Лабораторная работа №1

Сдать до 17.02

Тема: «Создание потоков».

Общее задание:

Задача. Написать программу для консольного процесса, который состоит из двух потоков: **main и worker.**

a) Поток **main** должен выполнить следующие действия:

1. Создать массив целых чисел, размерность и элементы которого вводятся с консоли(или сгенерировать случайно).
2. Ввести время для остановки и запуска потока `worker`.
3. Создать поток `worker`, передать в поток данные: размер массива, масив и т.д.
4. Приостановить поток **`worker(`**SuspendThread), затем через некоторое время снова запустить поток.
5. Уметь создавать поток командой _beginthreadex
6. Дождаться завершения потока `worker`.
7. Вывести на консоль результат работы потока `worker`
8. Завершить работу.

b) Глобальные переменные не использовать!

c) Обяснить: что такое идентификатор, дескриптор, работу функций.

Индивидуальные варианты:

d) Поток **worker** должен выполнить следующую работу:

1. Найти среднее арифметическое значение элементов массива. После каждого суммирования элементов «спать» 12 миллисекунд. Завершить свою работу.
2. Найти минимальный элемент массива.. Завершить свою работу.
3. Найти сумму элементов. После каждого суммирования элементов «спать» 20 миллисекунд. Завершить свою работу.
4. Найти максимальный элемент массива. Завершить свою работу.
5. Найти количество нулевых значений. После каждого действия элементов «спать» 12 миллисекунд. Завершить свою работу.
6. Ввести новый элемент X. Найти количество элементов массива, равных X.. Завершить свою работу.
7. Ввести новый элемент X.Найти количество элементов массива, больших X. Пред началом поиска «спать» 200 миллисекунд Завершить свою работу.
8. Ввести новый элемент X.Найти элементы массива, меньшие X.. Завершить свою работу.
9. Найти сумму квадратов элементов. После каждого суммирования элементов «спать» 200 миллисекунд. Завершить свою работу.
10. Вывести максимальный элемент из отрицательных элементов массива. После поиска «спать» 100 миллисекунд Завершить свою работу.
11. Вывести минимальный элемент из положительных элементов массива. Завершить свою работу.
12. Вывести количество четных элементов из элементов массива. Завершить свою работу.
13. Вывести элементы из отрезка [a,b]. Завершить свою работу.
14. Ввести количество нечетных элементов из элементов массива. Завершить свою работу.
15. Вывести количество элементов кратных 3 из элементов массива. Завершить свою работу.
16. Вывести количество элементов кратных 5 из элементов массива. Завершить свою работу.
17. Вывести количество элементов кратных 9 из элементов массива. Завершить свою работу.
18. Ввести новый элемент X - вещественный. Найти количество элементов массива, целая часть которых совпадает с целой частью X. Завершить свою работу.
19. Найти сумму квадратных корней элементов. После каждого суммирования элементов «спать» 200 миллисекунд. Завершить свою работу.

Примечания.

1. Для ожидания завершения работы потока `worker` использовать функцию:

```
DWORD WaitForSingleObject(
        HANDLE   hHandle,          // дескриптор объекта
        DWORD    dwMilliseconds    // интервал ожидания в миллисекундах
);
```

где второй параметр установить равным INFINITE. Например

```
WaitForSingleObject(hThread, INFINITE);      // ждать завершения потока
```

Здесь hThread – дескриптор потока `worker`.

2. Для засыпания использовать функцию:

```
VOID Sleep(
DWORD     dwMilliseconds              // миллисекунды
);
```

Например, Sleep(12); // спать 12 миллисекунд

Дополнительное (или штрафное после 20.02) задание:

a. Добавить третий поток **Count;**
b. Создать поток **Count** в потоке main, в подвешенном состоянии.
c. Запустить поток **Count**.

Поток **Count** выполняет:

Выводит на консоль числа фибоначчи, по возрастанию.

Лабораторная работа №2.

Тема: «Создание процессов».
Сдать до 3.03

Общее задание:

1. Два проекта (процессы) хранить в одном Solution (Решении) в VS Studio!
2. В Solution (Решениии) настроить, чтобы .exe файлы лежали в одном Debug!
3. Объяснить какие данные хранят структуры : STARTUPINFO, PROCESS_INFORMATION.
4. Написать программы двух консольных процессов Parent и Child, которые выполняют.

Процесс `Parent`:

- Согласно **индивидуальным вариантам** выполняет :

  - Ввести размер массива, ввести элементы массива;
  - Для вариантов 1,4, 6, 8, 9 – ввести необходимые дополнительные значения согласно варианту (A,B,X,K);
  - Формирует командную строку, которая содержит информацию об размерности массива, элементах и т.д. (согласно индивидуальному варианту);
  - Для консоли дочернего прооцесса устанавливает визуальные настройки, согласно индивидуальным вариантам:
    1. Установить любой цвет текста (не белый) для Child.
    2. Установить ширину буфера для Child.
    3. Установить высоту буфера для Child.
    4. Установить ширину (X) смещения от верхнего левого угла экрана.
    5. Установить высоту (Y) смещения от верхнего левого угла экрана.
    6. Установить любой цвет фона (не черный) для Child.
    7. Установить новый заголовок для окна Child.
    8. Установить любой цвет текста (не белый) для Child.
    9. Установить ширину буфера для Child.
    10. Установить высоту буфера для Child.
    11. Установить ширину (X) смещения от верхнего левого угла экрана.
    12. Установить высоту (Y) смещения от верхнего левого угла экрана.
    13. Установить любой цвет фона (не черный) для Child.
    14. Установить новый заголовок для окна Child.
    15. Установить любой цвет текста (не белый) для Child.
    16. Установить ширину (X) смещения от верхнего левого угла экрана.
    17. Установить высоту (Y) смещения от верхнего левого угла экрана.
    18. Установить любой цвет фона (не черный) для Child.
    19. Установить новый заголовок для окна Child.

  - Запускает дочерний процесс `Child`, которому через командную строку передается информация об размерности массива, элементах и т.д. (согласно варианту);

Процесс `Child`:

- Согласно **индивидуальным вариантам** `Child` выполняет:

  1. Выполнить суммирование элементов итогового массива до заданной позиции K. Полученный массив вывести. Тип элементов - вещественные числа.
  2. Найти в массиве повторяющиеся элементы (разместить их группы в массиве слева, остальные (одиночные) - соответственно справа). Полученный массив вывести. Тип элементов - вещественные числа.

3. Сортировка методом “пузырька”. Полученный массив вывести. Тип элементов - вещественные числа двойной точности
4. Поиск в массиве элементов из диапазона [A,B] (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Полученный массив вывести. Тип элементов - целые числа без знака.
5. Сортировка выбором. Полученный массив вывести. Тип элементов - символы.
6. Поиск в массиве элементов >A (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Полученный массив вывести. Тип элементов - целые числа.
7. Поиск в массиве простых чисел (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа без знака.
8. Поиск в массиве элементов =X (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - символы.
9. Выполнить произведение элементов (без 0) итогового массива до заданной позиции K. Полученный массив вывести. Тип элементов - целые числа, без знака.
10. Поиск в массиве лексем, (разделители – цифры). Полученные лексемы поместить в массиве слева, разделитель - пробел, остальные элементы - заполнить символом ‘0’. Полученный массив вывести. Тип элементов массива - символы.
11. Приведение массива к палиндрому (получившейся палиндром поместить в массиве слева, а лишние элементы соответственно – справа). Полученный массив вывести. Тип элементов - символы
12. Сортировка вставками. Полученный массив вывести. Тип элементов - целые числа.
13. Сортировка Шелла. Полученный массив вывести. Тип элементов - вещественные числа.
14. Поиск в массиве чисел кратных 3. (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа.
15. Поиск в массиве чисел кратных 5. (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа без знака.
16. Сортировка Хоара. Полученный массив вывести. Тип элементов - целые числа.
17. Сортировка Подсчетом. Полученный массив вывести. Тип элементов - целые числа.
18. Сортировка бинарная. Тип элементов - целые числа.
19. Поиск в массиве элементов >0 (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Полученный массив вывести. Тип элементов - целые числа.

Примечания.

Для ожидания завершения работы процесса `Child` использовать функцию:

```
DWORD WaitForSingleObject(
    HANDLE          hHandle,        // дескриптор объекта
    DWORD dwMilliseconds            // интервал ожидания в миллисекундах
);где второй параметр установить равным INFINITE, например

    WaitForSingleObject(hProcess, INFINITE);        // ждать завершения процесса
```

Здесь hProcess – дескриптор процесса `Child`.

В Solution (Решениии) настроить, что бы .exe файлы лежали в одном Debug!

Дополнительное (или штрафное после 6.03) задание:

1. завершить процесс с помощью функции TerminateProcess
2. завершить процесс Parent с помощью функции ExitProcess;
3. Запустить 2-й процесс **Count из Parent**. У процесса **Count** менять приоритет. Проецесс **Count** выводит на консоль числа фибоначчи, по возрастанию.

Лабораторная работа №3.

Тема: «Синхронизация потоков с помощью критических секций и событий».

Сдать до 17.03

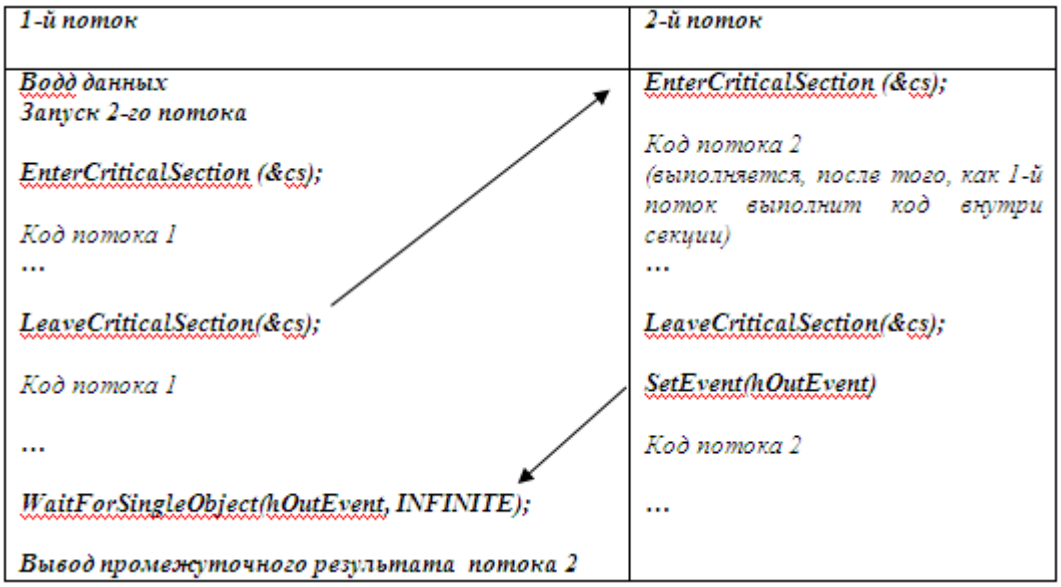
Написать программу для консольного процесса, который состоит из трёх потоков: **main , work, и третьего** (см. варианты)..

Общее задание:

Перед выполнением задания, нарисовать (**на бумаге**) примерную схему синхронизации потоков и показать преподавателю!

Пример для 2-х потоков синхронизации:

В двух потоках есть общая критическая секция. Второй поток ждет освобождения секции первым потоком, чтобы выполнить код в секции. Первый поток ждет события от второго потока, чтобы вывести промежуточный результат второго потока, схема для кода программы :



Индивидуальные варианты:

3.1 Объекты синхронизации :

Критическая секция синхронизирует работу потока ***work*** и потока ***main*** (для вывода массива в ***main***) ***;***

Критическая секция №2- синхронизация потока ***main*** и потока ***SumElement*** *(сигнализирует о начале суммирования);*

Событие - устанавливает поток ***SumElement для потока main*** (для вывода в ***main результата SumElement).***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив (**тип символы**), размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work(** в подвешенном состоянии);
5. запустить поток **SumElement(** в подвешенном состоянии);
6. ввести число k;
7. запустить поток **work**;
8. запустить поток **SumElement**;
9. Приостановить поток main на 1-2 милисекунды(Sleep)
10. Получить от потока **work** сигнал о начале суммирования(критическая секция);
11. Выводить на экран элементы массива (итогового);
12. известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов массива) (*использовать* ***критическую секцию***);
13. Выводить на экран элементы массива (итогового);
14. Дождаться сигнала потока **SumElement** (*использовать* ***событие***);
15. Вывести на экран результат работы потока **SumElement**

Поток **work** должен выполнить следующие действия:

- запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
- Поиск в массиве элементов, соответствующих цифрам (слева поместить в массив цифры, а остальные элементы массива - заполнить пробелами). Элементы - символы;
- Если цифр меньше k, то k уменьшить до количества цифр);
- после каждого готового элемента отдыхать в течение заданного интервала времени;
- известить поток **main** о начале суммирования (критическая секция).

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком* **main** - *использовать* ***критическую секцию, событие!*****):**

1. ждёт от потока **main** сообщения о начале суммирования (*использовать* ***критическую секцию***);
2. вычислить среднее арифметическое до позиции k;
3. сигнализировать потоку **main** о выводе результата (*использовать* ***событие***);

3.2 Объекты синхронизации :

Событие №1 устанавливает поток ***work для потока main*** (для вывода массива в ***main***) ***;***

Событие №2 устанавливает поток ***main*** для потока ***CountElement*** *(сигнализирует о начале суммирования);*

Критическая секция - синхронизация ***CountElement и*** *потока* ***main (***вывод ***результа CountElement).***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив(**тип символы**),, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. ввести число k;
5. запустить поток **work**;
6. запустить поток **CountElementм**;
7. Получить от потока **work** сигнал о начале суммирования (*использовать* ***событие***);
8. Выводить на экран элементы массива (итогового);
9. известить поток **CountElement** о начале суммирования (момент запуска произойдёт после того, будут выведены все элементы массива) (*использовать* ***событие***).
10. Выводить на экран элементы массива (итогового до позиции k);
11. Дождаться сигнала потока **CountElement** (*использовать* ***критическую секцию***);
12. Вывести на экран результат работы потока **CountElement**

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве элементов, соответствующих не цифрам и не символам латинского алфавита (слева поместить в массив , а остальные элементы массива - заполнить пробелами). Элементы - символы.
3. выводить на экран поэлементно элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **main** о начале суммирования(*использовать* ***событие***);

Поток **CountElement** должен выполнить следующие действия (*Для синхронизации с потоком* **main,** *использовать* ***критическую секцию, событие!*****):**

1. ждёт от потока main сообщения о начале суммирования (использовать событие);
2. выполнить подсчёт только символов соответствующих знакам препинания итогового массива;
3. сигнализировать потоку main о выводе результата (использовать критическую секцию);.

3.3 Объекты синхронизации :

Событие №1- устанавливает поток ***work для потока main*** (для вывода массива в ***main***) ***;***

Событие №2- устанавливает поток ***main*** для потока ***MultElement*** *(сигнализирует о начале выполненния вычислений* ***MultElement****);*

Критическая секция - синхронизация ***MultElement и*** *потока* ***main (***вывод ***результа MultElement).***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив(**тип unsigneg int**), размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work (** в подвешенном состоянии);
5. ввести число k;
6. запустить поток **work** ;
7. запустить поток **MultElement**;
8. Получить от потока **work** сигнал о начале умножения (*использовать* ***событие***);
9. Выводить на экран элементы массива (итогового);
10. известить поток **MultElement** о начале работы (момент запуска произойдёт после того, будут выведены на консоль k элементов массива), *использовать* ***событие.***
11. Вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве элементов четных элементов (разместить их в массиве слева, остальные элементы массива - справа). Элементы - целые числа без знака .
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. известить поток **main** о начале умножения (*использовать* ***событие***);

Поток **MultElement** должен выполнить следующие действия (*Для синхронизации с потоком* ***main*** *- использовать* ***событие*** **и критическую секцию**):

1. ждёт от потока **main** сигнал о начале умножения (*использовать* ***событие***);
2. выполнить произведение элементов итогового массива до заданной позиции k (0-вые элементы не перемножать);
3. известить поток **main** о выводе результата (*использовать* ***критическую секцию)*** .
4. вывести итоговое произведение

3.4 Объекты синхронизации :

Критическая секция - синхронизация work и потока CountElement (сигнализирует о начале запуска вычислений в CountElement);

Событие - устанавливает поток ***CountElement*** для потока ***main*** (для вывода в ***main*** результата *CountElement*)**.**

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив(**тип short**),, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work(** в подвешенном состоянии);
5. создать поток **CountElement(** в подвешенном состоянии);
6. Запросить число X.
7. Запустить поток **work**;
8. запустить поток **CountElement**;
9. Дождаться сигнала потока **CountElement** (*использовать* ***событие***);
10. Вывести на экран результат работы потока **CountElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве положительных элементов кратных 5 (разместить их в массиве слева, остальные элементы массива - справа).
3. выводить на экран поэлементно элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **CountElement** о начале работы (момент запуска произойдёт после того, будет сформирован итоговый массив (*использовать* ***критическую секцию)*** .

Поток **CountElement** должен выполнить следующие действия (*Для синхронизации с потоком work - использовать* ***критическую секцию, с*** потоком **main** (***событие!*)**:

1. ждёт от потока **work** сигнал о начале суммирования (*использовать* ***критическую секцию)*** ;
2. подсчитать количество элементов равных X;
3. известить(*использовать* ***событие)*** поток main о выводе результата

3.5 Объекты синхронизации :

Критическая секция - синхронизация work и потока CountElement (сигнализирует о начале запуска вычислений в CountElement);

Событие - устанавливает поток ***CountElement*** для потока ***main*** (для вывода в ***main*** результата *CountElement*)**.**

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив целых чисел, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work(** в подвешенном состоянии);
5. создать поток **CountElement(** в подвешенном состоянии);
6. Запросить число X.
7. Запустить поток **work**;
8. Запустить поток **CountElement**;
9. Дождаться сигнала потока **CountElement** (*использовать* ***событие***);
10. вывести на экран результат работы потока **CountElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве отрицательных элементов кратных 3 (разместить их в массиве слева, остальные элементы массива - справа). Элемент - X ввести в потоке main.
3. выводить на экран поэлементно элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **CountElement** о начале работы (момент запуска произойдёт после того, будет сформирован итоговый массив (*использовать* ***критическую секцию)*** .

Поток **CountElement** должен выполнить следующие действия (*Для синхронизации с потоком work - использовать* ***критическую секцию, с*** потоком **main** *-****событие!*)**:**

1. ждёт от потока **work** сигнал о начале суммирования (*использовать* ***критическую секцию)*** ;
2. подсчитать количество элементов равных X;
3. известить(*использовать* ***событие)*** поток main о выводе результата

-

3.6 Объекты синхронизации :

Критическая секция- синхронизация main и поток work (сигнализирует о начале запуска work, после воода K);

Событие1 (с ручным сбросом) – в потоке ***work*** устанавливает сигнал **для потока *main*** (для вывода массива) и для потока ***CountElement (*** *сигнализирует о начале вычислений)***.**

Событие2 - устанавливает поток ***CountElement*** для потока ***main (***вывод ***результа CountElement);***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив вещественных чисел, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work**;
5. запустить поток **CountElement**;
6. Запросить число K.
7. Сигнализировать Work о начале работы (*использовать **критическую секцию)***
8. Получить от потока **work** сигнал о выводе K элементов массива (*использовать **событие с ручным сбросом***);
9. Вывести на экран элементы массива до K
10. Дождаться сигнала потока **CountElement** (*использовать **событие2***);
11. Вывести на экран результат работы потока **CountElement**;
12. Вывести на экран оставшиеся элементы массива

Поток **work** должен выполнить следующие действия:

1. Ждать от Main сигнала о начале работы, после ввода K(*использовать **критическую секцию)***
2. Поиск в массиве элементов чисел >0 (разместить их в массиве слева, остальные элементы массива – справа)
3. Известить поток **main и CountElement** о начале суммирования (момент запуска произойдёт после того, будет будет сформировано K лементов итогового массива), *использовать **событие1(ручной сброс)***;

Поток **CountElement** должен выполнить следующие действия ((*Для синхронизации с потоком* **main** - *использовать обытие 2*, с потоком **work – *событие1*):**

1. ждёт от потока **work** сообщения о начале вычислений *(использовать **событие1***);
2. выполнить подсчёт целых элементов итогового массива из первых первых K элементов;
3. сигнализировать потоку **main** о выводе результата (*использовать **событие 2***);

3.7 Объекты синхронизации :

Критическая секция- синхронизация main и поток work (сигнализирует о начале запуска work, после вооода K);

Событие1 (с ручным сбросом) – в потоке ***work*** устанавливает сигнал **для потока *main*** (для вывода массива) и для потока ***CountElement (*** *сигнализирует о начале вычислений)***.**

Событие2 - устанавливает поток ***CountElement*** для потока ***main (***вывод ***результа CountElement);***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив(**тип символы**), размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work**;
5. запустить поток **CountElement**;
6. Запросить число K.
7. Сигнализировать Work о начале работы (*использовать **критическую секцию)***
8. Получить от потока **work** сигнал о выводе K элементов массива (*использовать **событие с ручным сбросом***);
9. Вывести на экран элементы массива до K
10. Дождаться сигнала потока **CountElement** (*использовать **событие2***);
11. Вывести на экран результат работы потока **CountElement**;
12. Вывести на экран осавшиеся элементы массива

Поток **work** должен выполнить следующие действия:

1. Ждать от Main сигнала о начале работы, после ввода K(*использовать **критическую секцию)***
2. Поиск в массиве элементов, соответствующих латинскому алфавиту (слева поместить в массив символы, а остальные элементы массива - заполнить пробелами). Элементы - символы;
3. известить поток **main и CountElement** о начале работы(момент запуска произойдёт после того, будет сформировано K лементов итогового массива), *использовать **событие1(ручной сброс)***;

Поток **CountElement** должен выполнить следующие действия ((*Для синхронизации с потоком* **main** - *использовать обытие 2*, с потоком **work – *событие1*):**

1. ждёт от потока **work** сообщения о начале вычислений *(использовать **событие1***);
2. вычислить количество гласных символов итогового массивав из первых K элементов;
3. сигнализировать потоку **main** о выводе результата (*использовать **событие 2***);

3.8 Объекты синхронизации :

Событие1 - устанавливает поток ***Work для потока main*** (для вывода итгового массива)**.**

Критическая секция- синхронизация main и потока MultElement (сигнализирует о начале запуска MultElement);

Событие2 - устанавливает поток ***MultElement для потока main*** (для вывода в ***main результата MultElement***)**.**

Поток** main **должен выполнить следующие действия:

1. Инициализировать необходимые события и критические секции.
2. создать массив(**тип long**), размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work**;
5. запустить поток **MultElement**;
6. Дождаться сигнала потока **work** (*использовать **событие***);
7. Выводить на экран элементы массива (итогового);
8. известить поток **MultElement** о начале умножения (момент запуска произойдёт после того, будут выведены на консоль элементы массива) (*использовать **критическую секцию***).
9. Дождаться сигнала потока **MultElement** (*использовать **событие***);
10. вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве квадратов простых чисел (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Элементы - целые числа без знака. Числа A,B ввести в потоке main.
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. выводить на экран поэлементно элементы массива (итогового);
5. известить поток **main** о начале работы (момент запуска произойдёт после того, будет массив (*использовать **событие).***

Поток **MultElement** должен выполнить следующие действия (*Для синхронизации с потоком **main** - использовать **событие и** **критическую секцию**!*):

1. ждёт от потока **main** сигнал о начале работы(*использовать **критическую секцию)*** ;
2. выполнить произведение элементов из диапазона [0, 10] из итогового массива
3. известить*)* поток **main** о выводе результата (*использовать **событие***);

3.9 Объекты синхронизации :

Событие1 - устанавливает поток ***Work для потока main*** (для вывода итгового массива)**.**
Критическая секция- синхронизация main и потока Count (сигнализирует о начале запуска Count);
Событие2 - устанавливает поток ***Count для потока main*** (для вывода в ***main результата Count***)**.**

Поток** main **должен выполнить следующие действия:

1. Инициализировать необходимые события и критические секции.
2. Создать массив(**тип __int16**), размерность и элементы которого вводятся пользователем с консоли;
3. Вывести размерность и элементы исходного массива на консоль;
4. Запустить поток **work**;
5. Запустить поток **Count**;
6. Дождаться сигнала потока **work** для вывода массива(*использовать **событие***);
7. Выводить на экран элементы массива (итогового);
8. Известить поток **Count** о начале умножения (момент запуска произойдёт после того, будут выведены на консоль элементы массива) (*использовать **критическую секцию***).
9. Дождаться сигнала потока **Count** (*использовать **событие***);
10. Вывести на экран результат работы потока **Count**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Замена положительных элементов на их корень квадратный (разместить их в массиве слева. Элементы - вещественные числа без знака. Числа A,B ввести в потоке main.
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. выводить на экран поэлементно элементы массива (итогового);
5. известить поток **main** о начале работы (момент запуска произойдёт после того, будет массив (*использовать **событие).***

Поток **Count** должен выполнить следующие действия (*Для синхронизации с потоком **main** - использовать **событие и** **критическую секцию**!*):

1. ждёт от потока **main** сигнал о начале работы(*использовать **критическую секцию)*** ;
2. найти количество целых чисел из итогового массива;
3. известить*)* поток **main** о выводе результата (*использовать **событие***);

3.10 Объекты синхронизации :

Событие №1- устанавливает поток ***work для потока main*** (для вывода части массива в ***main*) ;**
Событие №2- устанавливает поток ***main*** для потока *SumElement (сигнализирует о начале суммирования);*
Критическая секция №1- синхронизация ***SumElement для** потока **main*** (вывод в ***main SumElement);***

*Поток **main** должен выполнить следующие действия:*

1. создать массив(**тип символы**), размерность и элементы которого вводятся пользователем с консоли;
2. вывести размерность и элементы исходного массива на консоль;
3. Инициализировать необходимые события и критические секции.
4. ввести число k;
5. запустить поток **work**;
6. запустить поток **SumElement**;
7. Получить от потока **work** сигнал о начале суммирования (*использовать **событие***);
8. Выводить на экран элементы массива (до k);
9. известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов массива (*использовать **событие***);
10. Вывести на экран результат работы потока **SumElement**;
11. Получить от потока **work** сигнал (завершение потока) о выводе итогового массива (начиная с k);
12. Выводить на экран элементы итогового массива (начиная с k);

Поток **work** должен выполнить следующие действия:

- запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
- найти в массиве неповторяющиеся элементы (разместить их в массиве слева, остальные соответственно справа). Элементы - символы.
- после каждого готового элемента отдыхать в течение заданного интервала времени;
- известить поток **main** о начале суммирования (момент запуска произойдёт после того, будут сформированы k элементов массива (*использовать **событие***);

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком **main,** использовать **критическую секцию, событие)***

1. ждёт от потока **main** сигнал о начале суммирования(*использовать **событие***);
2. посчитать количество цифр массива до заданной позиции k;
3. известить(*использовать **критическую секцию)*** поток main о выводе результата

3.11 Объекты синхронизации :

Критическая секция- синхронизация work и потока MultElement (сигнализирует о начале запуска MultElement);

Событие - устанавливает поток ***MultElement и для*** *потока* ***main*** (вывод в ***main***

*Поток **main** должен выполнить следующие действия:*

1. Инициализировать необходимые события и критические секции.
2. создать массив, элементы - вещественные числа, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work**;
5. запустить поток **MultElement**;
6. вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Найти в массиве повторяющиеся элементы (разместить их группы в массиве слева, остальные соответственно справа). Элементы - вещественные числа.
3. выводить на экран поэлементно элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **MultElement** о начале произведения (момент запуска произойдёт после того, будет сформирован итоговый массив(*использовать **критическую секцию)***.

Поток **MultElement** должен выполнить следующие действия (*Для синхронизации с потоком work - использовать **критическую секцию, с*** потоком **main** - ***событие!*)**:

1. ждёт от потока **work** сигнал о начале произведения (*использовать **критическую секцию)***;
2. выполнить произведение элементов итогового массива;
3. известить(*использовать **событие)*** поток main о выводе результата

3.12 Объекты синхронизации :

Критическая секция №1- синхронизация потока ***work и*** вывод в потоке ***main*** (для вывода части массива в ***main*)*;***

Событие устанавливает поток ***main*** для потока ***SumElement*** *(сигнализирует о начале суммирования).*

Критическая секция №2- синхронизация ***SumElement и*** *потока* ***main (***вывод ***результа SumElement);***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив, элементы - вещественные числа двойной точности, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;

4. запустить поток **work (** в подвешенном состоянии);
5. ввести число k;
6. запустить поток **work**
7. запустить поток **SumElement**;
8. Приостановить работу потока на 3 мс (Sleep)
9. Получить от потока **work** сигнал о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов) (*использовать **критическую секцию)*;
10. вывести на экран элементы массива (итогового до k элементов);
11. известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов) (*использовать **событие***);.
12. вывести на экран результат работы потока **SumElement**;
13. вывести на экран элементы массива (итогового после k элементов);

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Сортировка методом “пузырька”. Элементы - вещественные числа двойной точности.
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. известить поток **main** о начале суммирования (момент запуска произойдёт после того, будут отсортированы k элементов) (*использовать **критическую секцию)*;

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком main**, использовать **событие*** **и критическую секцию*!***):

1. ждёт от потока **main** сигнал о начале суммирования (*использовать **событие***);
2. выполнить суммирование элементов итогового массива до заданной позиции k;
3. вывести итоговую сумму.
4. известить(*использовать **критическую секцию)*** поток **main** о выводе результата

3.13 Объекты синхронизации :

Критическая секция- синхронизация work и потока MultElement (сигнализирует о начале запуска MultElement);

Событие - устанавливает поток ***MultElement для потока main*** (для вывода в ***main результата MultElement*).**

Поток** main **должен выполнить следующие действия:

1. Инициализировать необходимые события и критические секции.
2. создать массив, элементы - целые числа без знака, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work**;
5. запустить поток **MultElement**;
6. вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве элементов из диапазона [A,B] (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Элементы - целые числа без знака. Числа A,B ввести в потоке main.
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. выводить на экран поэлементно элементы массива (итогового);
5. известить поток **MultElement** о начале работы (момент запуска произойдёт после того, будет сформирована часть итогового массива (когда будут найдены все элементы из диапазона [A, B]) (*использовать **критическую секцию).***

Поток **MultElement** должен выполнить следующие действия (*Для синхронизации с потоком **main** - использовать **событие, с*** потоком **work - критическую секцию***!*):

1. ждёт от потока **work** сигнал о начале работы(*использовать **критическую секцию)*** ;
2. выполнить произведение элементов из диапазона [A, B] в итоговом массиве
3. известить*)* поток main о выводе результата (*использовать **событие***);

3.14 Объекты синхронизации :

Событие№1 - устанавливает поток ***work для потока main*** (для вывода части массива в ***main*) ;**

Событие №2- устанавливает поток ***main*** для потока *SumElement (сигнализирует о начале суммирования).*

Критическая секция - синхронизация ***SumElement и*** *потока* ***main (***вывод ***результа SumElement);***

Поток** main **должен выполнить следующие действия:

1. Инициализировать необходимые события и критические секции.
2. создать массив, **элементы - символы,** размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. ввести число k;
5. запустить поток **work**;

6. запустить поток **SumElement**;
7. Получить от потока **work** сигнал о начале суммирования (*использовать **событие***);
8. Выводить на экран элементы массива (итогового *до позиции k*);
9. известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов массива), *использовать **событие***;
10. вывести на экран результат работы потока **SumElement**;
11. Выводить на экран элементы массива (итогового *с позиции k*);

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Сортировка выбором. Элементы - символы.
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. известить поток **main** о начале суммирования, момент запуска произойдёт после того, будут отсортированы на консоль k элементов массива (*использовать **событие***);

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком work**, использовать критическую секцию, событие!***):**

1. ждёт от потока **main** сигнал о начале суммирования (*использовать **событие***);
2. вычичлить среднее арифметическое (кодов символов) итогового массива до заданной позиции k;
   или посчитать количество цифр массива до заданной позиции k;
3. известить(*использовать **критическую секцию)*** поток **main** о выводе результата

3.15 Объекты синхронизации :

Событие (с ручным сбросом) – в потоке ***work*** устанавливает сигнал ***для потока main*** (для вывода массива) и для потока ***SumElement (*** *сигнализирует о начале суммирования)***.**

Событие2 (с автоматическим сбросом) – в потоке ***work*** устанавливает сигнал ***для потока main*** (для вывода массива)**.**

Критическая секция- синхронизация ***SumElement и*** *потока* ***main (***вывод ***результа SumElement);***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив, **элементы - целые числа без знака,** размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. Запросить числа A, K1, K2 (0<K1<K2<размерности массива).
5. запустить поток **work**;
6. запустить поток **SumElement**;
7. получить от потока **work** сообщение о начале работы потока **SumElemen** (*использовать **событие с ручным сбросом).***
8. выводить элементы массива (до K2);
9. получить от потока **SumElemen** сигнал (*использовать **критическую секцию***) о выводе результата и вывести на экран результат работы потока **SumElement**;
10. получить сигнал о выводе остатка массива (*дождаться завершения потока* **work** ***).***
11. выводить элементы массива (с K2);

Поток **work** должен выполнить следующие действия:

- запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
- Поиск в массиве элементов >A (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Элементы - целые числа без знака. Число A ввести в потоке main.
- после каждого готового элемента отдыхать в течение заданного интервала времени;
- известить поток **SumElement** и поток **main** о начале суммирования (момент запуска произойдёт после того, будет сформировано K2 элементов итогового массива (*использовать **событие с ручным сбросом)***

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком* **main** - *использовать **критическую секцию, с*** потоком ***work - событие!***):

1. ждёт от потока **work** сигнал о начале суммирования (*использовать **событие с ручным сбросом***);
2. выполнить суммирование элементов от позиции K1 до позиции K2 итогового массива;
3. известить(*использовать **критическую секцию)*** поток **main** о выводе результата

3.16 Объекты синхронизации :

Критическая секция №1- синхронизация ***work и*** вывод в ***main***(для вывода части массива в ***main***) ***;***

Событие устанавливает поток ***main*** для потока *SumElement (сигнализирует о начале суммирования).*

Критическая секция №2- синхронизация ***SumElement и*** *потока* ***main (***вывод ***результа SumElement);***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.

2. создать массив, **элементы - целые числа без знака,** размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work (** в подвешенном состоянии);
5. ввести число k;
6. запустить поток **work**;
7. запустить поток **SumElement**;
8. Приостановить работу потока на 3 мс (Sleep)
9. ждёт от потока **work** сигнал о начале суммирования (*использовать **критическую секцию 1)*** ;
10. Выводить на экран элементы массива (итогового);
11. известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут готовы к печати k - элементов массива), (*использовать **событие)***.
12. ждёт от потока **SumElement** сигнал о выводе результата (*использовать **критическую секцию 2)*** ;
13. вывести на экран результат работы потока **SumElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве простых чисел (разместить их в массиве слева, остальные элементы массива - справа). Элементы - целые числа без знака.
3. известить поток **main** о начале суммирования (момент запуска произойдёт после того, будут готовы к печати k - элементов массива), (*использовать **критическую секцию1)***.
4. после каждого готового элемента отдыхать в течение заданного интервала времени;

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком main**, использовать событие и критическую секцию**!*):

1. ждёт от потока **main** сигнал о начале суммирования (*использовать **событие***);
2. выполнить суммирование элементов итогового массива до заданной позиции k;
3. известить (*использовать **критическую секцию2)*** поток **main** о выводе результата

3.17 Объекты синхронизации :

Критическая секция- синхронизация work и потока CountElement (сигнализирует о начале запуска вычислений в CountElement);

Событие - устанавливает поток ***CountElement*** для потока ***main*** (для вывода в ***main*** результата *CountElement*).

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив, **элементы - символы,** размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work(** *в подвешенном состоянии*);
5. создать поток **CountElement(** в *подвешенном состоянии*);
6. запросить символ X.
7. запустить поток**Work**;
8. запустить поток **CountElement**;
9. вывести на экран результат работы потока **CountElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. поиск в массиве элементов =X (разместить их в массиве слева, остальные элементы массива - справа). Элементы - символы. X ввести в потоке main.
3. выводить на экран поэлементно элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **CountElement** о начале работы (момент запуска произойдёт после того, будет сформирован итоговый массив (*использовать **критическую секцию)*** .

Поток **CountElement** должен выполнить следующие действия (*Для синхронизации с потоком **work** - использовать **критическую секцию, с*** потоком **main** - ***событие!*)**:

1. ждёт от потока **work** сигнал о начале суммирования (*использовать **критическую секцию)*** ;
2. подсчитать количество элементов равных X;
3. известить(*использовать **событие)*** поток main о выводе результата

3.18 Объекты синхронизации :

Событие №1 устанавливает поток ***work для потока main*** (для вывода в ***main*).**

Событие №2- устанавливает поток ***main*** для потока ***MultElement*** *(сигнализирует о начале выполненния вычислений **MultElement**);*

Критическая секция - синхронизация ***MultElement и*** *потока* ***main (***вывод ***результата MultElement);***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.

2. создать массив, элементы - вещественные числа, размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work (** в подвешенном состоянии);
5. ввести числа k,A;
6. запустить поток **work** ;
7. запустить поток **MultElement**;
8. Получить от потока **work** сигнал о начале умножения (*использовать **событие***);
9. Выводить на экран элементы массива (итогового);
10. известить поток **MultElement** о начале работы (момент запуска произойдёт после того, будут выведены на консоль k элементов массива), *использовать **событие.***
11. Вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве элементов <A (разместить их в массиве справа, остальные элементы массива - слева). Элементы - вещественные числа.
3. Выводить на экран элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **main** о начале умножения (*использовать **событие***);

Поток **MultElement** должен выполнить следующие действия (*Для синхронизации с потоком main**, использовать **событие** **и критическую секцию**!*):

1. ждёт от потока **main** сигнал о начале умножения (*использовать **событие***);
2. выполнить произведение элементов итогового массива с заданной позиции k;
3. известить поток **main** о выводе результата (*использовать **критическую секцию)*** .
4. вывести итоговое произведение

3.19 Объекты синхронизации :

Событие (с ручным сбросом) – в потоке ***work*** устанавливает сигнал **для потока *main*** (для вывода массива) и для потока ***CountElement (*** *сигнализирует о начале вычислений)***.**

Критическая секция - синхронизация ***CountElement и*** *потока* ***main (***вывод ***результа CountElement);***

Поток *main* ***должен выполнить следующие действия:***

1. Инициализировать необходимые события и критические секции.
2. создать массив, **элементы - символы,** размерность и элементы которого вводятся пользователем с консоли;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток work;
5. запустить поток CountElement;
6. Получить от потока work сигнал о выводе массива (использовать событие с ручным сбросом);
7. Вывести на экран результат работы потока CountElement;

Поток **work** должен выполнить следующие действия:

1. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
2. Поиск в массиве лексем, начинающихся с цифры (разделители – пробел и тире). Полученные лексемы поместить в массиве слева, а лишние элементы -заполнить символом подчеркивания: «_»). Элементы - символы.
3. известить поток **main и CountElement** о начале суммирования (момент запуска произойдёт после того, будет сформирован итоговый массив), *использовать **событие(ручной сброс)***;

Поток **CountElement** должен выполнить следующие действия ((*Для синхронизации с потоком* **main** - *использовать **критическую секцию,*** с потоком **work** - ***событие!*****):**

1. ждёт от потока **work** сообщения о начале вычислений *(использовать **событие***);
2. выполнить подсчёт элементов (до символов подчеркивания: «_») итогового массива,;
3. сигнализировать потоку **main** о выводе результата (*использовать **критическую секцию***);

Лабораторная работа №4.
Сдать до 14.04
Тема: «Синхронизация процессов при помощи событий, мьютексов и семафоров».

Общее задание:

1. В другой процесс можно передать количество сообщений через файл.

*При реализации **синхронизации** процессов использовать функции ожидания сигнального состояния объекта только с **равным нулю или бесконечности интервалом** ожидания. Каждый отдельный процесс открывать в **отдельном консольном окне.** Использовать функцию* WaitForMultipleObject *для ожидания одного из группы событий.*
ПЕРЕДАЧА СООБЩЕНИЙ : Отправить сообщение, например, А или В от одного процесса другому, в данном задании означает : создаем события соответсвующие сообщениям А и В. Затем вводится одно из сообщений (А или В) с консоли в одном процессе и устанавливается соответсвующее событие в сигнальное состояние. В другом процессе ожидается одно из событии и выводится на консоль соответствующее сообщение..
АКТИВНЫЙ процесс**- процесс, который может отправить сообщение, введённое с консоли и получить сообщение.**

Индивидуальные варианты:

4.1. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события(`с автоматическим сбросом`), которые обозначают сообщения: "A", "B", "A2" , "B2" и конец сеанса для процессов* **Reader и Writer**.
Одновременно отправлять сообщения могут **только ОДИН АКТИВНЫЙ процесс Writer (использовать мьютекс)** и принимать и отправлять **ДВА АКТИВНЫХ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания);

Процесс **Administrator**:
1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Reader и Writer**, которые он должен запустить;
3. запрашивает у пользователя кол-во отправленных сообщений для процесса **Writer** и кол-во принятых сообщений для процесса **Reader(общее количество отправленных Writer и принятых Reader сообщений должно совпадать)**;
4. запускает заданное количество процессов **Reader и Writer**;
5. принимает от каждого процесса **Reader** сообщение *"A2" или "B2"* и выводит его на консоль в одной строке.
6. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
7. завершает свою работу.

Процесс **Writer**:
1. синхронизировать работу процессов **Writer** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения(*"A" или "B")* , и передает их (по одному) процессу **Reader**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:
1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщение от процесса **Writer**;
4. выводит на консоль сообщение *"A" или "B"* ;
5. передает сообщение*"A2" или "B2"* от **Writer** - процессу **Administrator**;
6. передает сообщение о завершении сеанса процессу **Administrator**;
7. завершает свою работу.

4.2. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события(`с автоматическим сбросом`), которые обозначают сообщение "A" , сообщение "B", и конец сеанса для процессов* **Reader и Writer**.
Одновременно принимать и отправлять сообщения могут **только ТРИ АКТИВНЫХ процесса Writer(использовать семафор)**, и **ОДИН АКТИВНЫЙ процесс Reader(использовать мьютекс)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).
Процесс **Administrator**:
1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Reader и Writer**, которые он должен запустить;
3. запрашивает у пользователя кол-во отправленных сообщений для процесса **Writer** и кол-во принятых сообщений для процесса **Reader(общее количество отправленных и принятых сообщений должно совпадать)**;

4. запускает заданное количество процессов **Reader и Writer.** Одновременно принимать и отправлять сообщения могут **только три процесса Writer(использовать семафор)**, и **один процесс Reader(использовать мьютекс)**, передача остальных сообщений от других процессов должна блокироваться(находиться в режиме ожидания);;
5. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
6. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, и передает их (по одному) процессу **Reader**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения от процесса **Writer**;
4. выводит на консоль сообщение;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.3. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом)***, которые обозначают сообщение "A"(два события) , сообщение "B" (два события), сообщение "C"(два события и конец сеанса для процессов* **Reader и Writer**.

Одновременно принимать и отправлять сообщения могут **только ОДИН АКТИВНЫЙ процесс Writer(использовать мьютекс)** и **ОДИН АКТИВНЫЙ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Writer(Reader)**;
3. запрашивает у пользователя кол-во отправленных сообщений процессом **Writer(**и принятых процессом **Reader)**;
4. запускает заданное количество процессов **Reader и Writer**;
5. принимает от каждого процесса **Writer** сообщение и выводит на консоль, затем отправляет его процессу **Reader.**
6. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
7. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее (*"A" или "B" или "C")* и передает их (по одному) процессу **Administrator**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения от процесса **Administrator**;
4. выводит на консоль сообщение;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.4. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. Для моделирования передачи сообщений ввести специальные события(**с ручным сбросом для Reader**), которые обозначают сообщение "A" , сообщение "B", и автоматическое событие - конец сеанса для процессов **Reader и Writer**.

Одновременно принимать и отправлять сообщения могут **только ОДИН АКТИВНЫЙ процесс Writer(использовать мьютекс)**, и **ДВА АКТИВНЫХ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя **k**-количество процессов **Writer (** количество процессов **Reader =2*k)**, которые он должен запустить;
3. запрашивает у пользователя кол-во отправленных и принятых сообщений для процессов **Writer** и **Reader**

4. запускает заданное количество процессов **Reader и Writer**
5. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
6. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью событий с ручным сбросом
3. апрашивает с консоли сообщения, и передает их (по одному) процессам **Reader**;
4. передает сообщение (с автоматическим сбросом) о завершении сеанса процессу Administrator;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения (**с ручным сбросом)** от процесса **Writer**;
4. выводит на консоль сообщения;
5. передает сообщение (с автоматическим сбросом) о завершении сеанса процессу **Administrator**;

завершает свою работу.

4.5. Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом)***, которые обозначают «A» , «B» и конец сеанса для процессов* **Parent и Child.**

Принимать сообщение можно **только от ОДНОГО АКТИВНОГО процесса Child(использовать мьютекс) и ОДНОГО АКТИВНОГО процесса Parent(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить;
3. запрашивает кол-во сообщений, отправленных **Parent и** кол-во сообщений отправленных **Child;**
4. запускает заданное количество процессов **Parent, Child**;
5. принимает от каждого процесса **Parent, Child** сообщения, выводит сообщения и кто его отправил на консоль в одной строке.
6. завершает свою работу.

Процесс **Parent**:

1. синхронизировать работу процессов **Parent** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. передаёт сообщения **Boss** только один активный процесс, передача остальных сообщений от других процессов должна временно блокироваться с помощью **семафора**;
4. запрашивает с консоли сообщения, состоящее из *«A»* и передает их (по одному) процессу Boss;
5. завершает свою работу.
6. Принимает от процесса **Boss о** завершении работы

Процесс **Child**:

1. синхронизировать работу процессов **Child** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. передаёт сообщения **Boss** только один активный процесс, передача остальных сообщений от других процессов должна временно блокироваться с помощью **мьютекса**;
4. запрашивает с консоли сообщения, состоящее из *«B» »* и передает их (по одному) процессу Boss;
5. завершает свою работу.
6. Принимает от процесса **Boss о** завершении работы

4.6. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение «A» , «B», «C» , «D» и конец сеанса для процессов* **Reader и Writer**. *Для сообщений «C» и «D»* использовать **события с ручным сбросом.**

Одновременно принимать и отправлять сообщения могут **только два АКТИВНЫХ процесса Writer(использовать мьютексы)** и **два АКТИВНЫХ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Writer(Reader)**;
3. запрашивает у пользователя кол-во отправленных сообщений для процессов **Writer и** кол-во полученных сообщений **Reader (общее количество отправленных и принятых сообщений должно совпадать)**;
4. запускает заданное количество процессов **Reader и Writer**;
5. принимает от каждого процесса **Writer** сообщения и выводит на консоль, затем отправляет их процессам **Reader.**

6. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
7. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютексов**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее из *"A" или "B",* и передает их (по одному) процессу **Administrator**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения *«C» , «D»* от процесса **Administrator**;
4. выводит на консоль сообщения;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.7. Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом)***, которые обозначают «A» , «B», «C» , «D» и конец сеанса для процессов* **Parent и Child.**

Принимать сообщение можно **только от двух АКТИВНЫХ процессов Child(использовать семафор) и одного АКТИВНОГО процесса Parent(использовать мьютекс)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить;
3. запрашивает кол-во сообщений, принятых от **Parent или Child**
4. запускает заданное количество процессов **Parent, Child**;
5. принимает от каждого процесса **Parent, Child** сообщения, выводит и кто его отправил на консоль в одной строке.
6. завершает свою работу.

Процесс **Parent**:

1. синхронизировать работу процессов **Parent** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее *«A» или «B»* и передает их (по одному) процессу Boss;
4. завершает свою работу.

Процесс **Child**:

1. синхронизировать работу процессов **Child** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее *«C» или «D»* и передает их (по одному) процессу Boss;
4. завершает свою работу.

4.8. Написать программы для консольного процесса Boss и консольных процессов Parent, Child. ***Для моделирования передачи сообщений ввести специальные 5 событий(*** **с автоматическим сбросом)***, которые обозначают «A» , «B», «C» , «D»,, и конец сеанса для процессов* **Parent и Child.**

Отправить сообщение можно **только пяти АКТИВНЫМ процессам из всех процессов Child и Parent (использовать семафор)**, отправка и передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания). Больше 4-х процессов **Child не создавать!**

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Parent** и количество(<=4) процессов **Child**, которые он должен запустить.
3. запрашивает кол-во сообщений, отправленных (полученных) **Parent и Child**
4. запускает заданное количество процессов **Parent, Child**;
5. запрашивает с консоли (можно автоматически получив сообщ. A - отправить сообщение C, получив сообщ. B - отправить D) и отправляет сообщение для процессов **Child**
6. принимает от процессов **Parent** сообщения, выводит на консоль в одной строке.
7. принимает от процессов Child и Parent сообщение о завершении сеанса процесса
8. завершает свою работу.

Процесс **Parent**:

1. синхронизировать работу процессов **Parent и Child** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее *«A» или «B»* и передает их (по одному) процессу Boss;

4. передает сообщение о завершении сеанса процессу **Boss**
5. завершает свою работу.

Процесс **Child**:

1. синхронизировать работу процессов **Parent и Child** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. получает сообщения, состоящее *«C» или «D»* от процесса Boss и выводит его на консоль;

a. передает сообщение о завершении сеанса процессу **Boss**

4. завершает свою работу.

4.9. Написать программы для консольного процесса **Boss** и консольных процессов **Employee**. *Для моделирования передачи сообщений ввести специальные события (**с ручным сбросом**), которые «0» , «1», «2», «3» , «4» и конец сеанса для процессов* **Employee** .

Посылать сообщение можно **только трём АКТИВНЫМ процессам Employee (использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Employee**, которые он должен запустить;
3. запрашивает у пользователя количество сообщений для процессов **Employee**, которые он должен отправить;
4. запускает заданное количество процессов **Employee**;
5. запрашивает с консоли, сообщение состоящее из *«0» , «1», «2», «3»*, конец сеанса работы и передает (по одному) его процессу **Employee** и выводит его на консоль в одной строке.
6. выводит его на консоль сообщение об окончании работы очередного процесса **Employee**.
7. завершает свою работу.

Процесс **Employee**:

1. синхронизировать работу процессов **Employee** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения от процесса **Boss**;
4. передаёт сообщение о завершении работы процессу **Boss**
5. завершает свою работу.

4.10 Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события (***с ручным сбросом)***, которые обозначают сообщение "A" , сообщение "B", и события(автоматический сброс) - конец сеанса для процессов* **Reader и Writer**.

Одновременно принимать и отправлять сообщения могут **только два АКТИВНЫХ процесса Writer(использовать мьютексы)** и **два АКТИВНЫХ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Reader и Writer**, которые он должен запустить;
3. запрашивает у пользователя кол-во отправленных сообщений для процесса **Writer**. Кол-во принятых сообщений для процесса **Reader** вычислить **(общее количество отправленных и принятых сообщений должно совпадать)**;
4. запускает заданное количество процессов **Reader и Writer**;
5. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
6. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютексов**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщение, и передает их (по одному) процессу **Reader**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения от процесса **Writer**;
4. выводит на консоль сообщения;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.11. Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для моделирования передачи сообщений ввести специальные событи я (***с автоматическим сбросом)** *, которые обозначают любые 4-е цифры и конец сеанса для процессов* **Parent и Child (**сообщения для **Parent и Child** *должны быть разные***).**

Принимать и отправлять сообщение может **только один АКТИВНЫЙ процесс Parent (использовать мьютекс) и три АКТИВНЫХ процесса Child (использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить;
3. запрашивает кол-во сообщений, отправленных каждому **Parent и** каждому **Child(общее количество отправленных** процессом **Parent и принятых** процессом **Child сообщений должно совпадать)**
4. запускает заданное количество процессов **Parent, Child**;
5. запрашивает с консоли сообщение, отправляет сообщение процессу **Parent** и выводит сообщение.
6. завершает свою работу.

Процесс **Parent**:

1. синхронизировать работу процессов **Parent** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. получает сообщения, от процесса Boss и выводит его на консоль;
4. запрашивает с консоли сообщение, отправляет сообщение процессам **Child**
5. завершает свою работу.

Процесс **Child**:

1. синхронизировать работу процессов **Child** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. получает сообщения, от процесса **Parent** и выводит его на консоль;
4. завершает свою работу.

4.12. Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для моделирования передачи сообщений ввести специальные события(`с автоматическим сбросом`), которые обозначают любые 4-е цифры и конец сеанса для процессов* **Parent и Child.**

Принимать и отправлятьс ообщение может **только два АКТИВНЫХ процесса Parent (использовать семафор) и один АКТИВНЫЙ процесс Child (использовать мьютекс)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить;
3. запрашивает кол-во сообщений, отправленных **Parent (и Child**);
4. запускает заданное количество процессов **Parent, Child**;
5. отправляет сообщение процессу **Parent** и выводит сообщение и кто его отправил.
6. завершает свою работу.

Процесс **Parent**:

1. синхронизировать работу процессов **Parent** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. получает сообщения, от процесса Boss и выводит на консоль;
4. отправляет любое другое сообщение (не такое какое получил от **Boss)** процессу **Child**
5. завершает свою работу.

Процесс **Child**:

1. синхронизировать работу процессов **Child** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. получает сообщения, от процесса **Parent** и выводит на консоль;
4. завершает свою работу.

4.13. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события(`с автоматическим сбросом`), которые обозначают сообщение «A» , «B», «C» , «D» и конец сеанса для процессов* **Reader и Writer**. *Для сообщений «C» и «D»* **.**

Одновременно принимать и отправлять сообщения могут **только один АКТИВНЫЙ процесс Writer(использовать мьютекс)** и **два АКТИВНЫХ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Writer(Reader)**;
3. запрашивает у пользователя кол-во отправленных сообщений для процессов **Writer и** кол-во полученных сообщений **Reader (общее количество отправленных и принятых сообщений должно совпадать)**;
4. запускает заданное количество процессов **Reader и Writer**;
5. принимает от каждого процесса **Writer** сообщения и выводит на консоль, затем отправляет их процессам **Reader.**

6. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
7. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее из *“A” или “B”,* и передает их (по одному) процессу **Administrator**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения *«C» , «D»* от процесса **Administrator**;
4. выводит на консоль сообщения;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.14. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом***), которые обозначают сообщение “A” , сообщение “B”, , сообщение “C” и конец сеанса для процессов* **Reader и Writer**.
Одновременно принимать и отправлять сообщения могут **только ДВА АКТИВНЫХ процесса Writer(использовать семафор)**, и **ОДИН АКТИВНЫЙ процесс Reader(использовать мьютекс)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Reader и Writer**, которые он должен запустить;
3. запрашивает у пользователя кол-во отправленных сообщений для процесса **Writer** и кол-во принятых сообщений для процесса **Reader(общее количество отправленных и принятых сообщений должно совпадать)**;
4. запускает заданное количество процессов **Reader и Writer.** Одновременно принимать и отправлять сообщения могут **только три процесса Writer(использовать семафор)**, и **один процесс Reader(использовать мьютекс)**, передача остальных сообщений от других процессов должна блокироваться(находиться в режиме ожидания);;
5. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
6. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, и передает их (по одному) процессу **Reader**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения от процесса **Writer**;
4. выводит на консоль сообщение;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.15. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом***), которые обозначают сообщение “A”, сообщение “B”, сообщение“C”, сообщение “D” и конец сеанса для процессов* **Reader и Writer**.

Одновременно принимать и отправлять сообщения могут **только ОДИН АКТИВНЫЙ процесс Writer(использовать мьютекс)** и **ОДИН АКТИВНЫЙ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Writer(Reader)**;
3. запрашивает у пользователя кол-во отправленных сообщений процессом **Writer(**и принятых процессом **Reader)**;

4. запускает заданное количество процессов **Reader и Writer**;
5. принимает от каждого процесса **Writer** сообщение и выводит на консоль, затем соответвующее (например, *"A"* соответствует *"C"*) сообщение процессу **Reader.**
6. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
7. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее (*"A" или "B")* и передает их (по одному) процессу **Administrator**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения (*"C" или "D")* от процесса **Administrator**;
4. выводит на консоль сообщение;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.16. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. Для моделирования передачи сообщений ввести специальные события которые обозначают сообщение "A" , сообщение "B", сообщение "C"(**с ручным сбросом**), и автоматическое событие - конец сеанса для процессов **Reader и Writer**.

Одновременно принимать и отправлять сообщения могут **только ОДИН АКТИВНЫЙ процесс Writer(использовать мьютекс)**, и **ДВА АКТИВНЫХ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя **k**-количество процессов **Writer (** количество процессов **Reader =2*k)**, которые он должен запустить;
3. запрашивает у пользователя кол-во отправленных и принятых сообщений для процессов **Writer** и **Reader**
4. запускает заданное количество процессов **Reader и Writer**
5. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
6. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. апрашивает с консоли сообщения, и передает их (по одному) процессам **Reader**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения от процесса **Writer**;
4. выводит на консоль сообщения;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.17. Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом)***, которые обозначают «A» , «B» и конец сеанса для процессов* **Parent и Child.**

Принимать сообщение можно **только от ОДНОГО АКТИВНОГО процесса Child(использовать мьютекс) и ДВУХ АКТИВНЫХ процесса Parent(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить;
3. запрашивает кол-во сообщений, отправленных **Parent и** кол-во сообщений отправленных **Child (общее количество отправленных и принятых сообщений должно совпадать);**
4. запускает заданное количество процессов **Parent, Child**;
5. принимает от каждого процесса **Parent, Child** сообщения, выводит сообщения и кто его отправил на консоль в одной строке.
6. завершает свою работу.

Процесс **Parent**:

1. синхронизировать работу процессов **Parent** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. передаёт сообщения **Boss** только один активный процесс, передача остальных сообщений от других процессов должна временно блокироваться с помощью **семафора**;
4. запрашивает с консоли сообщения, состоящее из *«A»* и передает их (по одному) процессу Boss;
5. завершает свою работу.
6. Принимает от процесса **Boss о** завершении работы

Процесс **Child**:

1. синхронизировать работу процессов **Child** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. передаёт сообщения **Boss** только один активный процесс, передача остальных сообщений от других процессов должна временно блокироваться с помощью **мьютекса**;
4. запрашивает с консоли сообщения, состоящее из *«B» »* и передает их (по одному) процессу Boss;
5. завершает свою работу.
6. Принимает от процесса **Boss о** завершении работы

4.18. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события, которые обозначают сообщение «A» , «B», «C» , «D» и конец сеанса для процессов* **Reader и Writer**. *Для сообщений «C» и «D»* использовать **события с ручным сбросом.**

Одновременно принимать и отправлять сообщения могут **только два АКТИВНЫХ процесса Writer(использовать мьютексы)** и **два АКТИВНЫХ процесса Reader(использовать семафор)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Administrator**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Writer(Reader)**;
3. запрашивает у пользователя кол-во отправленных сообщений для процессов **Writer и** кол-во полученных сообщений **Reader (общее количество отправленных и принятых сообщений должно совпадать)**;
4. запускает заданное количество процессов **Reader и Writer**;
5. принимает от каждого процесса **Writer** сообщения и выводит на консоль, затем отправляет сообщения процессам **Reader.**
6. принимает от каждого процесса **Reader и Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
7. завершает свою работу.

Процесс **Writer**:

1. синхронизировать работу процессов **Writer** с помощью **мьютексов**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее из *“A” или “B”,* и передает их (по одному) процессу **Administrator**;
4. передает сообщение о завершении сеанса процессу **Administrator**;
5. завершает свою работу.

Процесс **Reader**:

1. синхронизировать работу процессов **Reader** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. принимает сообщения *«C» , «D»* от процесса **Administrator**;
4. выводит на консоль сообщения;
5. передает сообщение о завершении сеанса процессу **Administrator**;
6. завершает свою работу.

4.19. Написать программы для консольного процесса Boss и консольных процессов Parent, Child. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом***), которые обозначают «A» , «B», «C» , «D» и конец сеанса для процессов* **Parent и Child.**

Принимать сообщение можно **только от ТРЕХ АКТИВНЫХ процессов Child(использовать семафор) и одного АКТИВНОГО процесса Parent(использовать мьютекс)**, передача остальных сообщений от других процессов должна временно блокироваться (находиться в режиме ожидания).

Процесс **Boss**:

1. Инициализация объектов синхронизации;
2. запрашивает у пользователя количество процессов **Parent** и количество процессов **Child**, которые он должен запустить;
3. запрашивает кол-во сообщений, принятых от **Parent или Child (общее количество отправленных и принятых сообщений должно совпадать)**
4. запускает заданное количество процессов **Parent, Child**;
5. принимает от каждого процесса **Parent, Child** сообщения, выводит и кто его отправил на консоль в одной строке.

6. завершает свою работу.

Процесс **Parent**:

1. синхронизировать работу процессов **Parent** с помощью **мьютекса**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее *«A» или «B»* и передает их (по одному) процессу Boss;
4. завершает свою работу.

Процесс **Child**:

1. синхронизировать работу процессов **Child** с помощью **семафора**
2. передачу сообщений реализовать с помощью **событий**
3. запрашивает с консоли сообщения, состоящее *«C» или «D»* и передает их (по одному) процессу Boss;
4. завершает свою работу.

Лабораторная работа №5

Сдать до 5.5

Тема: "Обмен данными по анонимному каналу с сервером".

Общее задание:

1. В некоторых вариантах использовать события.

Индивидуальные варианты:

5.1. Написать программы двух консольных процессов **Server и Client**, которые обмениваются сообщениями по анонимным каналам **(2 шт.):** 1 процесс записывает в первый канал, читает из второго канала, 2 процесс записывает во второй канал, читает из первого канала .Создать наследуемые дескрипторы каналов.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

- Размер массива вводится с консоли. Тип массива: **long**
- Запрашивает число числа N и M (N < M).
- Запускает процесс Client.
- Передает процессу-**Client** по анонимным каналам размер массива и числа N и M.
- Получает от процесса-**Client** по анонимным каналам элементы массива.
- Выводит переданную и полученную информацию по каналу на консоль.
- Закончить работу после нажатия клавиши - “Q”.

Процесс- Client , который выполняет следующие действия.

- Генерирует элементы массива в диапазоне [N ,M] и передает их по анонимному каналу процессу **Server**.
- Выводит полученную и переданную информацию по каналу на консоль.
- Закончить работу после нажатия клавиши - “Q”.
- Заканчивает работу.

5.2. Написать программы для консольных процессов **Server и Part,** которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

- Размер массива вводится с консоли. Тип массива: **short**
- Генерирует элементы массива
- Запускает процесс **Part**.
- Передаёт массив процессу **Part**.
- Получает массив от процесса- **Part** .
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Part, который выполняет следующие действия.

- Получает размер массива чисел по анонимному каналу от процесса **Server**
- Получает массив чисел по анонимному каналу от процесса **Server**
- Запрашивает число числа N и M (N < M).
- Определяет какие из чисел попали в отрезок [N,M], передаёт их по анонимному каналу процессу **Server**.
- Элементы массива передаются поэлементно.
- Выводит переданную и полученную информацию по каналу на консоль.
- Заканчивает работу.

5.3. Написать программы для консольных процессов **Server и Sum,** которые обмениваются сообщениями по анонимному каналу. Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

- Размер массива вводится с консоли. Тип массива: **double**
- Генерирует элементы массива
- Запускает процесс **Sum**.
- Запрашивает с консоли число N.
- Передает число N, размер массива процессу **Sum**
- Передаёт массив процессу **Sum**.
- Получает массив от процесса- **Sum** .
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Sum, который выполняет следующие действия.

- Получает число N, размер массива, массив по анонимному каналу от процесса-сервера
- Находит числа в массиве >N
- Выводит полученный массив на консоль.
- Вычисляет сумму квадратов чисел массива, больших N
- Передаёт квадраты элементов массива по анонимному каналу процессу-серверу.
- Передаёт сумму по анонимному каналу процессу-серверу.
- Выводит сумму на консоль.

5.4. Написать программы для консольных процессов **Server и Mult**, которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

- Размер массива вводится с консоли. Тип массива: **float**
- Генерирует элементы массива
- Запускает процесс **Mult**.
- Передаёт массив процессу **Mult**.
- Получает результат от процесса- **Mult** .
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс- Mult, который выполняет следующие действия.

- Получает массив чисел по анонимному каналу от процесса- **Server**.
- Выводит полученный массив
- Вычисляет произведение чисел массива
- Передаёт произведение по анонимному каналу **Server**.
- Выводит произведение на консоль

5.5. Написать программы для консольных процессов **Server и Sort**, которые обмениваются сообщениями по анонимным каналам **(2 шт.):** 1 процесс записывает в первый канал, читает из второго канала, 2 процесс записывает во второй канал, читает из первого канала . Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

- Размер массива вводится с консоли. Тип массива: **__int8**
- Генерирует элементы массива
- Запускает процесс **Sort**.
- Передаёт массив процессу **Sort**.
- Получает массив от процесса **Sort;**
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Sort, который выполняет следующие действия.

- Получает массив символов по анонимному каналу от процесса **Server**;
- Сортирует массив;
- Передаёт отсортированный массив по анонимному каналу процессу.
- Элементы массива передаются поэлементно.
- Выводит отсортированный массив на консоль.

5.6. Написать программы для консольных процессов Server и Hignt, которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

Размер массива вводится с консоли. Тип массива: **_ _int16**

- Запускает процесс **Hignt**.
- Передаёт размер массива процессу **Hignt**.
- Получает массив от процесса **Hignt;**
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Hignt, который выполняет следующие действия.
- Получает размер массива чисел по анонимному каналу от процесса- **Server**
- Генерирует элементы массива
- Запрашивает число N.
- Определяет какие из чисел массива >N передаёт их по анонимному каналу процессу- **Server**.
- Выводит полученные числа на консоль.

5.7. Написать программы для консольных процессов Server и Simple, которые обмениваются сообщениями по анонимному каналу. Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
Размер массива вводится с консоли. Тип массива: **_int32**
- Запускает процесс **Simple**.
- Передаёт размер массива процессу **Simple**.
- Получает массив от процесса **Simple;**
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Simple, который выполняет следующие действия.
- Получает размер массива чисел по анонимному каналу от процесса- **Server**
- Генерирует элементы массива
- Находит и передает простые числа по анонимному каналу процессу-серверу.
- Выводит полученные числа на консоль.
- Элементы массива передаются поэлементно.

5.8. Написать программы для консольных процессов Server и **Small**, которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива и элементы массива вводятся с консоли. Тип массива: **int**
- Число N вводится с консоли
- Запускает процесс **Small**.
- Передаёт размер массива, элементы массива и число N процессу **Small**.
- Получает массив от процесса **Small;**
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс- Small, который выполняет следующие действия.
- Получает размер массива и массив чисел по анонимному каналу от процесса-сервера
- Определяет какие из чисел >0 и <N
- Передаёт их количество и сами числа по анонимному каналу процессу-серверу. Элементы массива передаются поэлементно.
- Выводит полученные числа на консоль.

5.9. Написать программы для консольных процессов Server и **Alfavit**, которые обмениваются сообщениями по анонимным каналам **(2 шт.):** 1 процесс записывает в первый канал, читает из второго канала, 2 процесс записывает во второй канал, читает из первого канала .Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива и элементы массива вводятся с консоли. Тип массива: **char**
- Число N вводится с консоли
- Запускает процесс **Alfavit l**.
- Передаёт размер массива и элементы массива процессу **Alfavit**.
- Получает массив от процесса **Alfavit;**
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Alfavit, который выполняет следующие действия.
- Получает массив символов по анонимному каналу от процесса-сервера.
- Определяет символы, принадлежащие латинскому алфавиту и передает их по анонимному каналу процессу-серверу.
- Выводит оба массива на консоль.

- Элементы массива передаются поэлементно.

5.10. Написать программы двух консольных процессов **Server и Figure**, которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива вводится с консоли. Тип массива: **char**
- Запускает процесс **Figure**.
- Передает процессу- **Figure** по анонимному каналу размер массива символов.
- Получает от процесса- **Figure** по анонимному каналу массив символов. Выводит полученную и переданную информацию на консоль.
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Figure, который выполняет следующие действия.
- Получает размер массива символов по анонимному каналу от процесса-сервера.
- Генерирует элементы массива
- Определяет цифры и передает их по анонимному каналу процессу-серверу.
- Выводит переданную и полученную информацию по каналу на консоль.
- Элементы массива предаются по одному.

5.11. Написать программы двух консольных процессов **Server и Palindrom**, которые обмениваются сообщениями по анонимному каналу. Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива и элементы массива вводятся с консоли. Тип массива: **char**
- Запускает процесс **Palindrom**.
- Передает процессу- **Palindrom** по анонимному каналу размер массива символов и элементы массива.
- Получает от процесса- **Palindrom** по анонимному каналу массивы(палиндромы) символов.
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс-Palindrom, который выполняет следующие действия.
- Получает размер массива символов и элементы массива по анонимному каналу от процесса-сервера.
- Запрашивает символ-разделитель для лексем в строке
- Находит среди лексем палиндромы в строке и передает палиндромы по анонимному каналу процессу-серверу.
- Выводит переданную и полученную информацию по каналу на консоль.

5.12. Написать программы двух консольных процессов **Server и Consume**, которые обмениваются сообщениями по анонимным каналам **(2 шт.):** 1 процесс записывает в первый канал, читает из второго канала, 2 процесс записывает во второй канал, читает из первого канала .Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива и элементы массива вводятся с консоли. Тип массива: **char**
- Запускает процесс **Consume**.
- Передает процессу- **Consume** по анонимному каналу размер массива символов и элементы массива.
- Получает от процесса- **Consume** по анонимному каналу массив символов.
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс- Consume, который выполняет следующие действия.
- Получает размер массива символов и элементы массива по анонимному каналу от процесса-сервера.
- Выводит полученные данные на консоль.
- Количество чисел, которые должны быть потреблены, запрашивается с консоли.
- Индексы потреблённых чисел генерируются случайно
- Выводит оставшиеся числа на консоль и передает её по анонимному каналу процессу-серверу.

5.13. Написать программы двух консольных процессов **Server и Searh**, которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива1 и элементы массива1 вводятся с консоли. Тип массива: **char**
- Размер массива2 и элементы массива2 вводятся с консоли. Тип массива: **_int8**

- Запускает процесс **Searh**.
- Передает процессу- **Searh** по анонимному каналу размеры массивов и элементы массивов.
- Получает от процесса- **Searh** по анонимному каналу итоговый массив
- Выводит полученную и переданную информацию на консоль.

Процесс- Searh, который выполняет следующие действия.
- Получает каналу размеры массивов и элементы массивов по анонимному каналу от процесса-сервера.
- Выводит полученные данные на консоль.
- Определяет совпадающие элементы (цифры) из массивов,
- Выводит новый массивов на консоль
- Передает новый массив по анонимному каналу процессу-серверу.

5.14. Написать программы двух консольных процессов **Server и Union**, которые обмениваются сообщениями по анонимному каналу.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива и элементы массива вводятся с консоли. Тип массива: **float**
- Запускает процесс **Union**.
- Передает процессу- **Union** по анонимному каналу размер и элементы массива символов.
- Получает от процесса- **Union** по анонимному каналу размер и элементы массива3. Выводит полученную и переданную информацию на консоль.

Процесс-Union, который выполняет следующие действия.
- Получает размер и элементы массива символов по анонимному каналу от процесса-сервера.
- Генерирует Размер массива2 и элементы массива2. Тип массива:**float**
- Объединяет массивы в массив3(тип **float**) и передает по анонимному каналу процессу-серверу.
- Выводит переданную и полученную информацию по каналу на консоль.
- Элементы массива передаются по одному.

5.15. Написать программы двух консольных процессов **Server и Searh**, которые обмениваются сообщениями анонимным каналам **(2 шт.):** 1 процесс записывает в первый канал, читает из второго канала, 2 процесс записывает во второй канал, читает из первого канала .Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива1 и элементы массива1 вводятся с консоли. Тип массива: **_ int16**
- Размер массива2 и элементы массива2 вводятся с консоли. Тип массива: **_ _int8**
- Запускает процесс **Searh**.
- Передает процессу- **Searh** по анонимному каналу размеры массивов и элементы массивов.
- Получает от процесса- **Searh** по анонимному каналу итоговый массив (пары индексов). Выводит полученную и переданную информацию на консоль.

Процесс- Searh, который выполняет следующие действия.
- Получает каналу размеры массивов и элементы массивов по анонимному каналу от процесса-сервера.
- Выводит полученные данные на консоль.
- Определяет не совпадающие элементы из массивов, удалить дубли.
- Выводит выводит новый массив на консоль
- Передает новый массив по анонимному каналу процессу-серверу.

5.16. Написать программы двух консольных процессов **Server и Searh**, которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива1 и элементы массива1 вводятся с консоли. Тип массива: **_ int64**
- Запускает процесс **Searh**.
- Передает процессу- **Searh** по анонимному каналу размеры массивов и элементы массивов.
- Получает от процесса- **Searh** по анонимному каналу итоговый массив (пары индексов). Выводит полученную и переданную информацию на консоль.

Процесс- Searh, который выполняет следующие действия.
- Получает каналу размеры массивов и элементы массивов по анонимному каналу от процесса-сервера.
- Выводит полученные данные на консоль.

- Получает «разность» массивов: совпадающие элементы удалить из первого массива.
- Выводит новый массив, образованный из первого на консоль
- Передает новый массив по анонимному каналу процессу-серверу.

5.17. Написать программы двух консольных процессов **Server и Consume**, которые обмениваются сообщениями по анонимному каналу. Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива и элементы массива вводятся с консоли. Тип массива: **char**
- Запускает процесс **Consume**.
- Передает процессу- **Consume** по анонимному каналу размер массива символов и элементы массива.
- Получает от процесса- **Consume** по анонимному каналу массив символов.
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс- Consume, который выполняет следующие действия.
- Получает размер массива символов и элементы массива по анонимному каналу от процесса-сервера.
- Выводит полученные данные на консоль.
- Определяет знаки препинания.
- Выводит найденнын символы на консоль и передает её по анонимному каналу процессу-серверу.
-

5.18. Написать программы для консольных процессов **Server и Sum,** которые обмениваются сообщениями по анонимному каналу. Создать ненаследуемый дескриптор канала и создать наследуемый дубликат дескриптора канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива вводится с консоли. Тип массива: **double**
- Генерирует элементы массива
- Запускает процесс **Client**.
- Передает размер массива процессу **Client**.
- Передаёт массив процессу **Client**.
- Получает массив от процесса- **Client**.
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс- Client, который выполняет следующие действия.
- Получает число N, размер массива, массив по анонимному каналу от процесса-сервера
- Находит числа в массиве >N
- Выводит полученный массив на консоль.
- Вычисляет квадратные корни чисел массива, больших N
- Передаёт элементы полученного массива по анонимному каналу процессу-серверу.
- Выводит элементы полученного массива на консоль.

5.19. Написать программы для консольных процессов **Server и Mult**, которые обмениваются сообщениями анонимным каналам**(2 шт.):**: 1 процесс записывает в первый канал, читает из второго канала, 2 процесс записывает во второй канал, читает из первого канала .Создать наследуемый дескриптор канала.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:
- Размер массива вводится с консоли. Тип массива: **float**
- Генерирует элементы массива
- Число с вводится с консоли
- Запускает процесс **Client**.
- Передаёт массив процессу **Client**.
- Получает результат от процесса- **Client**.
- Выводит переданную и полученную информацию по каналу на консоль.

Процесс- Client, который выполняет следующие действия.
- Получает массив чисел по анонимному каналу от процесса- **Server**.
- Выводит полученный массив
- Находит элементы массива из диапазона [N, N2]
- Передаёт новый массив по анонимному каналу **Server**.
- Выводит новый массив на консоль
-

Дополнительное задание (штрафное для не сдавших предыдущие лабораторные):

Процесс- Server, выполняет следующие действия (в основной проект дописать):

- Создать 2 канал
- Создать **процесс Controler**
- Передать все данные в **процесс Controler**

Процесс- Controler, выполняет следующие действия:

- Выводит всю полученную информацию на консоль

Лабораторная работа №6а
Тема: «Создание и синхронизация потоков в стандарте с++ 11 мьютексов и condition_variable(или condition_variable_any), бинарный семафор(стандарт с++20)».

Общее задание:

- Реализовать лабораторную 3 на с++11 (стандарт с++11):
  1. Использовать для создания потоков и работы с ними std::thread.
  2. События реализвать через std::condition_variable_any (или std::condition_variable) и std::mutex
  3. Вместо критической секции (win32) - использовать std::mutex или std::binary_semaphore (C++20)

Лабораторная работа 6б (вместо лабораторной 6а)
Для тех, у кого компьютеры под UNIX
Тема: « Создание и синхронизация потоков в Unix с помощью библиотеки Posix».

Общее задание:

1. Реализовать лабораторную 3

Лабораторная работа * (бонусная)
Тема: Классические задачи.

Номер на выбор:

1. Задача "Производители-Потребители" (Producer-Consumer problem);
2. Задача"Читатели-Писатели"(Readers-Writers problem);
3. Задача"Обедающие философы"(DiningPhilosopher problem);
4. Задача"Спящий брадобрей"(SleepingBarber problem).