено также, что когда-то отведенная под эти данные память уже занята другим объектом. Обычно такие ошибки возникают, когда поток хранит ссылки на локальные переменные, которые вышли из области видимости до завершения функции потока, но это не единственный сценарий.

Методы поиска ошибок, связанных с параллелизмом
Анализ кода на предмет выявления потенциальных ошибок

- Использование доступных инструментов (желательно в CI/CD):
  - «выкрутить» диагностику компилятора на максимум;
  - прогнать в однопоточном режиме, чтобы выявить связь с многопоточностью;
  - прогнать в нагрузочном режиме, максимально увеличив число возможных потоков;
  - прогнать через статический анализатор;
  - прогнать через специальный профилировщик.
- «Ручной» просмотр:
  - посмотреть самому,
  - показать другу,
  - объяснить кому-то, как работает ваш код (можно плюшевому мишке или утёнку).

Анализ кода на предмет выявления потенциальных ошибок

- Над какими вопросами следует задуматься при анализе многопоточного кода. Подобные вопросы от Энтони Уильямса:
  - Какие данные нужно защищать от одновременного доступа?
  - Как вы обеспечиваете защиту этих данных?
  - В каком участке программы могут в этот момент находиться другие потоки?
  - Какие мьютексы удерживает данный поток?
  - Какие мьютексы могут удерживать другие потоки?
  - Существуют ли ограничения на порядок выполнения операций в этом и каком-либо другом потоке? Как гарантируется соблюдение этих ограничений?
  - Верно ли, что данные, загруженные этим потоком, все еще действительны? Не могло ли случиться, что их изменили другие потоки?
  - Если предположить, что другой поток может изменить данные, то к чему это приведет и как гарантировать, что этого никогда не случится?

Методы поиска ошибок, связанных с параллелизмом

- Поиск связанных с параллелизмом ошибок путем тестирования:
  - проверить в однопоточном режиме,
  - протестить в многопоточном.

Проектирование с учетом тестопригодности

- Обязанности всех функций и классов четко очерчены.
- Каждая функция короткая и решает ровно одну задачу.
- Тесты способны полностью контролировать окружение тестируемого кода.
- Код, выполняющий конкретную тестируемую операцию, находится приблизительно в одном месте, а не разбросан по всей системе.
- Автор сначала думал о том, как будет тестировать код, а только потом приступал к его написанию.
- Применялись другие приёмы построения адаптивного кода.

Приемы тестирования многопоточного кода

- **Тестирование грубой силой** — код прогоняется многократно на большом количестве потоков (например, стресс-тест).
  - Позволяет «поймать» редкие ошибочные ситуации.
  - Недостаток метода грубой силы в том, что он может вселять ложную уверенность.
- **Комбинаторное имитационное тестирование** — прогонять код под управлением специальной программы которая имитирует реальную среду.
- **Обнаружение возникающих во время тестирования проблем с помощью специальной библиотеки**

Вопросы?

Фетисов Михаил Вячеславович
fetisov.michael@bmstu.ru