Критическая задача №2

Васько Мария, 22126, @tgentrepreneur

Формулировка

Easy. Prove that Regular MRSW Integer construction is correct and wait-free.

Описание

Ниже приведён псевдокод реализации RegMRSWRegister для Integer:

```Java
public class RegMRSWRegister implements Register<Integer> {
  RegMRSWRegister[M] r_bit;

  public void write(Integer x) {
    r_bit[x].write(true);
    for (int i = x - 1; i >= 0; i--)
        r_bit[i].write(false);
  }

  public Integer read() {
    for (int i = 0; i < range; i++)
      if (r_bit[i].read()) return i;
    return -1;
  }
}
```

- `write(i)`:
  1. Устанавливает значение в конкретный бит по индексу `i`. Первым выставляется `true` в `r_bit[i]` и затем в `r_bit[j]` — `false` для $\forall j < i$.
  2. После завершения `write(i)`, в массиве `r_bit` гарантированно:
     1. `r_bit[i] = true`
     2. $\forall j < i$: `r_bit[j] = false`
- `read()` проходит с начала `r_bit`, т.е. слева направо, и возвращает первый индекс `i`, где `r_bit[i] = true`.

Корректность

Определение. Будем называть *регулярным* тот регистр `R`, для которого выполнены следующие условия:

1. Если вызов `read()` не пересекается с вызовом `write(i)`, то вызов `read()` возвращает последнее записанное значение.
2. Если вызов `read()` пересекается с вызовом `write(i)`, то вызов `read()` возвращает либо последнее записанное значение, либо новое значение.

Доказательство. Рассмотрим обе ситуации из определения:

1. Нет пересечения:

- `writer`: пишет значение в конкретный бит по индексу `i`. Первым выставляется `true` в `r_bit[i]` и затем в `r_bit[j]` — `false` для $\forall j < i$.
- `reader`: проходит с начала `r_bit`, т.е. слева направо. Тогда возвращенный `i` — индекс последнего измененного бита (записанный не пересекающим `write`'ом), `r_bit[i]` $\forall j < i$ очищены.
- Так как нет пересечения, то есть запись завершена, то `reader` мог встретить только то значение, которое было в последний раз выставлено.

2. Пересечение есть:
   - `writer`: пишет значение в конкретный бит по индексу `i`. Первым выставляется `true` в `r_bit[i]` и затем в `r_bit[j]` — `false` для $\forall j < i$.
   - `reader`: проходит с начала `r_bit`, т.е. слева направо. Тогда возможны два варианта:
     - `read()` успевает прочитать `r_bit[j] = true`, где $j < i$, до того, как `write(i)` сбросит этот бит → `read()` вернёт старое значение
     - `read()` видит `r_bit[i] = true`, установленное `write(i)` → `read()` вернёт новое значение
   - Таким образом, если вызов `read()` пересекается с вызовом `write(i)`, то в момент чтения `reader` может вернуть либо последнее записанное значение, либо новое значение.
   - Что, соответственно, не нарушает определение регулярности.

Т.к. в регистр писать может только один поток — иных ситуаций не будет.

Wait-free свойство

Определение. Будем считать, что регистр `R` обладает свойством `wait-free`, если вызовы `read()` и `write(i)` завершаются за конечное число шагов.

Доказательство.

- `write`: Вызовы `r_bit[i].write(bool)` являются не блокирующими и завершаются за конечное число шагов. Цикл также завершается за конечное число шагов.
- `read`: Вызовы `r_bit[i].read()` так же не являются блокирующими. Цикл завершается за конечное число шагов.
- Итого: регистр `R` обладает свойством `wait-free`.