Лабораторная работа №1

Сдать до 17.09

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 фкнкциями _beginthreadex и CreateThread
6. Дождаться завершения потока `worker`.
7. Вывести на консоль результат работы потока `worker`
8. Завершить работу.

b) Глобальные переменные не использовать!

c) Розбраться с типами данных, которые используются

d) Обяснить: что такое идентификатор и дескриптор, работу функций.

Индивидуальные варианты:

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

1. Вывести максимальный элемент из отрицательных элементов массива. После поиска «спать» 100 миллисекунд Завершить свою работу.
2. Вывести минимальный элемент из положительных элементов массива. Завершить свою работу.
3. Вывести количество четных элементов из элементов массива. Завершить свою работу.
4. Вывести элементы из отрезка [a,b]. Завершить свою работу.
5. Ввести количество нечетных элементов из элементов массива. Завершить свою работу.
6. Вывести количество элементов кратных 3 из элементов массива. Завершить свою работу.
7. Вывести количество элементов кратных 5 из элементов массива. Завершить свою работу.
8. Вывести количество элементов кратных 9 из элементов массива. Завершить свою работу.
9. Ввести новый элемент X - вещественный. Найти количество элементов массива, целая часть которых совпадает с целой частью X. Завершить свою работу.
10. 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 24.09) 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 24.09

Общее задание:

1. Два проекта (процессы) хранить в одном Solution (Решении) в VS Studio!
2. Написать программу для консольного процесса Child, использовать рекомендации (в материалах) для отладки проекта с входной командной строкой.
3. Настроить консольный процеса **Parent** для работы со строками типа char и :
   a) командную строку собирать как строку типа char;
   b) если командную строку типа string - преобразовать в строку типа char;

4a. Можно настроить проект для процесса Parent для работы со строками типа Unicode, изучить функции склеивания строк unicode и собрать командную строку или :
   a) если командную строку собирали как строку char - преобразовать в строку unicode;
   b) если командную строку типа string - преобразовать в строку unicode.
4. Объяснить какие данные хранят структуры : STARTUPINFO, PROCESS_INFORMATION.
5. Написать программу для процессов **Parent и Child** (согласно индивидуальным заданиям) :

Процесс `Parent`:

- 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 "**Child**" устанавливает(STARTUPINFO) визуальные настройки, **согласно индивидуальным вариантам**:
  1. Установить высоту буфера для Child.
  2. Установить ширину (X) смещения от верхнего левого угла экрана.
  3. Установить высоту (Y) смещения от верхнего левого угла экрана.
  4. Установить любой цвет фона (не черный) для Child. (в Windows11 установить новый заголовок для окна)
  5. Установить новый заголовок для окна Child.
  6. Установить любой цвет текста (не белый) для Child. (в Windows11 установить новый заголовок для окна)
  7. Установить ширину (X) смещения от верхнего левого угла экрана.
  8. Установить высоту (Y) смещения от верхнего левого угла экрана.
  9. Установить любой цвет фона (не черный) для Child. (в Windows11 установить новый заголовок для окна)
  10. 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. 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.
  2. Приведение массива к палиндрому (получившейся палиндром поместить в массиве слева, а лишние элементы соответственно – справа). Полученный массив вывести. Тип элементов - символы
  3. Сортировка вставками. Полученный массив вывести. Тип элементов - целые числа.
  4. Сортировка Шелла. Полученный массив вывести. Тип элементов - вещественные числа.
  5. Поиск в массиве чисел кратных 3. (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа.
  6. Поиск в массиве чисел кратных 5. (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа без знака.
  7. Сортировка Хоара. Полученный массив вывести. Тип элементов - целые числа.
  8. Сортировка Подсчетом. Полученный массив вывести. Тип элементов - целые числа.
  9. Сортировка бинарная. Тип элементов - целые числа.

10. Поиск в массиве элементов >0 (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Полученный массив вывести. Тип элементов - целые числа.

Примечания.

1.Для ожидания завершения работы процесса `Child` и его потока использовать функцию WaitForSingleObject

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

Дополнительное (или штрафное после 24.09) 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.

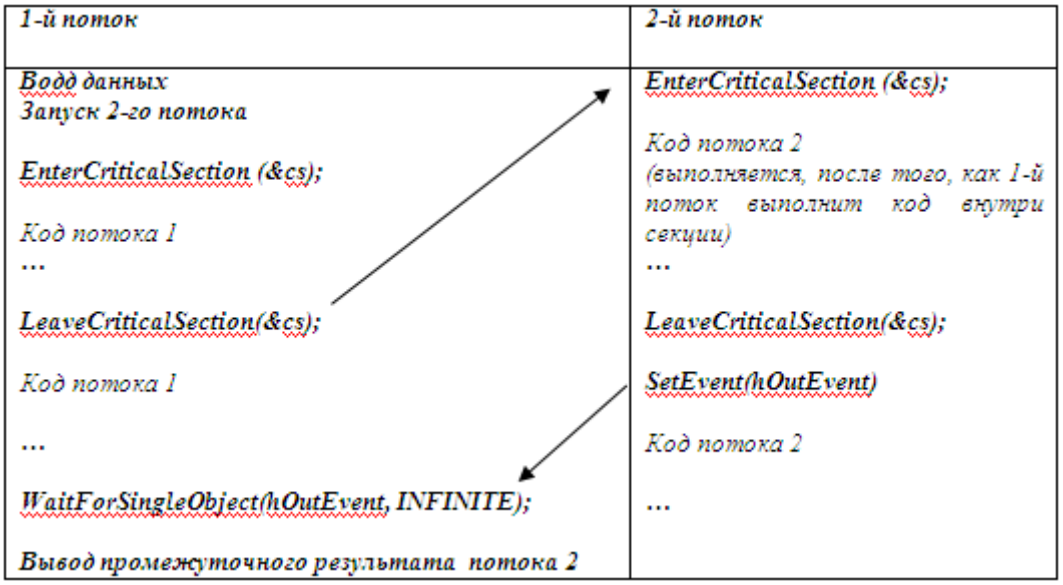
Лабораторная работа №3.
Сдать до 15.10

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

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 :



Общее задание:

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

1. Перед выполнением задания, нарисовать (**на бумаге**) примерную схему синхронизации потоков и показать преподавателю!
2. Если событие не помечено словами в скобках: “ручной сброс”, то оно должно быть с автоматическим сбросом.
3. После входа потоком в критичекую секцию, выводить на консль текст «вход в секцию потоком X»

Индивидуальные варианты:

3.1 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 или генерируются случайно;
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 **work – *событие, 1с потоком* main** - *использовать обытие 2* **):**

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***);

3.2 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 double**), размерность и элементы которого вводятся пользователем с консоли или генерируются случайно;
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.
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* **критическую секцию и *событие***):

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***);

3.3 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 или генерируются случайно;
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* ***событие2***;
10. получить от потока **SumElement** сигнал о выводе результата (*использовать* ***критическую секцию)***
11. вывести на экран результат работы потока **SumElement**;
12. дождаться завершения потока **work**
13. ыыводить 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 вывода массива, момент запуска произойдёт после того, будут отсортированы k элементов массива (*использовать* ***событие***);

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком* ***main,*** *использовать* ***критическую секцию и и событие2*****):**

1. ждёт от потока **main** сигнал о начале суммирования (*использовать* ***событие2***);
   посчитать количество цифр массива до заданной позиции k;
2. известить(*использовать* ***критическую секцию)*** поток **main** о выводе результата

3.4 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 или генерируются случайно;
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 (<размера мссива);
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(*использовать* ***критическую секцию 1***);
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 2***);
13. Дождаться сигнала потока **SumElement** (*использовать* ***событие***);
14. 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 элементов, соответствующих цифрам (слева поместить в массив цифры, а остальные элементы массива - заполнить пробелами). Элементы - символы;
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. известить поток **main** о начале вывода итогового массива (*использовать* ***критическую секцию 1***).

Поток **SumElement** должен выполнить следующие действия (*Для синхронизации с потоком* **main -** *использовать* ***критическую секцию №2, событие!*****):**

1. ждёт от потока **main** сообщения о начале суммирования (*использовать* ***критическую секцию №2***);
2. вычислить среднее арифметическое цифр до позиции k, если цифр меньше k, то k уменьшить до количества цифр);

3. сигнализировать потоку **main** о выводе результата (*использовать **событие***);

3.5 Объекты синхронизации :

Событие №1 устанавливает поток ***work для потока main*** (для вывода масива 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 или генерируются случайно;
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. получить от потока **MultElement** сигнал о выводе результата (*использовать **критическую секцию)***
12. 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 (0- неумножать);
3. известить поток **main** о выводе результата (*использовать **критическую секцию)*** .

3.6 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 или генерируются случайно;
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. Выводить на экран элементы массива (итогового);
8. Получить от потока **CountElement** сигнал о выводе результата(*использовать **критическую секцию***);
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 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 ручной сброс***);
2. выполнить подсчёт элементов (до символов подчеркивания: «_») итогового массива,;
3. сигнализировать потоку **main** о выводе результата (*использовать **критическую секцию***);

3.7 Объекты синхронизации :

Событие1 (с ручным сбросом) **–** в потоке ***main*** устанавливает сигнал **для потока *work*** (для начала работы) и для потока ***MultElement (*** *сигнализирует о начале вычислений)**.*

Критическая секция- синхронизация ***work*** и потока **main**(сигнализирует о начале вывода результатов ***work*** *в потоке* ***main);***

Событие 2 - 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 - **символы**, размерность и элементы которого вводятся пользователем с консоли или генерируются случайно;
3. вывести размерность и элементы исходного массива на консоль;
4. запустить поток **work**
5. запустить поток **MultElement**;
6. ввести заначеня A и B.
7. известить потоки **work** и **MultElement** о начале работы (*использовать* ***событие с ручным сбросом).***
8. Получить от потока **work** сигнал о выводе массива (использовать критическую секцию);
9. вывести на экран результат работы потока **work .**
10. Получить от потока **MultElement** сигнал о выводе результата (использовать событие2);
11. вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

1. Ждать сигнала от потока main о начале работы (*использовать* ***событие с ручным сбросом).***
2. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
3. Поиск в массиве элементов из диапазона [A,B] (разместить их в новом массиве слева, остальные элементы массива - заполнить *).
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. выводить на экран поэлементно элементы массива (итогового);
6. известить поток **main** о выводе результатов 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 ручным сбросоми и событие2***):

1. Ждать сигнала от потока **main** о начале работы (*использовать* ***событие с ручным сбросом).***
2. выполнить произведение полученных цифр (если есть) в **начальном** массиве
3. известить поток **main** о выводе результата (*использовать* ***событие2***);

3.8 Объекты синхронизации :

Событие1 (с ручным сбросом) **–** в потоке ***main*** устанавливает сигнал **для потока *work*** (для начала работы) и для потока ***CounttElement (*** *сигнализирует о начале вычислений)**.*

Событи 2 – устанавливает поток ***work*** для потока **main**(сигнализирует о начале вывода результатов ***work*** *в потоке* ***main);***

Критическая секция – между потоком ***CounttElement*** и потоком ***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 или генерируются случайно;
3. вывести размерность и элементы исходного массива на консоль;
4. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
5. Запустить поток **work**;
6. Запустить поток **CountElement**;
7. Запросить число X.
8. известить потоки **work** и **CountElement** о начале работы (*использовать* ***событие с ручным сбросом).***
9. Получить от потока work сигнал о выводе массива (*использовать* ***событие 2).***
10. вывести на экран результат работы потока **work .**
11. Дождаться сигнала потока **CountElement** (использовать критическую секцию);
12. вывести на экран результат работы потока **CountElement**;

Поток **work** должен выполнить следующие действия:

1. Ждать сигнала от потока **main** о начале работы (*использовать* ***событие с ручным сбросом).***
2. Поиск в массиве отрицательных элементов кратных 3 (разместить их в новом массиве слева, остальные элементы массива - справа).
3. после каждого готового элемента отдыхать в течение заданного интервала времени;

4. известить поток **main** о выводе результатов работы (момент запуска произойдёт после того, будет сформирована итоговый массив (*использовать **событие 2).***

Поток **CountElement** должен выполнить следующие действия (*Для синхронизации с потоком **main-** использовать **событие с ручным сбросом и критическую секцию***

1. Ждать сигнала от потока **main** о начале работы (*использовать **событие с ручным сбросом).***
2. подсчитать количество неотрицательных элементов=X, в начальном массиве;
3. известить поток main о выводе результата(использовать критическую секцию);

3.9 Объекты синхронизации :

Событие (с ручным сбросом) **–** в потоке ***main*** устанавливает сигнал **для потока *work*** (для начала работы) и для потока ***CounttElement (*** *сигнализирует о начале вычислений).*

Критическая секция – между потоком ***work*** и потоком ***main*** (для вывода в ***main результата work***).

Критическая секция 2 – между потоком ***CountElement*** и потоком ***main*** (для вывода в ***main результата 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 или генерируются случайно;
3. вывести размерность и элементы исходного массива на консоль;
4. запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;
5. Запустить поток **work**;
6. Запустить поток **CountElement**;
7. Запросить число X.
8. известить потоки **work** и **CountElement** о начале работы (*использовать **событие с ручным сбросом).***
9. Получить от потока **work** сигнал о выводе массива (использовать ***критическую секцию***);
10. вывести на экран результат работы потока **work** (использовать критическую секцию);
11. Дождаться сигнала потока **CountElement** (использовать критическую секцию);
12. вывести на экран результат работы потока **CountElement**;

Поток **work** должен выполнить следующие действия:

1. Ждать сигнала от потока **main** о начале работы (*использовать **событие с ручным сбросом).***
2. поиск в массиве элементов =X (разместить их в новом массиве слева, остальные элементы массива - справа).
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. известить поток **main** о выводе результатов работы (момент запуска произойдёт после того, будет сформирована итоговый массив (использовать ***критическую секцию*****).**

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

1. Ждать сигнала от потока **main** о начале работы (*использовать **событие с ручным сбросом).***
2. подсчитать количество элементов не равных X в начальном массиве;
3. известить поток main о выводе результата(использовать ***критическую секцию***);

Лабораторная работа №4.
Сдать до 12.11
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, A или B от одного процесса другому, в данном задании означает : создаем события соответсвующие сообщениям A и B. Затем вводится одно из сообщений (A или B) 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 "А"(два события) , сообщение "В" (два события), сообщение "С"(два событи) 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 (*"А" или "В" или "С")* 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 "А" , сообщение "В", 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) всеми **Parent и Child (**общее количество отправленных сообщений потоками **Parent =** общему количеству полученных сообщений потоками **Child)** ;
4. запускает заданное количество процессов **Parent, Child**;
5. запускает заданное количество процессов **Parent, Child**;
6. запрашивает с консоли (можно автоматически получив сообщ. A - отправить сообщение C, получив сообщ. B - отправить D) и отправляет сообщение для процессов **Child**
7. принимает от процессов **Parent** сообщения, выводит на консоль в одной строке.
8. принимает от процессов Child и Parent сообщение о завершении сеанса процесса
9. 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;
4. передает сообщение о завершении сеанса процессу **Boss**
5. 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.

Лабораторная работа №5

Сдать до 26.11

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

Общее задание:

1. В некоторых вариантах использовать события или мьютексы.

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 ненаследуемые дескрипторы канала и создать наследуемые дубликаты дескрипторов 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 наследуемые дескрипторы 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 ненаследуемые дескрипторы канала и создать наследуемые дубликаты дескрипторов 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 наследуемые дескрипторы 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 ненаследуемые дескрипторы канала и создать наследуемые дубликаты дескрипторов 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 наследуемые дескрипторы 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 не наследуемые дескрипторы канала и создать наследуемые дубликаты дескрипторов 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 наследуемые дескрипторы 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.

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

Сдать до 10.12

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. «Подвешенные» потоки (как в Win32 API) создать нельзя, данные вводите до запуска потоков (в некоторых вариантах).
2. События реализвать через std::condition_variable_any (или std::condition_variable) и std::mutex
3. Вместо критической секции (Win32 API) - использовать std::mutex или std::binary_semaphore (C++20)

Дополнительное :

Для передачи данных из одногопотока в другой использовать std::async, std::future и std::promise

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:

Реализовать лабораторную 3:

1. Использовать для создания потоков и работы с ними p_thread. «Подвешенные» потоки (как в Win32 API) создать нельзя, данные вводите до запуска потоков(в некоторых вариантах).
2. События реализвать через condition variable
3. Вместо критической секции (Win32 API) - использовать мьютекс или бинарный семафор.

Лабораторная работа 6в (вместо лабораторной 6а или 6б)
Тема: « Создание и синхронизация потоков с помощью библиотеки boost».

Лабораторная работа 7 (если сданы предыдущие)
Тема: Классические задачи.

Номер на выбор:

1. Задача "Производители-Потребители" (Producer-Consumer problem), доступ к стеку;
2. Задача "Производители-Потребители" (Producer-Consumer problem), доступ к очереди (усложненная);
3. Задача"Читатели-Писатели" (Readers-Writers problem);
4. Задача”Обедающие философы” (DiningPhilosopher problem);
5. Задача”Спящий брадобрей” (SleepingBarber problem).
6. Задача о “Курильщиках” (Smokers problem).
7. Задача “Санта-Клауса” (Santa Claus problem).