Выполнила: Белоусова Е., ИП-911

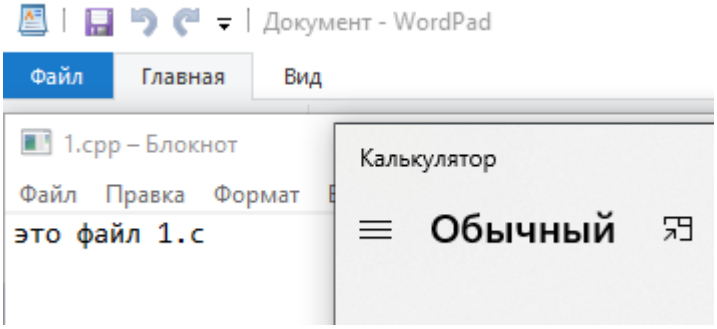
Цель: познакомиться с принципами синхронизации потоков разных процессов
с помощью объектов ядра.

Задание:

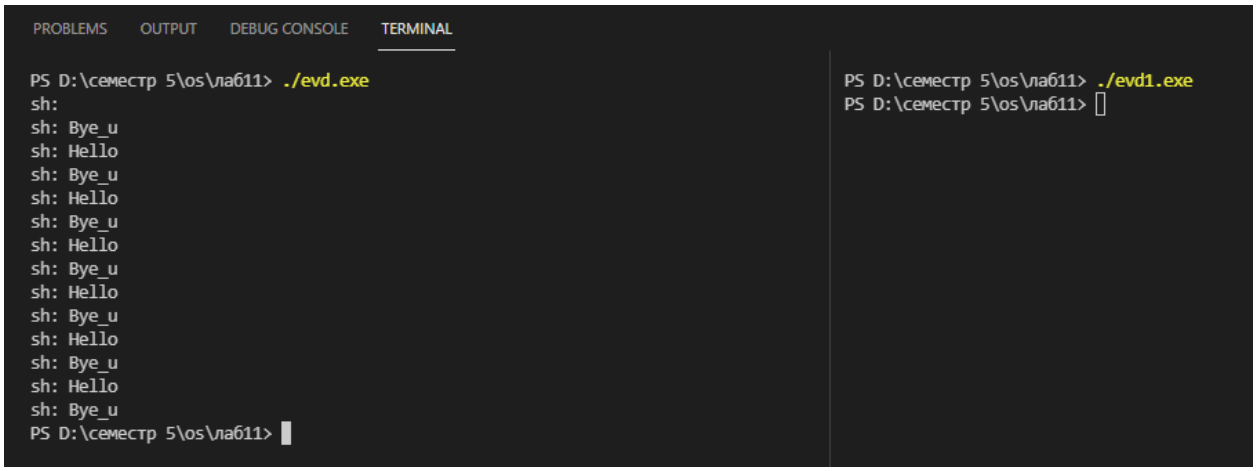
• протестировать программы Лекции11.
• синхронизировать потоки разных процессов с помощью мьютексов и семафоров.

Описание работы программы

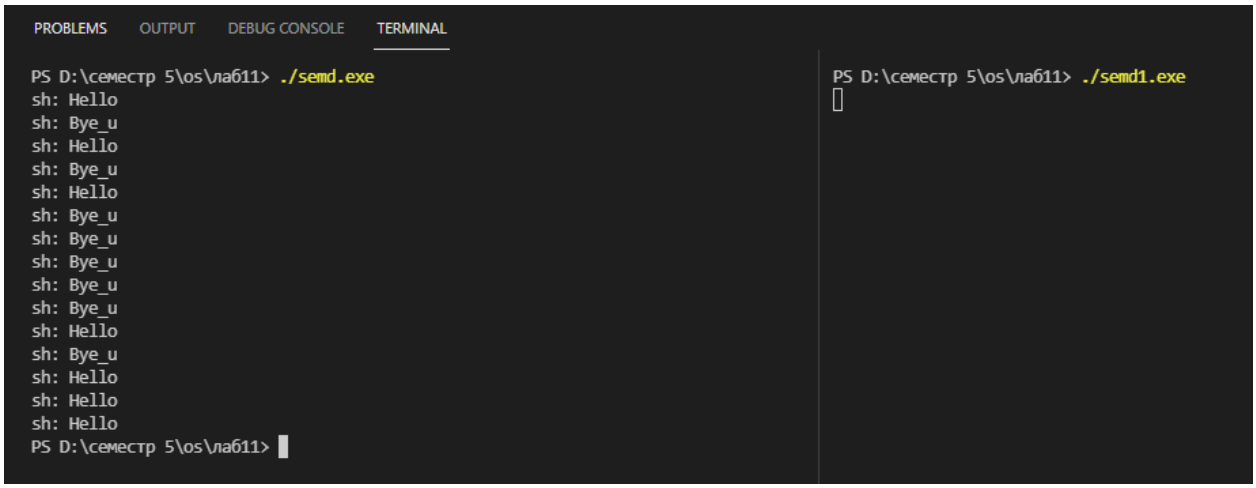
Протестируем программу барьерной синхронизации. Объявим массивы типа STARTUPINFO (используется с функцией CreateProcess, чтобы определить оконный терминал, рабочий стол, стандартный дескриптор и внешний вид основного окна для нового процесса), PROCESS_INFORMATION (заполняется функцией CreateProcess с информацией о недавно созданном процессе и его первичном потоке), HANDLE (дескрипторы). Программа открывает программы notepad с файлом 1.cpp, wordpad и калькулятор. После завершения работы notepad, независимо от завершения работы калькулятора и wordpad, программа останавливается. Если заменить wordpad и калькулятор на другие программы (mspaint, notepad++ или собственные), то тогда основная программа будет дожидаться завершения выполнения всех программ.



Синхронизация потоков разных процессов с помощью событий:



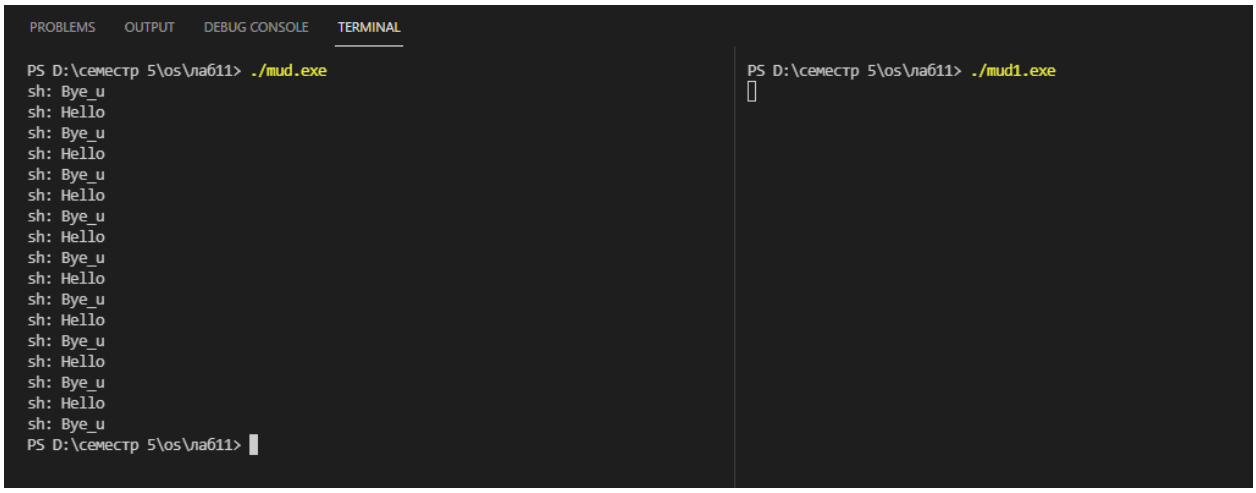
Синхронизация потоков разных процессов с помощью семафоров:




```
HANDLE OpenSemaphore(
        DWORD fdwAccess,
        BOOL bInheritHandle,
        PCTSTR pszName
        );
```

SEMAPHORE_ALL_ACCESS требует полного доступа к семафору
SEMAPHORE_MODIFY_STATE позволяет использовать функцию ReleaseSemaphore
SYNCHRONIZE позволяет использовать синхронизацию семафоров

Синхронизация потоков разных процессов с помощью мьютексов:



OpenMutex - функция открывает существующий мьютекс по его имени. Возвращаемым значением является дескриптор мьютекса.

HANDLE OpenMutex(//создание дескриптора для существующего именованного мьютекса

DWORD dwDesiredAccess, //Уровень доступа к мьютексу

BOOL bInheritHandle, //Возможность наследования

LPCTSTR lpName //Имя открываемого мьютекса

);

Листинг

//libevd1.cpp

```
#include <windows.h>


#pragma data_seg(".M_SH")
extern __declspec(dllexport)
char sh[6] = {'\0'};
#pragma data_seg()
#pragma comment(linker, "/SECTION:.M_SH,RWS")
```

//semd.cpp

```
#include <windows.h>
#include <stdio.h>
#pragma comment(lib, "libevd1")
HANDLE hSemaphore;
extern __declspec(dllimport)
char sh[6];
```

```
int main(void)
{
   hSemaphore = CreateSemaphore(NULL, 1, 2, "MySemaphore");
   while(1)
   {
      WaitForSingleObject(hSemaphore, INFINITE);
      printf("sh: %s\n", sh);
      ReleaseSemaphore(hSemaphore, 1, NULL);
      Sleep(100);
   }
   CloseHandle(hSemaphore);
   return 0;
}
//semd1.cpp
#include <windows.h>
#include <stdio.h>
#pragma comment(lib, "libevd1")
HANDLE hSemaphore;
extern __declspec(dllimport)
char sh[6];
int main(void)
{
   int counter = 0;
   hSemaphore = OpenSemaphore(SEMAPHORE_ALL_ACCESS, FALSE, "MySemaphore");
   while(1)
   {
      WaitForSingleObject(hSemaphore, INFINITE);
      if(counter%2)
      {
         sh[0] = 'H'; sh[1]='e'; sh[2]='l'; sh[3]='l'; sh[4]='o'; sh[5]='\0';
      }
      else
```

```
      {
         sh[0] = 'B'; sh[1] = 'y'; sh[2] = 'e'; sh[3] = '_'; sh[4] = 'u'; sh[5] = '\0';
      }
      ReleaseSemaphore(hSemaphore, 1, NULL);
      counter++;
      Sleep(100);
   }
}
//mud.cpp
#include <windows.h>
#include <stdio.h>
#pragma comment(lib, "libevd1")
HANDLE hMutex;
extern __declspec(dllimport)
char sh[6];
int main(void)
{
   hMutex = CreateMutex(NULL, FALSE, "MyMutex");
   while(1)
   {
      WaitForSingleObject(hMutex, INFINITE);
      printf("sh: %s\n", sh);
      ReleaseMutex(hMutex);
      Sleep(100);
   }
   CloseHandle(hMutex);
   return 0;
}
//mud1.cpp
#include <windows.h>
#include <stdio.h>
#pragma comment(lib, "libevd1")
```

```
HANDLE hMutex;
extern __declspec(dllimport)
char sh[6];
int main(void)
{
   int counter = 0;
   hMutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, "MyMutex"); //SYNCHRONIZE
   while(1)
   {
      WaitForSingleObject(hMutex, INFINITE);
      if(counter%2)
      {
         sh[0] = 'H'; sh[1]='e'; sh[2]='l'; sh[3]='l'; sh[4]='o'; sh[5]='\0';
      }
      else
      {
         sh[0] = 'B'; sh[1] = 'y'; sh[2] = 'e'; sh[3] = '_'; sh[4] = 'u'; sh[5] = '\0';
      }
      ReleaseMutex(hMutex);
      counter++;
      Sleep(100);
   }
}
```