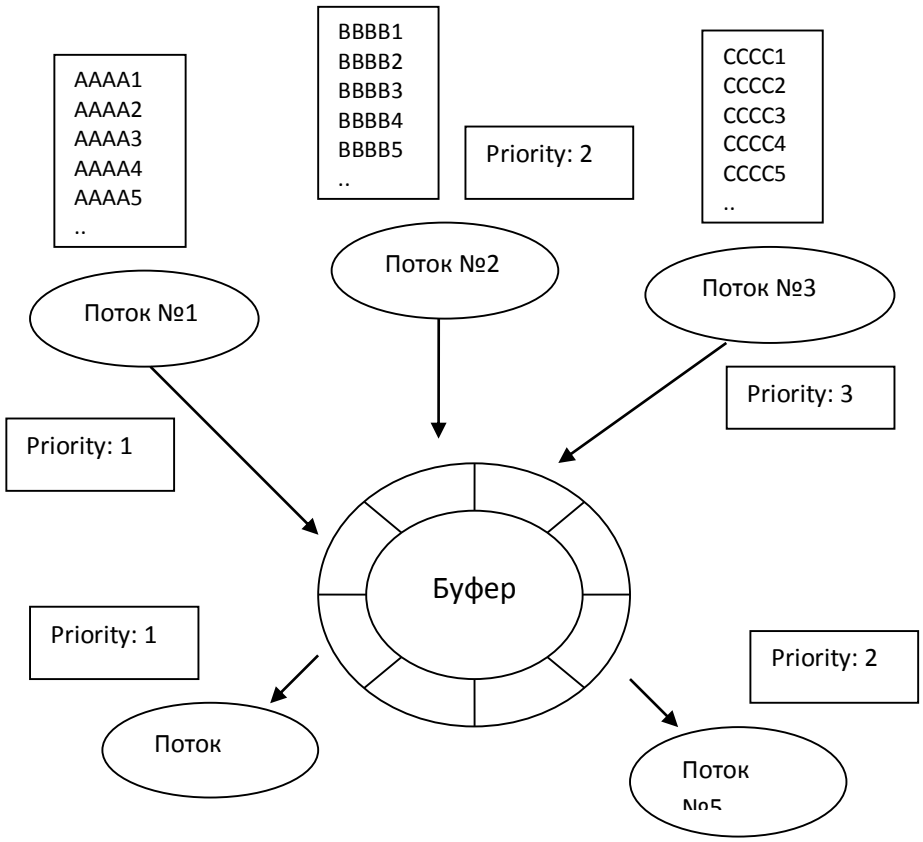
Лабораторная работа. Синхронизация приоритетного доступа к многоэлементному буферу

Задача

Несколько потоков работают с общим многоэлементным буфером. Потоки делятся на «читателей» и «писателей», каждый поток обладает приоритетом. Писатели осуществляют запись в буфер, если есть свободные ячейки. Читатели извлекают содержимое буфера, если есть заполненные ячейки. Работа приложения заканчивается после того, как все сообщения писателей будут обработаны читателями через общий буфер. В качестве буфера используется «кольцевой массив».



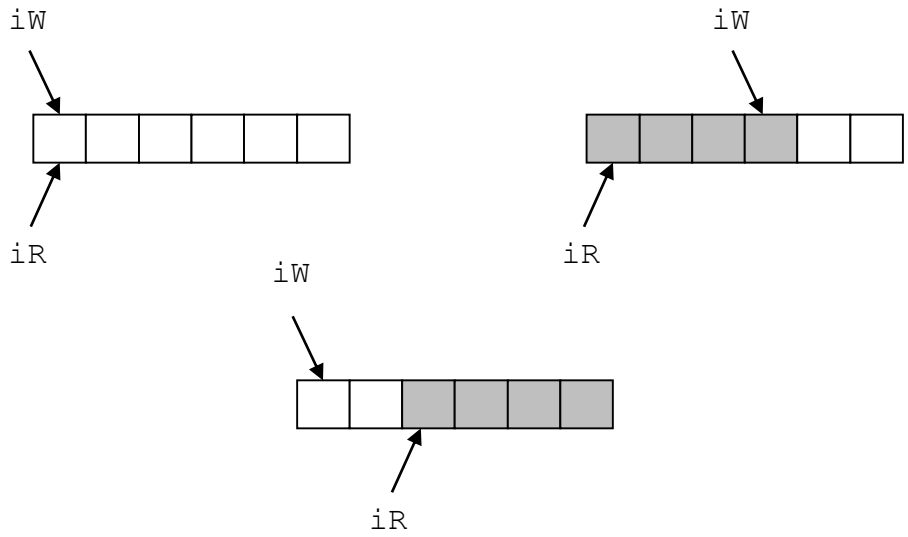
Задания

1. Реализуйте синхронизированное взаимодействие читателей и писателей с учетом приоритета. Аргументируйте выбор средств синхронизации.
2. Вывод программы включает: время работы каждого писателя и читателя; число сообщений, обработанных каждым писателем и читателем.
3. Выполните прогон программы при разных параметрах: разном числе писателей и читателей, разном объеме сообщений, разных приоритетах потоков. Результаты прогонов представьте в табличной форме.

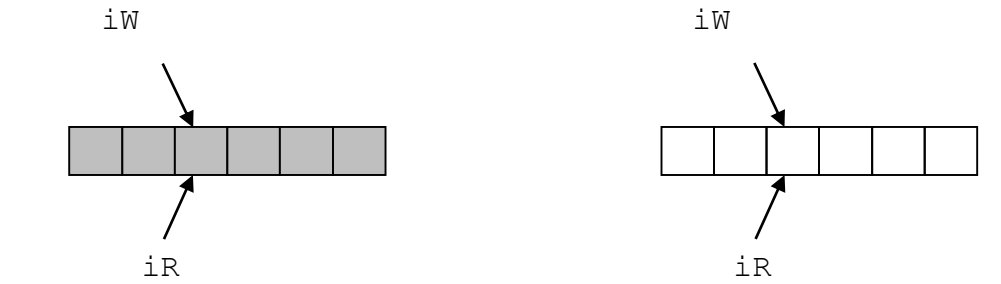
Методические указания

В качестве многоэлементного буфера используется кольцевой массив. Он представляет собой обычный массив размера n. Буфер называется кольцевым, так как при смещении текущего индекса после крайнего элемента следует первый. Для доступа к буферу используются два индекса: один для чтения и один для записи. Такая организация обеспечивает независимость операций чтения и записи – если в массиве есть свободные элементы и есть заполненные элементы, то операции чтения и записи могут производиться одновременно без каких-либо средств синхронизации.

В начале работы буфер является пустым – оба индекса указывают на первый элемент. При осуществлении операций чтения или записи соответствующие индексы смещаются.



Операция чтения блокируется, если буфер пуст, запись при этом разрешена. Операция записи блокируется, если буфер полностью заполнен, чтение при этом разрешено. Равенство индексов чтения и записи является признаком и занятости буфера, и пустоты. Чтобы различать эти ситуации необходимо контролировать, какая операция привела к равенству индексов. Если операция записи, то буфер заполнен.



Операции чтения и записи могут быть реализованы следующим образом:

```
bool Write(string Msg)
{
     if(bFull)
          return false;
     buffer[iW] = Msg;
     iW = (iW + 1) % n;

     // Если индексы совпали после записи,
     // буфер заполнен
     if(iW == iR)
          bFull = true;
     return true;
}

bool Read(ref string Msg)
{
     // Если индексы совпадают, но не после операции записи
     // буфер пуст
     if(iW == iR && !bFull)
          return false;

     Msg = buffer[iR];
     iR = (iR + 1) % n;

     // Если буфер был заполнен, то снимаем отметку
     if(bFull)
          bFull = false;
     return true;
}
```

Главный поток контролирует статус завершения операций чтения и записи. Если операция чтения не выполнена, то поток читателя блокируется.

Ситуация усложняется, если доступ к буферу осуществляют несколько читателей и несколько писателей. Один из вариантов решения проблемы – добавить конструкции критической секции в функции чтения и записи.

Другой подход заключается в реализации схемы «управляющий-рабочие», где управляющий контролирует все операции, требующие синхронизации. Рабочие потоки (читатели и писатели) обращаются к управляющему (основной поток) с сигналом о готовности осуществлять операцию чтения или записи. Управляющий поток фиксирует обращения читателей и писателей, вычисляет текущие индексы для чтения и записи, контролирует состояние буфера (полностью заполнен или полностью пуст), выбирает читателя и писателя, которым разрешает доступ. Операции чтения и записи по корректным

индексам, полученным от управляющего потока, осуществляются читателями и писателями уже без контроля.

Взаимодействие рабочих и управляющего удобно организовать с помощью сигнальных сообщений типа `ManualResetEventSlim`.

Сигналы о готовности `evReadyToRead, evReadyToWrite` генерируют читатели и писатели, готовые осуществлять операции с буфером. Управляющий контролирует состояние сигналов у каждого рабочего.

Сигналы о возможности операций чтения и записи `evStartReading, evStartWriting` генерируются управляющим потоком конкретным читателям и писателям. Перед генерацией сигналов управляющий вычисляет индекс чтения или записи и сохраняет его в индивидуальной ячейке конкретного рабочего.

Такая организация взаимодействия позволяет достаточно легко изменять правила доступа: вводить приоритеты читателей и писателей, учитывать время обращения к управляющему потоку и обеспечивать «справедливость» доступа в плане очередности.

```
void ReaderThread(int iReader,
          ManualResetEventSlim evReadyToRead,
          ManualResetEventSlim evStartReading)
{
     // Инициализация внутреннего буфера
     var  Messages = new List<string>();

     // Рабочий цикл чтения
     while(true)
     {
          // Сигнализирует о готовности
          evReadyToRead.Set();
          // Ждем сигнала от менеджера
          evStartReading.Wait();
          // Разрешено чтение по текущему индексу
          int k = ReadIndexCopy[iReader];
          Messages.Add(buffer[k]);
          // Сбрасываем сигнал о чтении
          evStartReading.Reset();
          // Проверяем статус завершения работы
          if (finish) break;
     }
}

// Код писателя практически идентичен коду читателя
void WriterThread(int iWriter,
     ManualResetEventSlim evReadyToWrite,
     ManualResetEventSlim evStartWriting)
```

```
     {
          // Инициализация массива сообщений писателя
          Messages = ..
          // Рабочий цикл записи
          while(true)
          {
               // Сигнализируем о готовности менеджеру
               evReadyToWrite.Set();
               // Ждем сигнала от менеджера
               evStartWriting.Wait();
               // Разрешена запись по текущему индексу
               k = WriteIndexCopy[iWriter];
               buffer[k] = Messages[j];
               // Проверяем статус завершения работы
               if (finish || j >= Messages.Length)
                    break;
               j++
          }
     }

// Код менеджера
void Manager(int nReaders, int nWriters)
{

     // Запуск читателей
     for(int i=0; i<nReaders; i++)
     {
          evReadyToRead[i] =
               new ManualResetEventSlim(false);
          evStartReading[i] =
               new ManualResetEventSlim(false);
          tReaders[i] = new Task( () =>
               Reader(i, evReadyToRead[i], evStartReading[i]));
          tReaders[i].Start();
     }
     // Запуск писателей
     for(int i=0; i < nWriters; i++)
     {
          var evReadyToWrite[i] =
               new ManualResetEventSlim(false);
          var evStartWriting[i] =
               new ManualResetEventSlim(false);
          tWriters[i] = new Task( () =>
               Writer(i, evReadyToWrite[i], evStartWriting[i]));
          tWriters[i].Start();
```

```
}

// Рабочий цикл
while(true)
{

     // Если в буфере есть свободные ячейки
     // пытаемся обработать готовых писателей
     if(!bFull)
     {
          // Получаем текущий индекс записи
          iW = GetBufferWriteIndex();
          if(iW != -1)
          {
               // Устанавливаем писателя,
               // которому разрешаем работать
               iWriter = GetWriter();
               if (iWriter != -1)
               {
                    // Сбрасываем сигнал готовности
                    // выбранного писателя
                    evReadyToWrite[iWriter].Reset();
                    // Сохраняем копию индекса для записи
                    ReadIndexCopy[iWriter] = iW;
                    // Разрешаем писателю начать работу
                    evStartWriting[iWriter].Set();
               }
          }
          else
               bFull = true;
     }
     // Если буфер не пуст, пытаемся
     // обработать готовых писателей
     if(!bEmpty)
     {
          // Получаем текущий индекс для чтения
          iR = GetBufferReadIndex();
          if(iR != -1)
          {
               //Устанавливаем готового читателя
               iReader = GetWriter();
               if (iReader != -1)
               {
                    evReadyToRead[iReader].Reset();
                    WriteIndexCopy[iReader] = iR;
                    evStartReading[iReader].Set();
```

```
                    }
                }
                else
                    bEmpty = false;

            }
        }
    }

    // Код функции получения номера готового писателя
    // с учетом приоритетов
        int GetWriter()
        {
            // Устанавливаем готовых писателей
            var ready = new List<int>();
            for(int i=0; i<nWriter; i++)
                if(evReadyToWrite[i].IsSet())
                    ready.Add(i);

            if(ready.Count == 0)
                return -1;

            return ready.OrderBy(i => WriterPriority[i]).First();
        }
```