Лабораторная работа №1

Сдать до 20.02

Тема: «Создание потоков».

Глобальные переменные не использовать!

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

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

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

Поток **`worker`** должен выполнить следующую работу (**только пункт с индивидуальным номером**):

1. Вывести положительные элементы, кратные 4 из массива. Тип элементов - long. После каждой итерации цикла - «спать» 12 миллисекунд. Завершить свою работу.
2. Вывести элементы из массива, сумма цифр которых равна k. Тип элементов - unsigned long. После каждой итерации цикла - «спать» 20 миллисекунд. Завершить свою работу.
3. Вывести отрицательные элементы кратне 9 из массива. Тип элементов - unsigned short. После каждой итерации цикла - «спать» 30 миллисекунд. Завершить свою работу.
4. Вывести нечетные элементы из массива. Тип элементо - short. Завершить свою работу.
5. Вывести элементы из массива, у которых последняя цифра 1. Тип элементов - unsigned int. После каждой итерации цикла - «спать» 10 миллисекунд. Завершить свою работу.
6. Найти сумму квадратов элементов в массиве и вывести. Тип элементов – long. После каждой итерации цикла - «спать» 12 миллисекунд. Завершить свою работу.
7. Вывести отрицательне элементы из массива, у которых последняя цифра 0. Тип элементов - __int8. После каждой итерации цикла - «спать» 5 миллисекунд. Завершить свою работу.
8. Вывести отрицательне элементы кратные 6 из массива. Тип элементов - int. После каждой итерации цикла - «спать» 5 миллисекунд. Завершить свою работу.
9. Вывести цифры из массива. Тип элементов – char. После каждой итерации цикла - «спать» 10 миллисекунд. Завершить свою работу.
10. Вывести элементы из массива, у которых часть после запятой <0,5. Элементы массива – double. После каждой итерации цикла - «спать» 25 миллисекунд. Завершить свою работу.
11. Вывести отрицательне элементы из массива. Элементы массива – double. После каждой итерации цикла - «спать» 10 миллисекунд. Завершить свою работу.
12. Вывести элементы с отрезка [a,b] из массива. Элементы массива – double. После каждой итерации цикла - «спать» 15 миллисекунд. Завершить свою работу.
13. Вывести элементы массива, котрые будут делителями числа X. Тип элементов - long long. После каждой итерации цикла - «спать» 13 миллисекунд. Завершить свою работу.

Примечания.

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

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

где второй параметр установить равным INFINITE. Например

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

Здесь hThread – дескриптор потока `worker`.

2. Для засыпания использовать функцию:

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

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

Дополнительное (или штрафное после 20.02) задание:

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

Поток **Count** выполняет:

Выводит на консоль числа фибоначчи, по возрастанию.

Лабораторная работа №2.

Тема: «Создание процессов».
Сдать до 6.03

Задание:

Написать программы двух консольных процессов `Parent` и `Child`, которые выполняют следующие действия.

Два проекта (процессы) хранить в одном Solution (Решении)!
В Solution (Решениии) настроить, что бы .exe файлы лежали в одном Debug!

Процесс `Parent`:

- Согласно **индивидуальным вариантам** выполняет :

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

- 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. Полученный массив вывести. Тип элементов - целые числа без знака.
2. Найти в массиве повторяющиеся элементы (разместить их группы в массиве слева, остальные (одиночные) - соответственно справа). Полученный массив вывести. Тип элементов - символы.
3. Сортировка методом “пузырька”. Полученный массив вывести. Тип элементов - символы.
4. Поиск в массиве элементов из диапазона [A,B] (разместить их в массиве слева, остальные элементы массива - заполнить нулями). Полученный массив вывести. Тип элементов - вещественные числа.
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 - вещественные числа..
7. Поиск в массиве простых чисел (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа long.
8. Поиск в массиве элементов =X (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа без знака.
9. Выполнить произведение элементов (не нулевых) итогового массива до заданной позиции K. Полученный массив вывести. Тип элементов - вещественные числа.

10. Поиск в массиве лексем, (разделители – подчеркивание и знаки препинания). Полученные лексемы поместить в массиве слева, разделитель - подчеркивание, остальные элементы - заполнить символом ‘0’. Полученный массив вывести. Тип элементов массива - символы.
11. Приведение массива к палиндрому (получившейся палиндром поместить в массиве слева, а лишние элементы зменить пробелами и разместить соответственно – справа). Полученный массив вывести. Тип элементов - символы
12. Сортировка Шелла. Полученный массив вывести. Тип элементов - целые числа.
13. Поиск в массиве чисел кратных 7 (разместить их в массиве слева, остальные элементы массива - справа). Полученный массив вывести. Тип элементов - целые числа.
14. Сортировка Хоара. Полученный массив вывести. Тип элементов - вещественные числа.
15. Сортировка Подсчетом. Полученный массив вывести. Тип элементов - вещественные числа.

Примечания.

Для ожидания завершения работы процесса `Child` использовать функцию:

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

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

Здесь hProcess – дескриптор процесса `Child`.

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

Дополнительное (или штрафное после 6.03) задание:

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

Лабораторная работа №3.

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

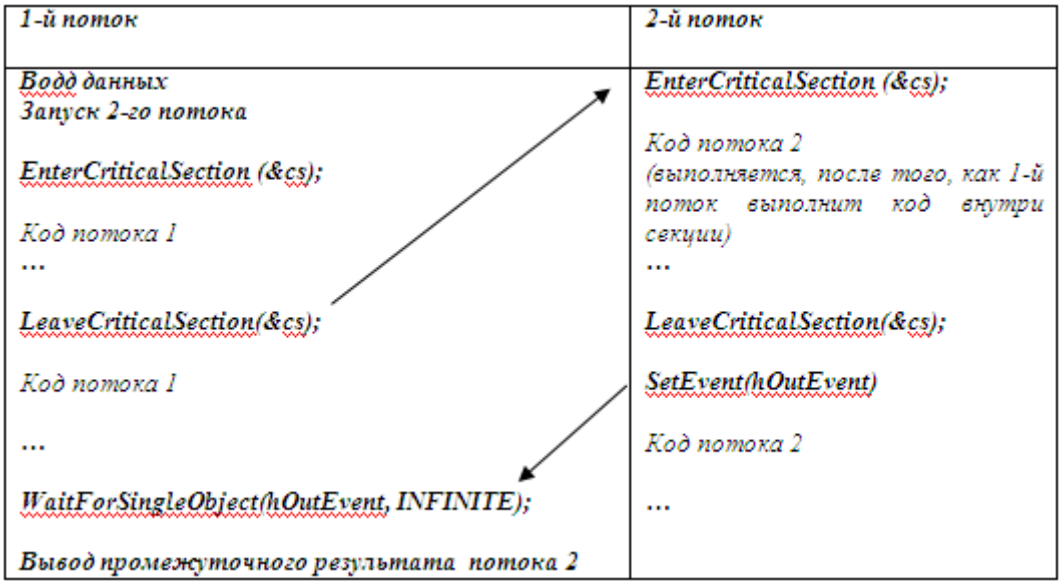
Сдать до 20.03

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

Общее задание:

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

Пример для 2-х потоков синхронизации.
В двух потоках есть общая критическая секция. Второй поток ждет освобождения секции первым потоком, чтобы выполнить код в секции. Первый поток ждет события от второго потока, чтобы вывести промежуточный результат второго потока, схема для кода программы :



Индивидуальные варианты:

3.1 Объекты синхронизации :

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:*

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

Поток **work** должен выполнить следующие действия:

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

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

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

3.2 Объекты синхронизации :

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

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

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

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- запустить поток **work**;

- запустить поток **MultElement**;
- вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

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

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

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

3.3 Объекты синхронизации :

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

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

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- запустить поток **work (** в подвешенном состоянии);
- ввести число k;
- запустить поток **work**
- запустить поток **SumElement**;
- Приостановить работу потока на 50 мс (Sleep)
- Получить от потока **work** сигнал о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов) (*использовать **критическую секцию)***;
- вывести на экран элементы массива (итогового до k элементов);
- известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов) (*использовать **событие***);.
- вывести на экран результат работы потока **SumElement**;
- вывести на экран элементы массива (итогового после k элементов);

Поток **work** должен выполнить следующие действия:

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

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

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

3.4 Объекты синхронизации :

Критическая секция- синхронизация work и потока MultElement (сигнализирует о начале запуска MultElement);
Событие - устанавливает поток ***MultElement для потока main*** (для вывода в ***main результата MultElement*).**

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

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

Поток **work** должен выполнить следующие действия:

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

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

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

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

Поток** main **должен выполнить следующие действия:

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- ввести число k;
- запустить поток **work**;
- запустить поток **SumElement**;
- Получить от потока **work** сигнал о начале суммирования (*использовать **событие***);
- Выводить на экран элементы массива (итогового *до позиции k*);
- известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов массива), *использовать **событие***;
- вывести на экран результат работы потока **SumElement**;
- Выводить на экран элементы массива (итогового *с позиции k*);

Поток **work** должен выполнить следующие действия:

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

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

- ждёт от потока **main** сигнал о начале суммирования (*использовать **событие***);
- вычичлить среднее арифметическое (кодов символов) итогового массива до заданной позиции k;
  или посчитать количество цифр массива до заданной позиции k;
- 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)**.***
Событие2 (с автоматическим сбросом) – в потоке ***work*** устанавливает сигнал ***для потока main*** (для вывода массива)**.**
Критическая секция- синхронизация ***SumElement и*** *потока* ***main (***вывод ***результа SumElement);***

Поток** main **должен выполнить следующие действия:

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- Запросить числа A, K1, K2 (0<K1<K2<размерности массива).
- запустить поток **work**;
- запустить поток **SumElement**;
- получить от потока **work** сообщение о начале работы потока **SumElemen** (*использовать **событие с ручным сбросом).***
- выводить элементы массива (до K2);
- вывести на экран результат работы потока **SumElement**;
- выводить элементы массива (с K2);

Поток **work** должен выполнить следующие действия:

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

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

ждёт от потока **work** сигнал о начале суммирования (*использовать **событие с ручным сбросом***);

- выполнить суммирование элементов от позиции K1 до позиции K2 итогового массива;
- известить(*использовать **критическую секцию)*** поток **main** о выводе результата

3.7 Объекты синхронизации :

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

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

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- запустить поток **work (** в подвешенном состоянии);
- ввести число k;
- запустить поток **work**;
- запустить поток **SumElement**;
- ждёт от потока **work** сигнал о начале суммирования (*использовать **критическую секцию)*** ;
- Выводить на экран элементы массива (итогового);
- известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут готовы к печати k - элементов массива), (*использовать **событие)***.
-
- вывести на экран результат работы потока **SumElement**;

Поток **work** должен выполнить следующие действия:

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

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

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

3.8 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:***

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

Поток **work** должен выполнить следующие действия:

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

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

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

3.9 Объекты синхронизации :

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

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

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

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

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- запустить поток **work (** в подвешенном состоянии);
- ввести числа k,A;
- запустить поток **work** ;
- запустить поток **MultElement**;
- Получить от потока **work** сигнал о начале умножения (*использовать **событие***);
- Выводить на экран элементы массива (итогового);
- известить поток **MultElement** о начале работы (момент запуска произойдёт после того, будут выведены на консоль k элементов массива), *использовать **событие.***
- Вывести на экран результат работы потока **MultElement**;

Поток **work** должен выполнить следующие действия:

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

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

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

3.10 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:***

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

Поток **work** должен выполнить следующие действия:

- запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;

- 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.
- 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!*)**:

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

3.11 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:***

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- запустить поток **work(** в подвешенном состоянии);
- запустить поток **SumElement(** в подвешенном состоянии);
- ввести число k;
- запустить поток **work**;
- запустить поток **SumElement**;
- Получить от потока **work** сигнал о начале суммирования(критическая секция или событие);
- Выводить на экран элементы массива (итогового);
- известить поток **SumElement** о начале суммирования (момент запуска произойдёт после того, будут выведены на консоль k элементов массива) (*использовать **критическую секцию***);
- Выводить на экран элементы массива (итогового);
- Дождаться сигнала потока **SumElement** (*использовать **событие***);
- Вывести на экран результат работы потока **SumElement**

Поток **work** должен выполнить следующие действия:

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

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

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

3.12 Объекты синхронизации :

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

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

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

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

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- ввести число k;
- запустить поток **work**;
- запустить поток **CountElementм**;
- Получить от потока **work** сигнал о начале суммирования (*использовать **событие***);
- Выводить на экран элементы массива (итогового);
- известить поток **CountElement** о начале суммирования (момент запуска произойдёт после того, будут выведены все элементы массива) (*использовать **событие***).
- Выводить на экран элементы массива (итогового до k);
- Дождаться сигнала потока **CountElement** (*использовать **критическую секцию***);
- Вывести на экран результат работы потока **CountElement**

Поток **work** должен выполнить следующие действия:

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

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

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

3.13 Объекты синхронизации :

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

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

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

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

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

Поток **work** должен выполнить следующие действия:

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

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

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

3.14 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:***

- Инициализировать необходимые события и критические секции.
- создать массив, размерность и элементы которого вводятся пользователем с консоли;
- вывести размерность и элементы исходного массива на консоль;
- запустить поток **work(** в подвешенном состоянии);
- создать поток **CountElement(** в подвешенном состоянии);
- Запросить число X.
- запустить поток **CountElement**;
- вывести на экран результат работы потока **CountElement**;

Поток **work** должен выполнить следующие действия:

- запросить у пользователя временной интервал, требуемый для отдыха после подготовки одного элемента в массиве;

- Поиск в массиве положительных элементов кратных 5 (разместить их в массиве слева, остальные элементы массива - справа). Элемент - X ввести в потоке main.
- выводить на экран поэлементно элементы массива (итогового);
- после каждого готового элемента отдыхать в течение заданного интервала времени;
- известить поток **CountElement** о начале работы (момент запуска произойдёт после того, будет сформирован итоговый массив (*использовать **критическую секцию**)* .

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

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

Лабораторная работа №4.
Сдать до 8.04

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, 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 сообщение "A", сообщение "B" 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**
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения(*"A" или "B"*) , 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**
- передачу сообщений реализовать с помощью **событий**
- принимает сообщение от процесса **Writer**;
- выводит на консоль сообщение *"A" или "B"* ;
- передает сообщение от **Writer** - 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 "А" , сообщение "В", 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**
- 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**
- 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 "А"(два события) , сообщение "В" (два события), сообщение "С"(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**
- 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**
- 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 “A” , сообщение “B”, и автоматическое событие - конец сеанса для процессов **Reader и Writer**.

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

Процесс **Administrator**:

- 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**
- передачу сообщений реализовать с помощью событий с ручным сбросом
- апрашивает 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**
- 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 «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**:

- 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**
- 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 *«A»* 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**
- 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 *«B» »* 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 «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 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**
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее из *"A" или "B",* 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**
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения *«C» , «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 «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**:

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

Процесс **Child**:

- синхронизировать работу процессов **Child** с помощью **семафора**
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее *«C» или «D»* и передает их (по одному) процессу Boss;
- 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**:

- Инициализация объектов синхронизации;

- запрашивает у пользователя количество процессов **Parent** и количество(<=4) процессов **Child**, которые он должен запустить.
- запрашивает кол-во сообщений, отправленных (полученных) **Parent и Child**
- запускает заданное количество процессов **Parent, Child**;
- запрашивает с консоли (можно автоматически получив сообщ. A - отправить сообщение C, получив сообщ. B - 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**
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее *«A» или «B»* 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**
- передачу сообщений реализовать с помощью **событий**
- получает сообщения, состоящее *«C» или «D»* от процесса Boss и выводит его на консоль;
- передает сообщение о завершении сеанса процессу **Boss**
- 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**:

- Инициализация объектов синхронизации;
- запрашивает у пользователя количество процессов **Employee**, которые он должен запустить;
- запрашивает у пользователя количество сообщений для процессов **Employee**, которые он должен отправить;
- запускает заданное количество процессов **Employee**;
- 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.
- выводит его на консоль сообщение об окончании работы очередного процесса **Employee**.
- завершает свою работу.

Процесс **Employee**:

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

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

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 **Reader и Writer**, которые он должен запустить;
- запрашивает у пользователя кол-во отправленных сообщений для процесса **Writer**. Кол-во принятых сообщений для процесса **Reader** вычислить **(общее количество отправленных и принятых сообщений должно совпадать)**;
- 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 **мьютексов**
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщение, 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**
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения от процесса **Writer**;
- выводит на консоль сообщения;
- передает сообщение о завершении сеанса процессу **Administrator**;
- завершает свою работу.

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

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

Процесс **Boss**:

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

Процесс **Parent**:

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

Процесс **Child**:

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

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

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

Процесс **Boss**:

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

Процесс **Parent**:

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

Процесс **Child**:

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

4.13. Написать программы для консольного процесса **Administrator** и консольных процессов **Reader и Writer**. *Для моделирования передачи сообщений ввести специальные события(***с автоматическим сбросом)***, которые*

обозначают сообщение «A» , «B», «C» , «D» и конец сеанса для процессов **Reader и Writer**. *Для сообщений «C» и «D»* **.**

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

Процесс **Administrator**:

- 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 **мьютекса**
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее из *“A” или “B”,* 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**
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения *«C» , «D»* от процесса **Administrator**;
- выводит на консоль сообщения;
- передает сообщение о завершении сеанса процессу **Administrator**;
- завершает свою работу.

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

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

Процесс **Administrator**:

- 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**
- 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**
- 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.

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

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

Процесс **Administrator**:

- 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 соответвующее (например, *"A"* соответствует *"C"*) сообщение 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**
- передачу сообщений реализовать с помощью **событий**
- запрашивает с консоли сообщения, состоящее (*"A" или "B")* 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**
- передачу сообщений реализовать с помощью **событий**
- принимает сообщения (*"C" или "D")* от процесса **Administrator**;
- выводит на консоль сообщение;
- передает сообщение о завершении сеанса процессу **Administrator**;

завершает свою работу.

Лабораторная работа №5

Сдать до 23.04.

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

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

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.

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.

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.

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 анонимному каналу.

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.

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.

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.

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 анонимному каналу.

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

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

Процесс-Alfavit, который выполняет следующие действия.

- Получает массив символов по анонимному каналу от процесса-сервера.
- Определяет символы, принадлежащие латинскому алфавиту и передает их по анонимному каналу процессу-серверу.
- Выводит оба массива на консоль.
- Элементы массива передаются поэлементно.

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

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

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

Процесс-Figure, который выполняет следующие действия.

- Получает размер массива символов по анонимному каналу от процесса-сервера.
- Генерирует элементы массива
- Определяет цифры и передает их по анонимному каналу процессу-серверу.
- Элементы массива предаются по одному.

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

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

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

Процесс-Palindrom, который выполняет следующие действия.

- Получает размер массива символов и элементы массива по анонимному каналу от процесса-сервера.
- Запрашивает символ-разделитель для лексем в строке
- Находит среди лексем палиндромы в строке и передает палиндромы по анонимному каналу процессу-серверу.
- Выводит полученные палиндромы на консоль.

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

Одновременно сообщение может передаваться только одним из процессов.

Процесс- Server, который выполняет следующие действия:

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

Процесс- Consume, который выполняет следующие действия.

- Получает размер массива символов и элементы массива по анонимному каналу от процесса-сервера.

- Выводит полученные данные на консоль.
- Количество чисел, которые должны быть потреблены, запрашивается с консоли.
- Индексы потреблённых чисел генерируются случайно
- Выводит оставшиеся числа на консоль и передает её по анонимному каналу процессу-серверу.
- Закончить работу после нажатия клавиши - “Q”.
-

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

Одновременно сообщение может передаваться только одним из процессов.

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

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

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

Одновременно сообщение может передаваться только одним из процессов.

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

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

Дополнительное (штрафное) задание:

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

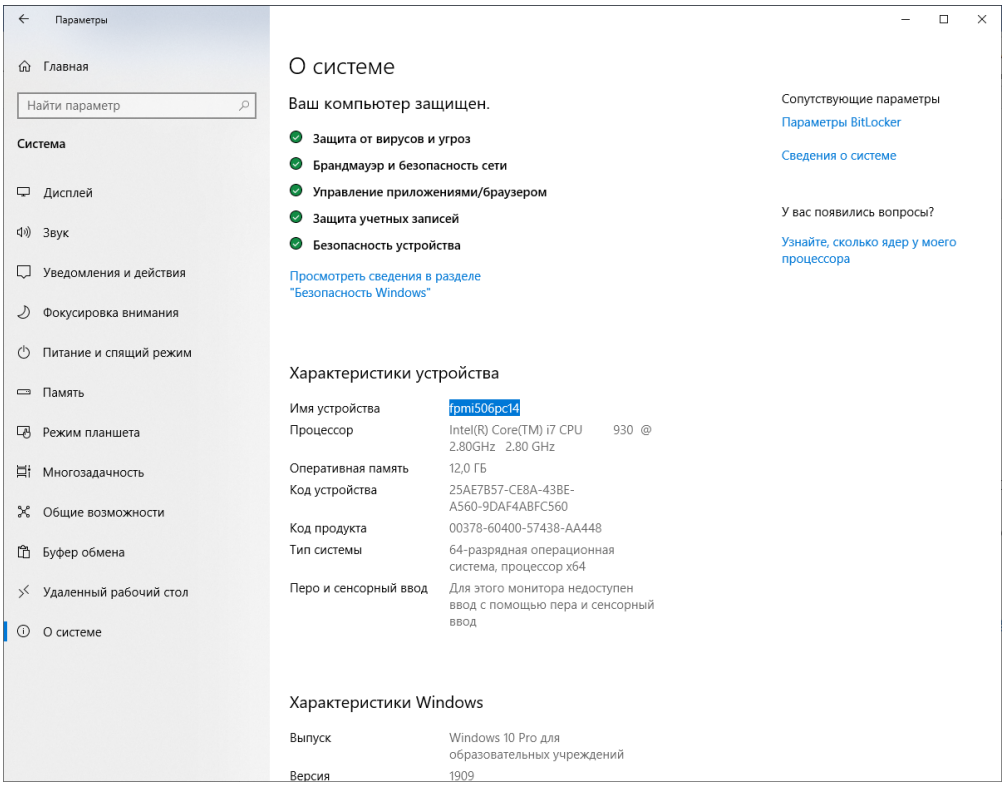
Лабораторная работа №6

Сдать до 8.05.

Тема: "Обмен данными *по именованному* каналу с сервером".

переделать лаб. №5, изменить для именованных каналов.
Объекты синхронизации не использовать.
Процессы должны работать на разных компьютерах!
Имя канала на сервере : . (точка - имя сервера) в имени канала (например: `\\\\.\\pipe\\demo_pipe`).
Имя компьютера, на котором ваше приложение подключается к каналу: fpmi506pc14 (например)

(например, имя канала для подключения к серверу: \\\\ fpmi506pc14\\pipe\\demo_pipe), найти в настройках имя сервера:



Лабораторная работа №7

Сдать до 17.05.

Тема: "Разработка монитора для синхронизации параллельного доступа к кольцевой очереди(стеку)" (варианты смотреть ниже).

1. Написать монитор для управления параллельным доступом к очереди или стеку (***Согласно вариантам - см. ниже***):

Интерфейс монитора для очереди:

```
class MonitorQueuer
{
        MonitorQueue(int nSize);            // nSize - размер очереди
        ~MonitorQueue();

        void  AddHead (type & nElement); // добавить элемент в «голову» очереди
        char  RemoveTail();                 // удалить элемент из «хвоста» очереди

};
```

Интерфейс монитора для стека:

```
class MonitorStack
{
        MonitorStack(int nSize);            // nSize - размер стека
        ~MonitorStack();

        void  Push(type & nElement);        // добавить элемент в стек
        char  Pop();        // удалить элемент из стека
};
```

2. Кольцевая очередь (стек) реализуется массивом. Размер кольцевой очереди задается в конструкторе.
3. Если поток обращается к методу Add… (Push) монитора, а очередь (стек) оказывается в этот момент полной, то монитор переводит поток в состояние ожидания до тех пор, пока в очереди (в стеке) не окажутся свободные ячейки.

5. Если поток обращается к методу Remove (Pop) монитора, а очередь(стек) оказывается в этот момент пустой, то монитор переводит поток в состояние ожидания до тех пор, пока в очереди (в стеке) не окажется новый элемент.

Дополнительные требования:

Для тестирования монитора написать программу для консольного процесса, который состоит из потока main и нескольких потоков consumer и producer.

Поток **main** должен выполнять следующие действия:

- создать монитор очереди (стека), размер которой вводится пользователем с клавиатуры;
- ввести с клавиатуры количество потоков producer и количество потоков consumer, которые он должен запустить;
- для каждого из потоков producer и consumer ввести количество продуктов (тип массива в индивидуальном варианте), которые эти потоки должны соответственно произвести и потребить;
- создать требуемое количество потоков producer и consumer;
- передать в качестве параметров каждому потоку producer кол-во продуктов (целое число и сам продукт (элемент), который поток должен произвести (записать в очередь\стек);
- передать в качестве параметра каждому потоку consumer положительное целое число, которое обозначает количество продуктов, которое поток должен потребить (удалить из очереди\стека);
- завершить свою работу после окончания работы всех потоков producer и consumer.

Поток **producer** должен выполнять следующие действия:

- произвести (записать в очередь\стек) требуемое количество экземпляров продукта ;
- после добавления очередного числа в очередь(стек) выводить на консоль сообщение следующего вида:
  "Произведен элемент (продукт) N ", где N ,записано в очередь\стек.

Поток **consumer** должен выполнять следующие действия:

- потребить (удалить из очереди\стека) заданное количество продуктов;
- при извлечении очередного символа из очереди (стека), выводить на консоль сообщение следующего вида:
  "\tУпотреблен элемент (продукт) N ", где N – элемент (продукт) который извлечен из очереди\стека.

Варианты индивидуальных заданий:

5.1. Доступ к стеку. Элемент (продукт) - символ char.
5.2. Доступ к очереди. Элемент (продукт) - int.
5.3. Доступ к стеку. Элемент (продукт) -doble.
5.4. Доступ к очереди. Элемент (продукт) - float.
5.5. Доступ к стеку. Элемент (продукт) - short.
5.6. Доступ к очереди. Элемент (продукт) - __int8.
5.7. Доступ к стеку. Элемент (продукт) -__int32.
5.8. Доступ к очереди. Элемент (продукт) -unsigneg int.
5.9. Доступ к стеку. Элемент (продукт) -__int16.
5.10. Доступ к очереди. Элемент (продукт) -__int64.
5.11. Доступ к стеку. Элемент (продукт) - long.
5.12. Доступ к очереди. Элемент (продукт) - unsigneg short.
5.13. Доступ к стеку. Элемент (продукт) - символ char.
5.14. Доступ к очереди. Элемент (продукт) - __int8.

Лабораторная работа №8 (на 9-10 баллов)
Тема: Классические задачи.

Номер на выбор:

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