Павел Степанов

Кафедра компьютерной математики и программирования СПб ГУАП

# Язык программирования Java

# Тема 4 Многопоточность

- Понятие многопоточности
- Примитивы синхронизации
- Проблемы
- Java.util.concurrent

# 4.1 Процесс и поток (POSIX-2001)

- Процесс (process) – состоит из единого адресного пространством, в котором выполняются потоки, этих потоков и ресурсов, принадлежащих этим потокам
- Поток (thread) – совокупность содержимого машинных регистров и стека.
  - Если машинные регистры загружены в какой-либо процессор, то поток выполняется

# 4.2 Бестиарий

- Главный поток
  - Завершение программы
- Присоединенные потоки
- Демоны
- Зомби

# 4.3 Диспетчер потоков

- Java полностью передает управление потоками операционной системе
  - Если она многопоточная, конечно
- По прерыванию наносекундного таймера все регистры сбрасываются в стек и управление получает планировщик.
- Планировщик переставляет указатель стека на тот поток, который должен получить управление и завершается.

# 4.4 Java-потоки

- Высокоуровневая абстракция – объект Thread
- Низкоуровневая реализация
  - Отображение в потоки платформы, если она это поддерживает
  - Либо специальная эмулирующая библиотека, если не поддерживает

# 4.5 Основные проблемы

- Синхронизация
- Разделение памяти
- Тупиковые ситуации (deadlock)
- Голодание (starvation)
- Активные блокировки (livelock)

# 4.6 Thread

```
public class SampleThread extends Thread{

    public void run() {
        //код треда
    }

    public static void main(String args[]) {
        new SampleThread().start();
    }

}
```

# 4.7 Runnable

```
public class SampleRunnable implements Runnable {

    public void run() {
        //код треда
    }

    public static void main(String args[]) {
        (new Thread(new SampleRunnable())).start();
    }

}
```

# 4.8 В чем разница?

```
public class SampleRunnable implements Runnable {

    public void run() {
        //код треда
    }

    public static void main(String args[]) {
        SampleRunnable code = new SampleRunnable();
        new Thread(code).start();
        new Thread(code).start();
        new Thread(code).start();
        new Thread(code).start();
    }

}
```

# 4.9 Thread.sleep()

- Прекращает работу на ПРИМЕРНО указанное в параметре время
- Не освобождает монитор!!!
- InterruptedException
- Thread.interrupt()
  - Специальный случай - блокировка на потоковом I/O
  - ClosedByInterruptException

# 4.10 Дополнительные аспекты

- Thread.join()
- Thread.currentThread()
- Thread.setDaemon()
- Thread.yeld()

# 4.11 Приоритеты потоков

- NORM_PRIORITY=5
- MAX_PRIORITY=10
- MIN_PRIORITY=1

# 4.12 Проблема thread interference

```
class Counter implements Runnable {

    private int c = 0;

    public void run() {

        for (;;) {
            c++;
            c--;
            System.out.println(c);
        }
    }

    public static void main(String[] args) {
        Counter counter = new Counter();
        new Thread(counter).start();
        new Thread(counter).start();
    }
}
```

# 4.13 Проблема

- Хорошая новость – предыдущий пример действительно работает не правильно
- Плохая новость – он работает не так, как вы ожидали

# 4.14 Атомарность

```
x++;        //  aload_0
            //  dup
            //  getfield        #2
            //  iconst_1
            //  iadd
            //  putfield        #2

x++;        //  aload_0
            //  dup
            //  getfield        #2
            //  iconst_1
            //  iadd
            //  putfield        #2
```

# 4.15 Проблема целостности памяти

- “Happens before”
- Барьеры памяти, оптимизация чтения и записи
- Объявление volatile

# 4.16 Понятие монитора

- Состав монитора (Хансен)
  - Мьютекс
  - Процедуры
  - Переменные
  - Условие
- Семантика Хоара
- Семантика Mesa

# 4.17 Синхронизация

- Объявление volatile
- Мониторы, секция synchronized
  - Память при прохождении synchronized
  - Вложенная синхронизация
- Методы wait notify notifyAll
  - При вложенной синхронизации
- Разница wait и sleep

# 4.18 Состояние потока

- New
- Runnable
- Blocked
  - Не то же самое, что блокировка на I/O
- Waiting
- Timed waiting
- Terminated

# 4.19 AtomicInteger

```
/**
 * Atomically sets the value to the given updated value
 * if the current value {@code ==} the expected value.
 *
 * @param expect the expected value
 * @param update the new value
 * @return true if successful. False return indicates that
 * the actual value was not equal to the expected value.
 */
public final boolean compareAndSet(int expect, int update) {
    return unsafe.compareAndSwapInt(this, valueOffset, expect, update);
}

/**...*/
public final boolean weakCompareAndSet(int expect, int update) {
    return unsafe.compareAndSwapInt(this, valueOffset, expect, update);
}

/**
 * Atomically increments by one the current value.
 *
 * @return the previous value
 */
public final int getAndIncrement() {
    for (;;) {
        int current = get();
        int next = current + 1;
        if (compareAndSet(current, next))
            return current;
    }
}
```

# 4.20 Дедлоки (deadlock)

- Пример кода
- Двудольный граф ожидания
- Эвристики для избегания дедлоков
- Средства обнаружения

# 4.21 Другие проблемы

- Ливлоки (Livelock)
- Голодание (Starvation)

# 4.22 Дополнительные аспекты

- Guarded block
- Immutable objects

# 4.23 Пакет java.util.concurrent

- Lock и ReentrantLock
  - Метод tryLock не блокирует процесс
- Executor, Callable и Future
  - Executor поддерживает пулы потоков
  - Callable позволяет потоку возвращать результат
  - Future представляет этот результат
- ForkJoinPool
- Многопоточные коллекции
- ThreadLocalRandom (JDK 7+)

# 4.24 Q & A