

Лабораторная работа №1

Сдать до 10.09

Тема: «Создание потоков».

Общее задание:

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

а) 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, массив и т.д.
4. Приостановить поток **worker** (SuspendThread), затем через некоторое время снова запустить поток.
5. Уметь создавать поток функциями `_beginthreadex` и `CreateThread`
6. Дождаться завершения потока **worker**.
7. Вывести на консоль результат работы потока **worker**
8. Завершить работу.

б) Глобальные переменные не использовать!

с) Разобраться с типами данных, которые используются.

д) Объяснить: что такое идентификатор и дескриптор, работу функций.

Индивидуальные варианты:

д) 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.

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 17.09) задание:

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

с. 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. В Solution (Решении) настроить, чтобы *.exe файлы лежали в одном Debug!
3. Написать программу для консольного процесса Child, использовать рекомендации (в материалах) для отладки проекта с входной командной строкой.
- 4a. Настроить консольный процесса **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;
- 4b. 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 процесса “Child” устанавливает(STARTUPINFO) визуальные настройки, **согласно индивидуальным вариантам**:
 1. Установить любой цвет текста (не белый) для Child. (в Windows11 не работает...)
 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. (в Windows11 не работает...)
 7. Установить новый заголовок для окна Child.
 8. Установить любой цвет текста (не белый) для Child. (в Windows11 не работает...)
 9. Установить ширину буфера для 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.

Примечания.

1. Для ожидания завершения работы процесса Child и его потока использовать функцию WaitForSingleObject
2. В Solution (Решении) настроить, что бы .exe файлы лежали в одном Debug!

Дополнительное (или штрафное после 1.10) 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. Процесс Count выводит на консоль числа фибоначчи, по возрастанию.

Лабораторная работа №3.

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

Сдать до 15.10

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

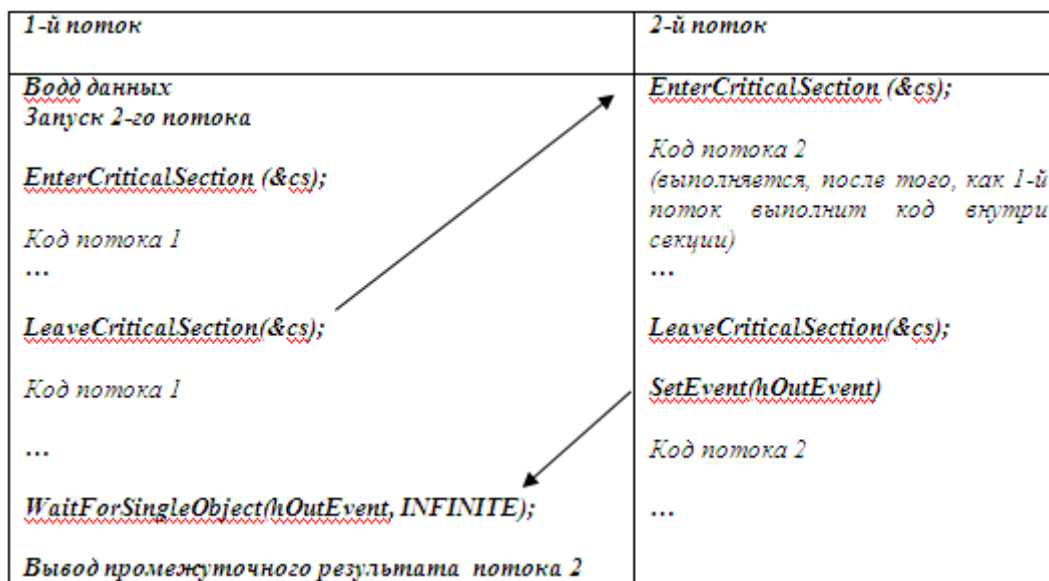
Пример для 2-х потоков синхронизации:

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:

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 критическую секцию, выводить на консль текст «вход в секцию потоком X»

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 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 массива);
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. 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:

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 до позиции k, если цифр меньше k, то k уменьшить до количества цифр);
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 **MultElement** (сигнализирует о начале выполнения 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(**тип unsigned int**), 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 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. Получить от потока **MultElement** сигнал о выводе результатов;
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 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 вывода итогового массива (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-ые элементы не умножать);
3. известить поток **main** о выводе результата (*использовать критическую секцию*) .

3.3 Объекты синхронизации :

Критическая секция- синхронизация main и потока work (сигнализирует о начале запуска work, после ввода 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 **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. запустить поток **work** ;
5. запустить поток **CountElement**
6. Запросить число K.
7. Сигнализовать 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 (ждать завершения потока **work**)

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 выводе итогового массива и начале суммирования соответственно (момент запуска произойдет после того, будет сформировано K элементов итогового массива), *использовать событие1(ручной сброс)*;

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

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

3.4 Объекты синхронизации :

Событие1 - устанавливает поток **Work** для потока **main** (для вывода итогового 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 или генерируются случайно;
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 произойдет 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). После каждого готового элемента отдыхать в течение заданного интервала времени;
3. известить поток **main** выводе итогового массива (момент запуска произойдет после того, будет сформирован массив (*использовать событие1*)).

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 итогового массива
3. известить) поток **main** о выводе результата (использовать *событие2*);

3.5 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 или генерируются случайно;
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 *событие1*);
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 *событие2*);
10. Получить от потока **SumElement** сигнал о завершении вычислений(использовать *критическую секцию*)
11. Вывести на экран результат работы потока **SumElement**;
12. Получить от потока **work** сигнал (завершение потока) о выводе итогового массива (начиная с k);
13. Выводить на экран элементы итогового массива (начиная с k);

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 неповторяющиеся элементы (разместить их в массиве слева, остальные - справа). Элементы - символы.
3. после каждого готового элемента отдыхать в течение заданного интервала времени;
4. известить поток **main** о выводе элементов итогового массива (момент запуска произойдёт после того, будут сформированы k элементов массива (использовать *событие1*);

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

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

3.6 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).

Критическая секция- синхронизация **SumElement** и потока **main** (вывод *результата 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 или генерируются случайно;
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 выводе элементов массива(использовать *событие с ручным сбросом*).
8. выводить элементы массива (до K2);
9. получить от потока **SumElement** 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:

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 слева, остальные элементы массива - заполнить нулями). Число A ввести в потоке **main**.
3. после каждого готового элемента отдыхать в течение заданного интервала времени;

4. известить поток **SumElement** и поток **main** о начале суммирования и выводе итогового массива (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

3.7 Объекты синхронизации :

Событие №1 (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**);

Событи №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 - целые числа без знака, 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 значения 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 событие);
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 нулями).
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, с потоком **work** - критическую секцию.):

1. Ждать сигнала от потока **main** о начале работы (*использовать событие с ручным сбросом*).
2. выполнить произведение элементов из диапазона [A, B] в начальном массиве
3. известить поток **main** о выводе результата (*использовать событие*);

3.8 Объекты синхронизации :

Событие1 (с ручным сбросом) – в потоке **main** устанавливает сигнал для потока **work** (для начала работы) и для потока **CountElement** (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 **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 **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 событие2*);
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). Элемент - X ввести в поток **main**.
3. выводить на экран поэлементно элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **main** о выводе массива (момент запуска произойдет после того, будет сформирован итоговый массив (*использовать событие2*);

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 в начальном массиве количество элементов равных X;
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 **CountElement** (*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 **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
10. вывести на экран результат работы потока **work** (*использовать критическую секцию*);
11. Дождаться сигнала потока **CountElement** (*использовать критическую секцию2*);
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). X ввести в поток **main**.
3. выводить на экран поэлементно элементы массива (итогового);
4. после каждого готового элемента отдыхать в течение заданного интервала времени;
5. известить поток **main** о выводе итогового массива (момент запуска произойдет после того, будет сформирован 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 2*);

3.10 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 или генерируются случайно;
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 произойдет 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 в итоговом массиве;
3. известить (*использовать событие*) поток **main** о выводе результата

Лабораторная работа №4. Сдать до 12.11

Тема: «Синхронизация процессов при помощи событий, мьютексов и семафоров».

*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 соответствующие сообщениям А и В. Затем вводится одно из сообщений (А или В) с консоли в одном процессе и устанавливается соответствующее событие в сигнальное состояние. В другом процессе ожидается одно из событий 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 сообщение "А", сообщение "В", "А2", сообщение "В2" 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**:

- Инициализация объектов синхронизации;
- запрашивает у пользователя количество процессов **Reader** и **Writer**, которые он должен запустить;
- 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);
- запускает заданное количество процессов **Reader** и **Writer**;
- принимает от каждого процесса **Reader** сообщение и выводит его на консоль в одной строке.
- принимает от каждого процесса **Reader** и **Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
- завершает свою работу.

Процесс **Writer**:

- синхронизировать работу процессов **Writer** с помощью мьютекса
- после захвата мьютекса выводить сообщение : мьютекс захвачен
- передачу сообщений реализовать с помощью событий
- запрашивает с консоли сообщения ("А" или "В") , и передает их (по одному) процессу **Reader**;
- передает сообщение о завершении сеанса процессу **Administrator**;
- завершает свою работу.

Процесс **Reader**:

- синхронизировать работу процессов **Reader** с помощью семафора

- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- принимает сообщение от процесса **Writer**;
- выводит на консоль сообщение “A” или “B” ;
- передает сообщения “A2” или “B2” - процессу **Administrator**;
- передает сообщение о завершении сеанса процессу **Administrator**;
- 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**:

- Инициализация объектов синхронизации;
- запрашивает у пользователя количество процессов **Reader** и **Writer**, которые он должен запустить;
- 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);
- 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);
- принимает от каждого процесса **Reader** и **Writer** сообщение о завершении сеанса и выводит его на консоль в одной строке.
- завершает свою работу.

Процесс **Writer**:

- синхронизировать работу процессов **Writer** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, и передает их (по одному) процессу **Reader**;
- передает сообщение о завершении сеанса процессу **Administrator**;
- завершает свою работу.

Процесс **Reader**:

- синхронизировать работу процессов **Reader** с помощью **мьютекса**
- после захвата мьютекса выводить сообщение : мьютекс захвачен
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения от процесса **Writer**;
- выводит на консоль сообщение;
- передает сообщение о завершении сеанса процессу **Administrator**;
- 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**:

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

Процесс **Writer**:

- синхронизировать работу процессов **Writer** с помощью **мьютекса**
- после захвата мьютекса выводить сообщение : мьютекс захвачен
- передачу сообщений реализовать с помощью **событий**

- запрашивает с консоли сообщения, состоящее ("А" или "В" или "С") и передает их (по одному) процессу **Administrator**;
- передает сообщение о завершении сеанса процессу **Administrator**;
- завершает свою работу.

Процесс **Reader**:

- синхронизировать работу процессов **Reader** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения от процесса **Administrator**;
- выводит на консоль сообщение;
- передает сообщение о завершении сеанса процессу **Administrator**;

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**:

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

Процесс **Writer**:

- синхронизировать работу процессов **Writer** с помощью **мьютекса**
- после захвата мьютекса выводить сообщение : мьютекс захвачен
- передачу сообщений реализовать с помощью событий с ручным сбросом
- запрашивает с консоли сообщения, и передает их (по одному) процессам **Reader**;
- передает сообщение (с автоматическим сбросом) о завершении сеанса процессу **Administrator**;
- завершает свою работу.

Процесс **Reader**:

- синхронизировать работу процессов **Reader** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения (с **ручным сбросом**) от процесса **Writer**;
- выводит на консоль сообщения;
- 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 «А», «В» 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**:

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

Процесс **Parent**:

- синхронизировать работу процессов **Parent** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- передаёт сообщения **Boss** только один активный процесс, передача остальных сообщений от других процессов должна временно блокироваться с помощью **семафора**;

- запрашивает с консоли сообщения, состоящее из «А» и передает их (по одному) процессу Boss;
- завершает свою работу.
- Принимает от процесса **Boss** о завершении работы

Процесс **Child**:

- синхронизировать работу процессов **Child** с помощью **мьютекса**
- после захвата мьютекса выводить сообщение : мьютекс захвачен
- передачу сообщений реализовать с помощью **событий**
- передаёт сообщения **Boss** только один активный процесс, передача остальных сообщений от других процессов должна временно блокироваться с помощью **мьютекса**;
- запрашивает с консоли сообщения, состоящее из «В» » и передает их (по одному) процессу Boss;
- завершает свою работу.
- 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 «А», «В», «С», «D» и конец сеанса для процессов **Reader** и **Writer**. Для сообщений «С» и «D» использовать **события с ручным сбросом**.

Одновременно принимать и отправлять сообщения могут только два **АКТИВНЫХ** процесса **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**:

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

Процесс **Writer**:

- синхронизировать работу процессов **Writer** с помощью **мьютексов**
- после захвата мьютекса выводить сообщение : мьютекс захвачен
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее из “А” или “В”, и передает их (по одному) процессу **Administrator**;
- передает сообщение о завершении сеанса процессу **Administrator**;
- завершает свою работу.

Процесс **Reader**:

- синхронизировать работу процессов **Reader** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения «С», «D» от процесса **Administrator**;
- выводит на консоль сообщения;
- передает сообщение о завершении сеанса процессу **Administrator**;
- 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 «А», «В», «С», «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**:

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

Процесс **Parent**:

- синхронизировать работу процессов **Parent** с помощью **мьютекса**
- после захвата мьютекса выводить сообщение : мьютекс захвачен

- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее «А» или «В» и передает их (по одному) процессу Boss;
- завершает свою работу.

Процесс **Child**:

- синхронизировать работу процессов **Child** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- о передачу сообщений реализовать с помощью **событий**
- о запрашивает с консоли сообщения, состоящее «С» или «D» и передает их (по одному) процессу Boss;
- о 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 «А», «В», «С», «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**:

- Инициализация объектов синхронизации;
- запрашивает у пользователя количество процессов **Parent** и количество(≤ 4) процессов **Child**, которые он должен запустить.
- запрашивает кол-во сообщений, отправленных (полученных) всеми Parent и Child (общее количество отправленных сообщений потоками Parent = общему количеству полученных сообщений потоками Child);
- запускает заданное количество процессов **Parent, Child**;
- запускает заданное количество процессов **Parent, Child**;
- запрашивает с консоли (можно автоматически получив сообщ. А - отправить сообщение С, получив сообщ. В - отправить D) и отправляет сообщение для процессов **Child**
- принимает от процессов **Parent** сообщения, выводит на консоль в одной строке.
- принимает от процессов Child и Parent сообщение о завершении сеанса процесса
- завершает свою работу.

Процесс **Parent**:

- синхронизировать работу процессов **Parent** и **Child** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее «А» или «В» и передает их (по одному) процессу Boss;
- передает сообщение о завершении сеанса процессу **Boss**
- завершает свою работу.

Процесс **Child**:

- синхронизировать работу процессов **Parent** и **Child** с помощью **семафора**
- после захвата мьютекса выводить сообщение : семафор захвачен
- передачу сообщений реализовать с помощью **событий**
- получает сообщения, состоящее «С» или «D» от процесса Boss и выводит его на консоль;
- о передает сообщение о завершении сеанса процессу **Boss**
- 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.

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

Процесс-Hight, 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 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 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 I**.
- 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 c++ 11 мьютексов и condition_variable(или condition_variable_any), бинарный семафор(стандарт c++20)».

Общее задание:

Реализовать лабораторную 3 на c++11 (стандарт c++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 реализовать 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 одного потока 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 реализовывать через `condition variable`
3. Вместо критической секции (Win32 API) - использовать мьютекс или бинарный семафор.

Лабораторная работа 6в (вместо лабораторной 6а\6б)

Тема: «Создание и синхронизация потоков с помощью библиотеки boost».

Лабораторная работа 7 (если сданы предыдущие лабораторные)

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 бравдобрей" (SleepingBarber problem).
6. Задача о "Курильщиках" (Smokers problem).
7. Задача "Санта-Клауса" (Santa Claus problem).