monitor-bufferdoppio

Esercizio 2: Buffer di accumulazione doppio

Si consideri il seguente problema. Esistono due classi di oggetti chiamati A e B. Gli oggetti di queste due classi vengono prodotti da un insieme di produttori P_{AB} e consumati da due insiemi di consumatori C_A e C_B. I produttori producono un oggetto di classe A e un oggetto di classe B con una singola chiamata al monitor DoubleBuf. I consumatori in C_A consumano oggetti di classe A e i consumatori in C_B consumano oggetti di classe B. Gli oggetti vengono mantenuti in due buffer limitati (entrambi di dimensione N, con N pari) controllati dal monitor DoubleBuf. Vi sono numerosi produttori/consumatori (il numero di istanze di ogni categoria supera N).

Il monitor è caratterizzato da due periodi: riempimento e svuotamento. Durante il riempimento è possibile solo aggiungere oggetti ai buffer, durante lo svuotamento è possibile solo rimuovere oggetti. Quando il buffer è vuoto, i produttori inseriscono oggetti fino a riempirlo. Quando il buffer è pieno, i consumatori consumano oggetti fino a svuotarlo. I consumatori devono alternarsi nel modo seguente: prima viene consumato un oggetto A, poi viene consumato un oggetto B, poi ancora un oggetto B, poi un oggetto A, e così via fino allo svuotamento del buffer.

```
process P_AB {
  while (true) {
    A a = produceA();
    B b = produceB();
    DoubleBuf.add(a, b);
  }
}
```

```
process C_A {
  while (true) {
    A a =
      DoubleBuf.getA();
    consume(a);
  }
}
```

```
process C_B {
  while (true) {
    B b =
      DoubleBuf.getB();
    consume(b);
  }
}
```

a) Scrivere il monitor DoubleBuf
b) Sono possibili situazioni di deadlock?
c) Sono possibili situazioni di starvation?

```
monitor DoubleBuffer {
  // Costanti: RIEMPIMENTO, SVUOTAMENTO, N, AA, BB
  // Indici per scorrere i buffer e gestire i turni;
  int index  = 0;
  int indexA = 0;
  int indexB = 0;
  // Buffer
  A[] bufA = new A[N];
  B[] bufB = new B[N];
  // Stato del monitor
  int status = RIEMPIMENTO;

  // Condition di attesa
  condition c_add, c_getA, c_getB;

  procedure entry void add(A a, B b)
  {
    if (status == SVUOTAMENTO)
      c_add.wait();
    bufA[index] = a; bufB[index] = b;
    index = index++;
    if (index == N) {
      status = SVUOTAMENTO;
      indexA = indexB = 0;
      turn = 0;
    }
    sveglia();
  }

  // getB è simmetrico
  procedure entry A getA()
  {
    if (status == RIEMPIMENTO || next(turn) == BB)
      c_getA.wait();
    A ret = A[indexA];
    indexA++;
    turn++;
    if (indexA == N && indexB == N) {
      status = RIEMPIMENTO;
      index = 0;
    }
    sveglia();
    return ret;
  }

  void sveglia()
  {
    if (status == RIEMPIMENTO) {
      c_add.signal();
    } else {
      if (next(turn)==AA)     c_getA.signal();
      else                    c_getB.signal();
    }
  }

  // Versione con alternanza A B B A
  int next(int turn)
  {
    return (turn%4==0 || turn%4==3) ? AA : BB;
  }
```