Лекция 6. Основные проблемы многопоточного программирования.

Содержание

- Общие проблемы многопоточности
- Проблемы с разделяемой памятью
- Механизмы синхронизации
- Проблемы с синхронизацией потоков

Общие проблемы многопоточности

Классификация проблем многопоточности

- Проблемы синхронизации, связанные с доступом к общим данным.
- Проблемы коммуникации, связанные с временны́ми задержками при взаимодействии потоков.
- Проблемы балансировки нагрузки между вычислительными устройствами.
- Проблемы масштабируемости. Нагрузит ли программа все ядра `Intel Phi`?

Общие проблемы многопоточности

Мы рассматриваем систему с общей памятью (сильносвязанную).

- Проблема ли это, если два и более потоков пытаются изменить значение одной переменной?

Общие проблемы многопоточности

Мы рассматриваем систему с общей памятью (сильносвязанную).

- Проблема ли это, если два и более потоков пытаются изменить значение одной переменной?
- Нет, если это не нарушает логику программы.

```
atomic<int> A,B,C;
// thread t1
   for (int i = 0; i < 10000; i++) {
      C++;
      B++;
   }

//thread t2
   for (int i = 0; i < 10000; i++) {
      B++;
      C++;
   }
```

Общие проблемы многопоточности

```
atomic<int> int A, B, C;
//thread t1
   for (int i = 0; i < 10000; i++) {
      A++;
      C += B;
   }

//thread t2
   for (int i = 0; i < 10000; i++) {
      B++;
      C += A;
   }
```

Чему будет равно значение переменных A, B, C после исполнения?

Критерий Бернстайна

Пусть $R(P1)$ множество переменных, значение которых поток (процесс) $P1$ использует в операциях чтения, через $W(P1)$ - множество переменных, использующихся в операциях записи. Тогда совокупное исполнение $P1$ и $P2$ детерминировано, если

$$W(P1) \cap W(P2) = \emptyset$$

$$R(P1) \cap W(P2) = \emptyset$$

$$W(P1) \cap R(P2) = \emptyset$$

Применение критерия Бернстайна для примера

Множества входных и выходных переменных для $P1$ есть

$$R(P1) = \{A, B, C\}$$

$$W(P1) = \{A, C\}$$

Для $P2$ такие множества есть

$$R(P2) = \{A, B, C\}$$

$$W(P2) = \{B, C\}$$

$$W(P1) \cap W(P2) = \{C\} \neq \emptyset$$

$$R(P1) \cap W(P2) = \{B, C\} \neq \emptyset$$

$$W(P1) \cap R(P2) = \{A, C\} \neq \emptyset$$

Согласно критерию Бернстайна, набор активностей программы не является явным образом детерминированным.

Проблемы синхронизации в системах с общей памятью

- **Условия гонок, *race conditions*** — проявление недетерминизма исполнителя программы при различном относительном порядке исполнения команд в различных потоках.
- **Взаимная блокировка, *deadlock*** — каждый из потоков ожидают событий, которые могут предоставить другие потоки.
- **Ожидание на блокировках, *Lock Contention*** — основное время потока проводится не в исполнении полезной работы, а в ожидании блокированного другим потоком ресурса. Вариант — **инверсия приоритета**.
- **Активная блокировка, *Live Lock*** — поток захватывает ресурс, но после того, как убедится, что завершить работу не может, освобождает ресурс, аннулируя результаты (*rollback*).

Примитивы синхронизации

- **Критическая секция** — временной участок исполнения программы, в котором требуется обеспечить детерминированность результата. В критической секции может находиться ровно один поток.
- Критическая секция — абстракция.
- Реализация критической секции требует аппаратной поддержки (`CAS` или аналога).
- Производится **сериализация** исполнения операций.

Проблема ABA

Возникновение:

- Потоку требуется использование нескольких объектов $O_1, O_2, \ldots, O_n$, находящихся в зависимости (имеется инвариант $I(O_1, O_2, \ldots, O_n) = true$).
- Поток начинает транзакцию. Если после окончания запланированной операции изменения инвариант будет сохранён, то транзакция свершается (`commit`), иначе не фиксируется (`rollback`).
- Поток сохраняет A одного из объектов (O_k)
- Поток изменяет состояние объекта $O_j, j \neq k$
- После обработки объекта O_j поток сравнивает значение O_k с вновь запрошенным объектом $O_{k'}$.
- Если $O_k == O_{k'}$, то поток полагает, что ничего не произошло, инвариант удовлетворяется и фиксирует изменения (свершает транзакцию).

Проблема ABA

А в это время . . .

- Другой поток решает изменить несколько объектов O_k.
- Для этого он изменяет значение O_k и записывает туда B.
- После этого он изменяет ещё несколько объектов и в O_k записывает A.

Первый поток полагает, что если O_k не изменился, то и остальные объекты не изменились и получать их значения не требуется. Катастрофа.

Варианты использования синхронизации

- Блокировать исполняемый код. Мониторы. Windows95.
- Блокировать обращения к конкретным данным (ресурсные скобки). Гранулярность.
- Явная и неявная синхронизации.

Варианты ожидания разблокировки

- Активное ожидание. Нет переключения потоков. Всё время ожидания расходуется на процессор.
- Ожидание события. Нет загрузки процессора. Есть переключение потоков.

Примитивы синхронизации

- Много примитивов в ядре `Windows` (21 вариант).
- Вся внутренняя организация основана на `CAS`.
- Подход с эксклюзивным доступом (`mutex`) или с пулом ресурсов (`semaphore`).
- Сколько требуется базовых примитивов? Сколько памяти потребуется на реализацию примитивов? Какую поддержку примитивов нужно заложить в систему команд процессора?

Проблемы работы с примитивами

- Инверсия приоритета. Низкоприоритетный процесс захватил блокировку. Он получает мало тактов процессора. Высокоприоритетный процесс ожидает освобождения блокировки от низкоприоритетного и также получает мало тактов.
- Конвоирование. Остановка блокирующего процесса приводит к остановке остальных, использующих этот ресурс.
- Тупики.
- Скорость. `CAS` — медленная операция. `CAS` работает только со словами, не с объектами. Алгоритмы `CAS` сложны.
- Проблемы композиции примитивов.

Условия возникновения тупиков

- Тупик — ситуация, возникающая когда два или более потока пытаются завладеть двумя или более ресурсами, проявляющаяся бесконечным взаимным ожиданием.
- При использовании совместного ресурса возникают фазы:
  - блокировки
  - использования
  - разблокировки

Состояние тупика

- Тупик — состояние взаимного ожидания параллельно исполняющихся вычислительных потоков, ожидающих разблокировку одних и тех же ресурсов.
- Тупики трудно диагностируются и появление состояния тупика в ядре операционной системы обычно означает отказ какого-то её компонента или всей системы.
- Процесс или поток находится в состоянии тупика, если он ожидает события, которое никогда не произойдёт.

Состояние тупика

- Множество потоков находится в тупиковой ситуации, если каждый поток ожидает события, которое только другой поток данного множества может предоставить.
- Так как все потоки находятся в состоянии ожидания, то ни один из них не может инициировать событие, которое могло бы активировать другого члена множества, и, следовательно, ситуация не может быть изменена.
- Потеря сигнала может привести к тупику.

Условия возникновения тупиков

1. Условие взаимоисключения. Одновременно использовать ресурс может только один поток.
2. Условие ожидания ресурсов. Потоки удерживают выделенные им ресурсы и могут запрашивать другие ресурсы.
3. Условия нераспределяемости. Ресурс, выделенный одному потоку не может быть принудительно у него забран. Только поток, захвативший ресурс может его освободить.
4. Условие кругового ожидания. Существует кольцевая цепь потоков, в которой каждый поток ждёт доступа к ресурсу, удерживаемому другим потоком цепи.

Для образования тупика необходимым и достаточным является выполнение всех четырёх условий.

Спасибо за внимание.

Следующая лекция — анализ программ на ошибки синхронизации.